МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Тюменский государственный университет»

Методические указания по выполнению лабораторных работ по дисциплине «ЯЗЫКИ ПРОГРАММИРОВАНИЯ»

для студентов среднего профессионального образования
10.02.05 Обеспечение информационной безопасности
автоматизированных систем

Тюмень, 2024

СОДЕРЖАНИЕ

Методические указания к лабораторным работам

Лабораторная работа №1 Введение в языки программирования. Общие принципы построения и использования языков программирования. Язык программирования C++

Цель работы: Знакомство с основами создания консольных приложений.

Теоретическая часть

В оконных приложениях используется Windows-интерфейс GUI (Graphical User Interface– графический интерфейс пользователя).

Существует три основных стиля пользовательских интерфейсов:

- однооконный интерфейс (SDI), например, реализованный в WordPad (в WordPad можно открыть только один документ; чтобы открыть другой документ, необходимо открыть первый);
- многооконный интерфейс (MDI), например, реализованный в Microsoft Excel (позволяет отображать несколько документов сразу, при этом каждый документ отображается в отдельном окне);
- интерфейс проводника – это одно окно с двумя панелями или областями; обычно слева представлена иерархия объектов, как в проводнике Microsoft Windows.

Выбор стиля интерфейса зависит от назначения приложения. Основой графического интерфейса пользователя является форма.

Форма – часть пространства экрана, обычно прямоугольной формы, которую можно использовать для представления сведений пользователю и для получения сведений от него. Форма является основной движущей силой взаимодействия с пользователем.

Термин "форма" можно считать синонимом окна – окна приложения, диалогового окна Значительная часть пользовательского интерфейса приложений реализуется именно с применением диалоговых окон. Это окна, предназначенные для открытия, сохранения, печати и закрытия документов, окна отображения и настройки всевозможных параметров и т.д.

Диалоговые окна принято делить на модальные и немодальные окна. Когда приложение открывает на экране модальное окно, его работа будет приостановлена до тех пор, пока пользователь не закроет это окно. Что же касается немодальных окон, то они работают одновременно с главным окном открывшего их приложения.

Всякий начинающий программист после изучения некоторых основ языка программирования, будь то C++, Pascal, Assembler, обязательно хочет написать свою первую программу. По устоявшейся традиции этой программой в большинстве случаев является приложение, выводяшее на экране монитора надпись "Hello world!" Итак, приступим к написанию первой программы! Первым шагом в написании программы является запуск самой среды программирования C++ Builder (Исходный текст программы приведен для версии 6.0). Затем выберем **File\New\Other...** Вы попадете в окно выбора типа вашего проекта (Это может быть как приложение под платформы DOS\Windows, либо DLL библиотеки или иные компоненты). Внешний вид открывшегося окна показан на рисунке ниже:

Форму можно полностью создать с помощью редактора кода. Однако, для создания и изменения форм проще использовать конструктор Windows Forms. Для создания новой формы с помощью конструктора форм необходимо выполнить команду

File→New→Project→WindowsFormsApplication.

На форме размещаются элементы управления.

Элемент управления (или управляющий элемент, программный элемент, компонент Windows-форм, или контрол) – это объект на форме, который придает форме новые функциональные возможности и формирует пользовательский интерфейс.

Первый из них – использовать Windows Forms, реализующий графический интерфейс пользователя и входящий в Microsoft.NET Framework. Данный подход упрощает доступ к интерфейсным элементам Microsoft Windows благодаря созданию обёртки для Win32 API в управляемом коде. Если изложить свои мысли в более понятной форме, то данный подход очень схож с построением консольного приложения, но чуть более сложен т.к. использует формы.

Второй способ основан на использовании Microsoft Foundation Classes (MFC), библиотеке, которая берет на себя заботу о создании каркаса приложения. В отличие от первого, MFC «из коробки» использует патерн MVC (Model-View-Controller). Данный подход сложнее первого, но опираясь на него можно запросто создать каркас для весьма интересных приложений, к примеру, текстового редактора или использовать компонент Ribbon и сделать меню как в небезызвестном MS Office 2010.

Создание приложения в MS Visual Studio

Давайте создадим новое приложение: File->New->Project. В появившемся окне как на рисунке выше найдите и выберите Windows Forms Application, далее укажите название (app1) и расположение нового проекта и подтвердите его создание нажатием кнопки «OK».

Оборудование и материалы: для выполнения данного практического занятия необходим компьютер с установленной операционной системой Windows 7 и программными продуктами: Visual Studio.

Указания по технике безопасности: к выполнению практических работ допускаются студенты, ознакомившиеся с правилами работы в лаборатории, прошедшие инструктаж безопасности.

Задания: для выполнения практической работы необходимо выполнить следующее:

1. Изучить рекомендуемую литературу.
2. Выполнить лабораторную работу.
3. Ответить на контрольные вопросы.
4. Оформить отчет.

Содержание отчета: отчет по практической работе должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman, интервал – полуторный, поля левое – 3 см., правое – 1,5 см., верхнее и нижнее – 2 см. Абзацный отступ – 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой практической работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись (Рисунок 1 – Создание подсистемы) и ссылку на него в тексте.

Контрольные вопросы:

1. Понятие Graphical User Interface.
2. Три основных стиля интерфейса.

3. Создание WindowsFormsApplication.
4. Понятие формы.
5. Модальные и немодальные окна.

Список литературы, рекомендуемый к использованию по данной теме:

1. Лафоре Р. Объектно-ориентированное программирование в С++. – М. – СПб.:
Питер, 2008 г. – 922 с.
2. Юров В. Ассемблер. Учебник для вузов. – СПб.: Питер, 2008.- 624 с.
3. Юров В. Ассемблер. Практикум. – СПб.: Питер, 2008. - 356 с.
4. Бен-Ари М. Языки программирования: Практический сравнительный анализ: Учебник. – М.: Мир. – 2007.
5. Молчанов А.Ю. Системное программное обеспечение: Лабораторный практикум.
– М.; СПб; Нижн.Новг.; Минск, 2009.

Лабораторная работа №2. Условные операторы if и switsh

Цель работы: познакомиться с работой условного оператора и оператора перехода.

Изучить оператор выбора варианта. Научиться применять их при составлении программ.

Теоретическая часть Условный оператор if

Рассматриваемая группа операторов позволяет организовать ветвление в программе.

Часто, например, необходимо в зависимости от того или иного результата реализовать одну либо другую группу операторов (инструкций). В языке СИ для этих целей используются операторы if (если) – else (иначе), switch (переключатель) и goto (идти к).

Оператор if имеет вид: if (проверка условия)
инструкция1; else инструкция2; Например:
if (a>b) z=a; else z=b;

Необходимо обратить внимание на точку с запятой после z=a. Здесь она обязательна, поскольку за if должна следовать инструкция, которая всегда заканчивается точкой с запятой.

В операторе if слово else может отсутствовать. В этом случае, если условие в скобках принимает истинное значение, выполняется инструкция 1, а если ложное, то инструкция 1 пропускается и управление передается следующему оператору по тексту программы. Например: if (num>10) num=2*num; printf("%d\n",num);

Оператор вывода будет выполняться всегда, а оператор присваивания только в том случае, если условие будет истинным.

Операции отношения, используемые для сравнения:

< - меньше
<= - меньше или равно
= = - равно
>= - больше или равно
> - больше
!= - не равно

Не следует путать операцию отношения "==" с операцией присваивания "=". Оператор безусловного перехода

Его можно представить в следующей форме: goto метка; Метка – это любой идентификатор.

Например: goto a2;

Оператор goto указывает, что выполнение программы необходимо продолжить, начиная с инструкции, перед которой записана метка. В программе обязательно должна быть строка, где указана метка, поставлено двоеточие и записана инструкция, к которой должен выполняться переход.

Например: a2: k=5;

Метки в программе описывать не нужно. Применение оператора безусловного перехода в языке СИ является нежелательным, так как он нарушает структурную наглядность программы.

Оператор выбора switch

Оператор switch позволяет выбрать одну из нескольких альтернатив. Он записывается в следующем виде:

```
switch (выражение)
{case константа1, вариант 1; break;
. . .
case константа n, вариант n; break; default: вариант n+1; break;}
```

В операторе switch вычисляется целое выражение в скобках (его называют селектором), и его значение сравнивается со всеми константами. При совпадении выполняется соответствующий вариант (одна или несколько инструкций). Все константы в записи оператора должны быть различными. Вариант с ключевым словом default (прочие) реализуется, если ни один другой не подошел (если слово default отсутствует, а все результаты сравнения отрицательны, то ни один вариант не выполняется. Для прекращения последующих проверок после успешного выбора некоторого варианта используется оператор break, обеспечивающий немедленный выход из оператора switch. Например:

```
#include<stdio.h> main( )
{
char y; scanf("%c",&y); switch(y)
{    case
‘1’:
printf("Ветвь 1\n"); break;
case ‘2’: case ‘3’:
printf("Ветвь 2 или 3\n"); break; default:
printf("Ветви 1,2,3 не работают\n");
}
}
```

Оператор scanf вводит переменную y. Ее значение в операторе switch сравнивается со всеми константами операторов case. Если ввести символ ‘1’, то на экране появится строка: Ветвь 1

по оператору break произойдет выход из переключателя switch, и программа завершит

свою работу. Если ввести символы ‘2’ или ‘3’, то на экран будет выведена строка: Ветвь 2 или 3

При вводе любого другого символа управление перейдет к ключевому слову default и на экране появится строка: Ветви 1,2,3 не работают.

Оборудование и материалы: для выполнения данного практического занятия необходим компьютер с установленной операционной системой Windows 7 и программными продуктами: Visual Studio.

Указания по технике безопасности:

К выполнению практических занятий допускаются студенты, ознакомившиеся с правилами работы в лаборатории, прошедшие инструктаж безопасности.

Задания: для выполнения практической работы необходимо выполнить следующее:

1. Изучить рекомендуемую литературу.
2. Выполнить практическую работу.
3. Ответить на контрольные вопросы.
4. Оформить отчет.

Содержание отчета: отчет по практическому занятию должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman, интервал – полуторный, поля левое – 3 см., правое – 1,5 см., верхнее и нижнее – 2 см. Абзацный отступ – 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой лабораторной работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись (Рисунок 1 – Создание подсистемы) и ссылку на него в тексте.

Контрольные вопросы:

1. Синтаксис условного оператора if.
2. Присутствие else в операторе if.
3. Какие операции отношения используются и форма их записи.
4. Синтаксис оператора выбора switch.

Список литературы, рекомендуемый к использованию по данной теме:

1. Лафоре Р. Объектно-ориентированное программирование в C++. – М. – СПб.:
Питер, 2008 г. – 922 с.
2. Юров В. Ассемблер. Учебник для вузов. – СПб.: Питер, 2008.- 624 с.
3. Юров В. Ассемблер. Практикум. – СПб.: Питер, 2008. - 356 с.
4. Бен-Ари М. Языки программирования: Практический сравнительный анализ: Учебник. – М.: Мир. – 2007.

Лабораторная работа №3. Программирование с использованием одномерных массивов

Цель работы: Знакомство с оператором цикла for

Теоретическая часть

Цикл — многократное повторение последовательности действий по некоторому условию. Известны три типа циклических алгоритмических структур: цикл с пред-условием, цикл с постусловием и цикл с параметром.

Оператор цикла for

Описание: for (выражение 1; выражение 2; выражение 3) оператор; Действие:

В круглых скобках содержится три выражения. Первое из них служит для инициализации счетчика. Она осуществляется только один раз – когда цикл for начинает выполняться. Второе выражение необходимо для проверки условия, которая осуществляется перед каждым возможным выполнением тела цикла. Когда выражение становится ложным, цикл завершается. Третье выражение вычисляется в конце каждого выполнения тела цикла, происходит приращение числа на шаг.

Комментарий: в операторе цикла for точка с запятой после закрывающейся круглой скобки не ставится. Любое из трех или все три выражения в операторе могут отсутствовать, однако разделяющие их точки с запятыми опускать нельзя. Если отсутствует выражение 2, имеем бесконечный цикл. Например:for (scanf("%d",&p);;p++) оператор;

В языке СИ предусмотрены две нетрадиционные операции: (++) – для увеличения на единицу и (--) – для уменьшения на единицу значения операнда. Операции ++ и -- можно записывать как перед операндом, так и после него. В первом случае (++n или --n) значение операнда (n) изменяется перед его использованием в соответствующем выражении, а во втором (n++ или n--) – после его использования.

Если отсутствуют выражения 1 и 3, цикл становится эквивалентным while.

Например:for (;a<20;) оператор;

Каждое из выражений может состоять из нескольких выражений, объединенных операцией "запятая". Например: for(i=0, j=1; i<100; i++, j++) a[i]=b[j]; Тело цикла заключается в фигурные скобки, если в нем более одного оператора.

Пример:

```
/*демонстрация цикла for*/ #include <stdio.h> main()
{int i,j=1,k;
for (i=1;i<=3;i++) printf("Минск\t");
/*В цикле for три раза выполняется функция вывода*/
/*Здесь i-управляющая переменная цикла*/ printf("\nУкажите число повторений цикла\n"); scanf("%d",&k); for (i=1;i<=k;i++)
{j*=i; printf("%d",j);}
/*Здесь две инструкции (более одной), поэтому они заключаются в фигурные скобки*/
j=i; printf("\n");
/*Переменной j присваивается значение 1 и осуществляется перевод курсора*/ /*В следующем цикле for выполняются те же действия, что и в предыдущем*/ for (i=1;i<=k;i ++) printf("%d ", j*=i); }
```

Результаты выполнения программы, следующие: Минск

Минск Минск

Укажите число повторений цикла; 5 1 2 6 24 120

1 2 6 24 120

Оборудование и материалы: для выполнения данного практического занятия необходим компьютер с установленной операционной системой Windows 7 и программными продуктами: Visual Studio.

Указания по технике безопасности к выполнению практических занятий допускаются студенты, ознакомившиеся с правилами работы в лаборатории, прошедшие инструктаж безопасности.

Задания: для выполнения практической работы необходимо выполнить следующее:

1. Изучить рекомендуемую литературу.
2. Выполнить практическую работу.
3. Ответить на контрольные вопросы.
4. Оформить отчет.

Содержание отчета: отчет по практическому занятию должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman, интервал – полуторный, поля левое – 3 см., правое – 1,5 см., верхнее и нижнее – 2 см. Абзацный отступ – 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой практической работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись (Рисунок 1 – Создание подсистемы) и ссылку на него в тексте. **Контрольные вопросы:**

1. Понятие цикл.
2. Синтаксис оператора цикла for.
3. Когда применяется оператор цикла for.
4. Отличия от других оператор цикла. **Список литературы, рекомендуемый к использованию по данной теме:**

1. Лафоре Р. Объектно-ориентированное программирование в C++. – М. – СПб.:
Питер, 2008 г. – 922 с.
2. Юров В. Ассемблер. Учебник для вузов. – СПб.: Питер, 2008.- 624 с.
3. Юров В. Ассемблер. Практикум. – СПб.: Питер, 2008. - 356 с.
4. Бен-Ари М. Языки программирования: Практический сравнительный анализ: Учебник. – М.: Мир. – 2007. 5. Молчанов А.Ю. Системное программное обеспечение: Лабораторный практикум. – М.; СПб; Нижн.Новг.; Минск, 2009.

Лабораторная работа № 4. Программирование с использованием многомерных массивов

Цель работы: Ознакомление с оператором цикла while.

Теоретическая часть

Цикл — многократное повторение последовательности действий по некоторому условию. Известны три типа циклических алгоритмических структур: цикл с пред-условием, цикл с постусловием и цикл с параметром.

Описание: while (выражение) оператор; Действие:

Выполняется оператор до тех пор, пока значение выражения в скобках истинно. Проверка значения выражения происходит перед каждым выполнением оператора. Когда значение выражения ложно, цикл while заканчивается. Если выражение ложно с самого начала, оператор не выполняется ни разу.

Комментарий:

Следует заметить, что после ключевого слова while и выражения, заключенного в круглые скобки, точка с запятой не ставится.

Оператор иногда называется телом цикла. В теле цикла должны выполняться действия, в результате которых меняется значение управляющего выражения. В противном случае можем получить бесконечный цикл.

Пример:

/*Демонстрация цикла while*/ #include
<stdio.h> main( ) { int i=1 while (getchar()!='R')
i++;
/*оператор getchar() вводит любой символ с клавиатуры*/ printf("Символ R %d-й",i);
}

Приведенная программа позволяет определить порядковый номер первой введенной буквы R в последовательности символов. Она показывает использование цикла while, в теле которого всего одна инструкция (i++ - увеличение значения целого числа i на единицу). Если запустить эту программу на выполнение и ввести последовательность символов, например: abFk!Rgm, то на экране появится строка: Символ R 6-й.

Оператор циклado-while

Описание: do оператор while (выражение);

Действие: В операторе do-while тело цикла выполняется по крайней мере один раз. Тело цикла будет выполняться до тех пор, пока выражение в скобках не примет ложное значение. Если оно ложно при входе в цикл, то его тело выполняется ровно один раз.

Комментарий: после слова while и выражения, заключенного в скобки, ставится точка с запятой. Если в теле цикла содержится более одной инструкции, то операторы цикла заключаются в фигурные скобки.

Пример:

/*Демонстрация цикла do-while */ #include <stdio.h> main()
{
int i=0; /*i=0, анеединице*/ do i++; while (getchar()!='R');
printf("Символ R %d-й",i); } Программа, представленная выше, теперь написана с циклом do-while.

Оборудование и материалы: для выполнения данного практического занятия необходим компьютер с установленной операционной системой Windows 7 и программными продуктами: Visual Studio.

Указания по технике безопасности: к выполнению практических занятий допускаются студенты, ознакомившиеся с правилами работы в лаборатории, прошедшие инструктаж безопасности.

Задания: для выполнения практической работы необходимо выполнить следующее:

1. Изучить рекомендуемую литературу.
2. Выполнить практическую работу.
3. Ответить на контрольные вопросы.
4. Оформить отчет.

Содержание отчета: отчет по практическому занятию должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman, интервал – полуторный, поля левое – 3 см., правое – 1,5 см., верхнее и нижнее – 2 см. Абзацный отступ – 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой практической работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись (Рисунок 1 – Создание подсистемы) и ссылку на него в тексте. **Контрольные вопросы:**

1. Понятие цикл.
2. Синтаксис оператора цикла while.
3. Синтаксис оператора цикла do while.
4. Отличия от оператора цикла for. **Список литературы, рекомендуемый к использованию по данной теме:**

1. Лафоре Р. Объектно-ориентированное программирование в C++. – М. – СПб.:
Питер, 2008 г. – 922 с.
2. Юров В. Ассемблер. Учебник для вузов. – СПб.: Питер, 2008.- 624 с.
3. Юров В. Ассемблер. Практикум. – СПб.: Питер, 2008. - 356 с.
4. Бен-Ари М. Языки программирования: Практический сравнительный анализ: Учебник. – М.: Мир. – 2007.
5. Молчанов А.Ю. Системное программное обеспечение: Лабораторный практикум. – М.; СПб; Нижн.Новг.; Минск, 2009.

Лабораторная работа №5 Функции

Цель работы: научиться правильно описывать различные массивы, уметь инициализировать массивы, распечатывать содержимое массива; научится решать задачи на использование массивов.

Теоретическая часть

Массив - это структурированный тип данных, который используется для описания упорядоченной совокупности фиксированного числа элементов одного типа, имеющих общее имя. Для обозначения элементов массива используются имя переменной-массива и индекс.

Массив - упорядоченные данные одного типа. Возможно создание массива, включающего массив другого типа. Массивом часто обозначают характеристики обьектов одного типа, имеющих одинаковые единицы измерения. Массив состоит из элементов, имеющих порядковые номера, т. е. элементы массива упорядоченны. Таким образом, если обьекты одного типа обозначить именем, например "A", то элементы обьекта будут A[1], A[2]

и т. д. В квадратных скобках указан номер элемента. Порядковый номер элемента массива, обычно не несет никакой информации о значении элемента, а показывает расположение элемента среди других. К элементам массива можно обращаться только по их номеру (индексу). Значения элементам массива присваиваются также как и другим переменным с учетом типа массива. Если элементы массива имеют один индекс, то массив называется одномерным или линейным, либо массив - вектор. Значения элементов одномерного массива обычно выводят на экран или бумагу в виде столбца или строки. В некоторых случаях удобно элементы массива пронумеровывать двуми независимыми индексами, такие массивы называются двумерными или матрицами. Значения элементов двумерного массива обычно выводят на экран в виде таблицы. Если элементы массива имеют три независимых индекса, то массив называется трехмерным. Значения элементов трехмерного массива обычно выводят на экран в виде набора таблиц.

Линейным массивом можно обозначить, например, оценки учеников класса. Каждая оценка является значением элемента массива оценок "A" и имеет порядковый номер (индекс). В Турбо-Паскале значение индекса указывается в квадратных скобках после имени массива. Можно создать массив фамилий "S" учеников класса. Значением элемента массива будет фамилия ученика, а индексом - порядковый номер по списку. Пусть дан список

№	Фамилии	Оценки
1	Иванов	5
2	Петров	4
3	Сидоров	5
4	Титов	5
...	...	...
30	Якупов	4

Описание массивов:

Var A : array[1..30] of byte;

S : array[1..30] of string; {или} SO: array[1..30] of string[12]; Присвоение значений элементам массива: "A" - A[1]:= 5; A[2]:= 4; и т. д.

"S " - S[1]:= 'Иванов'; S[2]:= 'Петров'; и т. д. Двумерные массивы

Массивы, рассмотренные выше, имеют элементы, упорядоченные по одному индексу, и называются одномерными массивами или векторами. Массив может быть двумерным, трехмерным и т. д. Двумерные массивы имеют элементы, упорядоченные по двум индексам, и часто называются матрицами. В Турбо-Паскале при описании многомерного массива диапазоны изменения индексов перечисляются через запятые, например:

Var

A: array[1..30, 1..7] of byte;

Рассмотрим пример работы с двумерными массивами. Обозначим массивом оценки учеников класса по нескольким предметам. Каждая оценка является значением элемента массива оценок "A" и имеет порядковый номер (два индекса). Поставим в соответствие первому индексу номер фамилии в списке учеников, а второму - номер предмета, по которому получена оценка. Тогда двумерный массив оценок можно представить в виде таблицы: каждый элемент a[i, j] находится на пересечении I-ой строки и J-го столбца.

Оборудование и материалы: для выполнения данного практического занятия необходим компьютер с установленной операционной системой Windows 7 и программными продуктами: Visual Studio.

Указания по технике безопасности:

К выполнению практических занятий допускаются студенты, ознакомившиеся с правилами работы в лаборатории, прошедшие инструктаж безопасности.

Задания: для выполнения практической работы необходимо выполнить следующее:

1. Изучить рекомендуемую литературу.
2. Выполнить практическую работу.
3. Ответить на контрольные вопросы.
4. Оформить отчет.

Содержание отчета: отчет по практическому занятию должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman, интервал – полуторный, поля левое – 3 см., правое – 1,5 см., верхнее и нижнее – 2 см. Абзацный отступ – 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой практической работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись (Рисунок 1 – Создание подсистемы) и ссылку на него в тексте. **Контрольные вопросы:**

1. Понятие массив.
2. Каким образом определяются переменные типа массив (одномерный и двумерный)?
3. Как осуществляется доступ к отдельному элементу одномерного и двумерного массива?
4. Каким образом выводятся элементы массива на экран?

Список литературы, рекомендуемый к использованию по данной теме:

1. Лафоре Р. Объектно-ориентированное программирование в С++. – М. – СПб.: Питер, 2008 г. – 922 с.
2. Юров В. Ассемблер. Учебник для вузов. – СПб.: Питер, 2008.- 624 с.
3. Юров В. Ассемблер. Практикум. – СПб.: Питер, 2008. - 356 с.
4. Бен-Ари М. Языки программирования: Практический сравнительный анализ: Учебник. – М.: Мир. – 2007.
5. Молчанов А.Ю. Системное программное обеспечение: Лабораторный практикум. – М.; СПб; Нижн.Новг.; Минск, 2009.

Лабораторная работа № 6. Структуры

Цель работы: ознакомиться с особенностями применения функций в языке Си++, с понятием прототипа и областью его применения, с понятием автоматических внешних, статических и регистровых переменных и их применением при составлении программ с использованием функций.

Теоретическая часть

Программы на языке СИ обычно состоят из большого числа отдельных функций (подпрограмм). Как правило, они имеют небольшие размеры и могут находиться как в одном, так и в нескольких файлах.

Связь между функциями осуществляется через аргументы, возвращаемые значения и внешние переменные.

Вызов функции осуществляется следующим образом: <тип функции >(параметр 1, параметр 2 , …);

Если функция имеет переменное число параметров, то вместо последнего из них указывается многоточие.

Передача одного значения из вызванной функции в вызвавшую происходит с помощью оператора возврата, который записывается в следующем виде:

return (выражение);

В этом случае значение выражения (в частном случае может быть просто переменная) передается в основную программу и подставляется вместо обращения к функции. Пусть вызывающая программа обращается к функции следующим образом:

a=fun(b,c);

Здесь b и c – аргументы, значения которых передаются в вызываемую подпрограмму.

Если описание функции начинается так: fun(i,j) , то переменные i и j получат значения a и b соответственно.

Пример 1. Оформить получение абсолютной величины числа в виде функции. Сама функция может быть оформлена в виде отдельного файла. В этом случае выполняется его включение процедурой #include. Программа имеет следующий вид:

```
#include <stdio.h> main()
{int a=10,b=0,c=-20; int d,e,f;
d=abs(a); /*обращение к функции abs*/ b=abs(b); f=abs(c);
printf("%d %d %d",d,b,f); }
#include "abc.c" /*включениефайла abc.c сфункцией abs*/ /*Функция, вычисляющая абсолютную величину числа */ abs(x) int x; /*Описание переменных, работающих в функции */
{int y;
y=(x<0)?–x:x; /*Определение абсолютной величины числа*/ return (y); /*Возвращает значение y вызывающей программе*/ }
```

В приведенной программе описание типа функции было опущено. Это возможно только в том случае, если возвращенное значение имеет целый тип. Во всех остальных случаях описание типа функции обязательно. приведем пример, когда результатом работы функции будет число двойной точности.

Пример 2. Оформить в виде функции вычисление f=Öx + y/z.

В первом примере функция хранилась в виде отдельного файла и включалась процедурой #include. Функция может быть включена в один файл с вызывающей программой. В этом случае процедура #include не требуется, а сама функция должна быть объявлена в основной программе, если она имеет не целый тип. Приведем программу для примера 2, оформленную таким способом. Программа имеет вид:

```
#include <stdio.h> main()
{ double f,x=5.5,y=10.1,z=20.5, vv() /*объявлены переменные и функция vv*/ f=vv(x,y,z);
```

```
/*обращение к функции vv*/
        printf("lf",f); /*вывод результата */
        }
        /*функция */ double vv(x,y,z)
        double x,y,z; /*объявление переменных функции */
        {double f;
        f=sqrt(x)+y/z; /*вычисление значения функции */ return(f); /*возврат вычисленного
значения функции */
        }
```

В языке СИ аргументы функции передаются по значению, т.е. вызванная функция получает временную копию каждого аргумента, а не его адрес. Это означает, что функция не может изменять оригинальный аргумент в вызвавшей ее программе. Однако это легко сделать, если передавать в функцию не переменные, а их адреса.

Пример 3. В приведенной ниже программе вводятся некоторые значения переменных a и b, потом в функции izm они меняются местами.

```
        #include <stdio.h> main()
        {int a,b;
        scanf ("%d %d", &a, &b); izm (&a, &b); /*обращение к функции izm; аргументами
        являются адреса переменных a
и b*/ printf("%d, %d",a, b); /*вывод на экран измененных значений */
        }
        #include "izm.c" /*включение файла izm.c с функцией izm */
        /*функция*/
        izm(a, d); /*аргументы a и b являются указателями */ int *a, *b; /*
        *a и *b – значения, на которые указывают указатели */ {int c; c=*a;
        *a = *b;
        *b=c; /*обменместами */
        }
```

Функция izm получает копию адресов переменных a и b, меняет местами значения, записанные по этим адресам, и передает управление в основную программу. Адреса &a и &b в основной программе не изменялись, а вот значения, на которые они указывают, поменялись местами.

Если в качестве аргумента функции используется имя массива, то ей передается адрес начала массива, а сами элементы не копируются. Функция может изменять элементы массива, сдвигаясь (индексированием) от его начала.

Пример 4. В массиве S поменять местами элементы: первый со вторым, третий с четвертым и т.д. Оформить этот алгоритм в виде функции reverse.

```
        #include <stdio.h> main()
        {int i,j,s[6]; /* описание переменных i,j и массива s целого типа */ for (i=0; i<6; i++)
        scanf("%d",&s[i]); /*вводэлементовмассива s*/ reverse(s); /*обращениекфункции
reverse*/
        for (i=0; i<6; i++)
        printf("%d",s[i]); /*вывод полученного массива */
        }
        include "reverse.c" /*включениефайла reverse.c сфункцией reverse */
```

```
/*функция*/ reverse(s)
int s[]; /*описание работающего в подпрограмме массива */
{
int a,i;
for (i=1; i<5; i+=2)
{a=s[i]; s[i]=s[i+1]; s[i+1]=a;} /*обменэлементовместами*/
}
```

Рассмотрим особенности работы функции с двумерным массивом. В предыдущем примере в функции массив был описан как int s[]; для двумерного массива а нельзя записать a[][]. В описании двумерного массива во второй квадратной скобке должно быть указано количество столбцов, например: a[][3].

Пример 5. Увеличить все элементы массива a(5,5) в два раза. Оформить этот алгоритм в виде подпрограммы.

```
#include <stdio.h> main()
{int a[5][5]; /*описаниемассива a*/ int i,j; /*объявление переменных i,j*/ for
(i=0;i<5;i++) for (j=0; j<5; j++) scanf("%d",a[i][j]); /*вводмассива*/ mas(a); /*обращение
к функции mas*/ for (i=0; i<5; i++) for (j=0; j<5; j++) printf("%d", a[i][j]); /*вывод
полученного результата*/ }
/*функция*/ mas(a)
int a[][5]; /*описание массива a*/
{int i,j; /*описание переменных i,j*/ for (i=0; i<5; i++) for
(j=0; j<5; j++) a[i][j] = 2*a[i][j]; /*увеличение элементов
массива в 2 раза*/
}
```

Классы памяти

В языке СИ различают четыре основных памяти: внешнюю (глобальную), автоматическую (локальную), статическую и регистровую.

Внешние переменные определены вне любой из функций, следовательно, доступны для многих из них. Область внешней переменной простирается от точки во входном файле, где она объявлена, и до конца файла. Если внешняя переменная определена в другом файле, то вступает в силу описание extern (внешний). На рис.1 показано, где объявляются и на что распространяется область действия внешних переменных, если программа main и вызываемая функция находятся в данном файле. На рис. 2 демонстрируются отличия, имеющие место, когда main и вызываемая функция находятся в разных файлах. В файле с вызываемой функцией внешние переменные будут доступны после их описания с помощью ключевого слова extern.

Пример 5. Оформить в виде функции вычисление выражения: f=a×x2+b×x+c;

В приведенной ниже программе заданные переменные объявлены как внешние, причем основная программа и функция находятся в одном файле.

```
#include <stdio.h> int a=5, b=7, c=10,x; /* Объявление внешних переменных a,b,c,x
целого типа*/ main ()
{ int f;
scanf ("%d", &x); /*Ввод значения переменной x*/ f=kv(); /*обращение к функции*/
printf ("%d",f); /*вывод на экран значения переменной f*/ }
/*функция*/ kv()
{int f;
```

```
f=a*x*x+b*x+c; /*вычисление значения f*/
return (f); /*возвращает значение f вызывающей программе*/
}
```

Если сравнить эту программу с программой, приведенной в примере 2, то можно обнаружить два различия:

1) после имени функции в скобках отсутствуют аргументы;
2) в функции не объявлены переменные, с которыми работает функция.

Это стало возможным потому, что переменные объявлены внешними, а значит они известны всему файлу, в том числе и функции.

Внешние переменные должны быть описаны до функции main(). Только в этом случае они становятся внешними (см. рис. 1).

Приведем программу для этого же примера, рассмотрев случай, когда основная программа и функция расположены в разных файлах.

```
#include <stdio.h> int a=5, b=7, c=10,x,f; /* Объявление внешних переменных a,b,c,x,f
целого типа*/ main
() { scanf ("%d", &x); /*Ввод значения переменной x*/ f=kv(); /*обращение к функции*/
printf ("%d",f); /*вывод на экран значения переменной f*/ }
#include "kv.c" /*включение файла kv.c функцией kv*/
/*функция*/ kv() {extern int a,b,c,x,f;
f=a*x*x+b*x+c; /*вычисление значения
f*/
return (f); /*возвращает значение f вызывающей программе*/ }
```

Как было сказано выше (см. рис. 2), если основная программа и функция расположены в разных файлах, то переменные в функции должны быть вписаны при помощи ключевого слова extern. Рассмотрим теперь статические переменные. Статические переменные имеют такую же область действия, как автоматические, но они не исчезают, когда содержащая их функция закончит свою работу. Компилятор хранит их значения от одного вызова функции до другого. Статические переменные объявляются с помощью ключевого слова static. Можно статические переменные описать вне любой функции. Это создает внешнюю статическую переменную. Разница между внешней переменной и внешней статической переменной заключается в области их действия. Обычная внешняя переменная может использоваться функциями в любом файле (с помощью ключевого слова extern), в то время как внешняя статическая переменная может использоваться только функциями того же самого файла.

Регистровые переменные относятся к последнему классу. Ключевое слово register указывает, что переменная, о которой идет речь, будет интенсивно использоваться. Если это возможно, значения таких переменных помещаются во внутренние регистры процессора, благодаря чему программа будет более быстрой.

Оборудование и материалы: для выполнения данного практического занятия необходим компьютер с установленной операционной системой Windows 7 и программными продуктами: Visual Studio.

Указания по технике безопасности:

К выполнению практических занятий допускаются студенты, ознакомившиеся с правилами работы в лаборатории, прошедшие инструктаж безопасности.

Задания: для выполнения практической работы необходимо выполнить следующее:

1. Изучить рекомендуемую литературу.

2. Выполнить практическую работу.
3. Ответить на контрольные вопросы.
4. Оформить отчет.

Содержание отчета: отчет по практическому занятию должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman, интервал – полуторный, поля левое – 3 см., правое – 1,5 см., верхнее и нижнее – 2 см. Абзацный отступ – 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой практической работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись (Рисунок 1 – Создание подсистемы) и ссылку на него в тексте. **Контрольные вопросы:**

1. Понятие функции.
2. Как объявляется функция?
3. Как вызывается функция?
4. Классы памяти. **Список литературы, рекомендуемый к использованию по данной теме:**

1. Лафоре Р. Объектно-ориентированное программирование в С++. – М. – СПб.:
Питер, 2008 г. – 922 с.
2. Юров В. Ассемблер. Учебник для вузов. – СПб.: Питер, 2008.- 624 с.
3. Юров В. Ассемблер. Практикум. – СПб.: Питер, 2008. - 356 с.
4. Бен-Ари М. Языки программирования: Практический сравнительный анализ: Учебник. – М.: Мир. – 2007.
5. Молчанов А.Ю. Системное программное обеспечение: Лабораторный практикум.
– М.; СПб; Нижн.Новг.; Минск, 2009.

Лабораторная работа №7. Запись и чтение данных файла

Цель работы:

Познакомиться с понятием структуры и структурной переменной.

Научиться создавать массивы структур и работать со вложенными структурами.

Теоретическая часть

Структура – это объединение одного либо более объектов (переменных, массивов, указателей, других структур). Как и массив, она представляет собой совокупность данных, но отличается от него тем, что к ее элементам необходимо обращаться по имени, и ее различные элементы не обязательно должны принадлежать одному типу.

Структуры удобно использовать там, где разнообразные данные, относящиеся к одному и тому же объекту, необходимо объединять. Например, ученика средней школы характеризуют следующие данные: фамилия, имя, дата рождения, класс, возраст.

Объявление структуры осуществляется с помощью ключевого слова struct, за которым следует ее тип, список элементов, заключенных в фигурные скобки. Ее можно представить в следующем общем виде:

struct тип {тип элемента 1 имя элемента 1; тип элемента n имя элемента n; };

Именем элемента может быть любой идентификатор. В одной строке можно записывать через запятую несколько идентификаторов одного типа. Например:

struct date { int day; int month; int

year;} ;

Русские буквы использовать в идентификаторе в языке СИ нельзя.

Следом за фигурной скобкой, заканчивающей список элементов, могут записываться переменные данного типа, например:

struct date {…} a, b, c;

При этом выделяется соответствующая память.

Выведенное имя типа можно использовать для объявления записи, например: struct date day;. Теперь переменная day имеет тип date.

Разрешается вкладывать структуры одна на другую. Для лучшего восприятия структуры используем русские буквы в идентификаторах, в языке СИ этого делать нельзя.

Например:

struct УЧЕНИК { char Фамилия [15]; имя [15]; struct

DATA ДАТА РОЖДЕНИЯ;

int класс, возраст;};

Определенный выше тип DATA включает три элемента: День, Месяц, Год, содержащие целые значения (int). Запись УЧЕНИК включает элементы: ФАМИЛИЯ [15]; ИМЯ[15]; ДАТА РОЖДЕНИЯ, КЛАСС, ВОЗРАСТ. ФАМИЛИЯ [15] и ИМЯ [15] символьные массивы из 15 компонент каждый. Переменная ДАТА РОЖДЕНИЯ представлена составным элементом (вложенной структурой) ДАТА. Каждой дате рождения соответствуют день месяца, месяц и год. Элементы КЛАСС и ВОЗРАСТ содержат значения целого типа (int). После введения типов ДАТА и УЧЕНИК можно объявить переменные, значения которых принадлежат этим типам.

Например:

struct УЧЕНИК УЧЕНИКИ [50]; массив УЧЕНИКИ состоит

из 50 элементов типа УЧЕНИК.

В языке СИ разрешено использовать массивы структуры; записи могут состоять из массивов и других записей.

Чтобы обратиться к отдельному компоненту структуры, необходимо указать ее имя, поставить точку и сразу за ней написать имя нужного элемента.

Например:

Ученики [1]. КЛАСС = 3;

Ученики [1]. ДАТА РОЖДЕНИЯ. ДЕНЬ=5; Ученики [1]. ДАТА РОЖДЕНИЯ. МЕСЯЦ=4; Ученики [1]. ДАТА РОЖДЕНИЯ. ГОД=1979;

Первая строка указывает, что 1-й ученик учится в третьем классе, а последующие строки – его дату рождения: 5.04.79.

Каждый тип элемента структуры определяется соответствующей строкой объявления в фигурных скобках. Например, массив УЧЕНИКИ имеет тип УЧЕНИК, год является целым числом. Так как каждый элемент записи относится к определенному типу, его составное имя может появляться везде, где разрешено использовать значение этого типа. Рассмотрим пример программы:

```
/* Демонстрация записи */ #include < stdio.h >
struct computer { int mem; int sp; char
model [20]; };
/* Объявление записи типа computer, состоящей из трех элементов: mem, sp, model */
struct computer pibm =
{512, 1, “ПЭВМЕС 1840.05”}
/* Объявление и инициализация переменной pibm типа computer */ main ( )
{ printf (“ персональная ЭВМ % s\n\n “, pibm.model); printf ( “объем
оперативной памяти - % d К байт \n”, pibm.mem); printf
(“производительность - % d млн. операций в секунду \n”, pibm.sp);
/* вывод на экран значений элементов структуры */ }
```

В данной программе объявляется запись computer, которая состоит из трех элементов: mem (память ЭВМ), sp (быстродействие), model [20] (модель ПЭВМ). Переменная pibm имеет тип computer и является глобальной. Строки pibm.model, pibm.mem, pibm. sp в операторе printf вызывают обращение к соответствующим элементам записи pibm типа computer, которым ранее были присвоены определенные значения.

Результат работы программы имеет вид: персональная ЭВМ ПЭВМ ЕС 1840.05 объем оперативной памяти – 512 К байт производительность – 1 млн. операций в секунду

Рассмотрим использование в программе вложенных структур:

```
/* Демонстрация вложенных структур*/ # include < stdio.h > struct
date { int day; int month;
int year; };
/* Объявление записи типа date*/ struct person { char fam [20]; char
im [20];
char ot [20]; struct date f1;};
/* Объявление структуры типа person; одним из элементов записи person является
запись f1 типа date */ main ( )
{ struct person ind1;
/* обьявление переменной ind1 типа person */
printf ( “Укажите фамилию, имя, отчество, день, \n месяц” “ и год рождения гражданина
ind1\n”);
scanf (“ % S % S % S %d %d”, &ind1.fam, &ind1.im, &ind1.ot, & ind1.f1.day,
&ind1.f1.month, &ind1.f1.year );
/* Ввод сведений о гражданине ind1 */ printf (“ Фамилия, имя, отчество: % S % S % S
\n”, ind1.fam, ind1.im, ind1.ot); printf (“
Годрождения - % d \n”, ind1.f1.year); printf (“ Месяцрождения - % d -й \n”, ind1.f1.month); printf
(“ День рождения - % d -й \n”,
ind1.f1.day);
/* Вывод сведений о гражданине ind1 */ }
```

Структура типа date (дата) содержит три элемента: day (день), month (месяц), year (год). Структура типа person (человек) содержит четыре элемента: fam[20] (фамилия), im[20] (имя) , ot[20] (отчество), f1 (дата рождения). Последний из них (f1) – это вложенная запись типа date. Результаты работы программы:

Укажите фамилию, имя, отчество, день, месяц и год рождения гражданина ind1
Алексеев

Сергей Петрович 3

5

1978

Подчеркнутая информация вводится пользователем. Сведения о гражданине ind1 Фамилия, имя, отчество: Алексеев Сергей Петрович Год рождения – 1978

Месяц рождения – 5-й День рождения – 3-й

В следующей программе рассмотрим использование структуры в виде элементов массива pibm. Каждый элемент состоит из следующих компонентов: mem (память), sp (объем винчестера), model [20] (модель ПЭВМ):

```
/* Массивы записей */ #include < stdio.h > struct
computer { int mem, sp; char model [20];
9)\n" ); pibm
[10];};
/* объявлениезаписитипа computer; объявление массива pibm типа computer */ main ( )
{ int i, j, k, priz; for
( i=0; i<10; i++)
{ printf ("Введите сведения о ПЭВМ %d и признак (0-конец; \n
другая цифра- продолжение)\n", i); printf (" модельПЭВМ - ");
scanf ("%S", &pibm [i].model );
printf ( "объемоперативнойпамяти -"); scanf ("%d", &pibm[i].mem); printf ("
объемвинчестера - "); scanf ( "%d , &pibm[i].sp "); printf ("признак - "); scanf ("
%d ", &priz ); k=i; if (!priz) break; }
/* Здесь !priz – операцияотрицания priz; break – выходизцикла for, если priz=0 */ for (i=0;
i<10, i++);
{ printf ( "\n О какой ПЭВМ Вы хотите получить сведения?\n (Введите номер от 0
до


scanf ( "%d ",&j ); if (j>k)
{ printf ("Нет сведений об этой ПЭВМ \n"); continue; } printf
(" персональнаяЭВМ %s\n ", pibm[j].model);
printf ("объем оперативной памяти - % d Мб \n ", pibm[j].mem); printf ("объем винчестера
- % d Мб \n ", pibm[j].sp);
printf (" признак – " );
scanf ( " %d ", &priz); if
(!priz) break; }
/* Ввод сведений о ПЭВМ и занесение в массив pibm записей типа computer (первый
цикл for); вывод на экран сведений о ПЭВМ (второй цикл for) */
}
```

Результаты работы программы:

Введите сведения о ПЭВМ и признак (0-конец; другая цифра – продолжение) модель ПЭВМ – AT 486 SX объем оперативной памяти – 32 объем винчестера – 4 Гбайта

признак – 1

Введите сведения о ПЭВМ и признак (0-конец; другая цифра – продолжение) модель ПЭВМ – AT 386 DX объем оперативной памяти – 64 объем винчестера – 14

Гбайт признак – 0 О какой ПЭВМ Вы хотите получить сведения?

(Введитн номер от 0 до 9) 1 модель ПЭВМ – AT 386 DX объем

оперативной памяти – 16 Мб объем винчестера – 2,5 Гбайт признак – 0

Оборудование и материалы: для выполнения данного практического занятия необходим компьютер с установленной операционной системой Windows 7 и программными продуктами: Visual Studio. **Указания по технике безопасности:**

К выполнению практических занятий допускаются студенты, ознакомившиеся с правилами работы в лаборатории, прошедшие инструктаж безопасности.

Задания: для выполнения практической работы необходимо выполнить следующее:

1. Изучить рекомендуемую литературу.
2. Выполнить практическую работу.
3. Ответить на контрольные вопросы.
4. Оформить отчет.

Содержание отчета: отчет по практическому занятию должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman, интервал – полуторный, поля левое – 3 см., правое – 1,5 см., верхнее и нижнее – 2 см. Абзацный отступ – 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой практической работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись (Рисунок 1 – Создание подсистемы) и ссылку на него в тексте.

Контрольные вопросы:

1. Понятие структуры.
2. Объявление структуры.
3. Доступ к элементам структуры.
4. Разрешено ли использование русских букв в идентификаторе структуры. **Список литературы, рекомендуемый к использованию по данной теме:**

1. Лафоре Р. Объектно-ориентированное программирование в C++. – М. – СПб.:
Питер, 2008 г. – 922 с.
2. Юров В. Ассемблер. Учебник для вузов. – СПб.: Питер, 2008.- 624 с.
3. Юров В. Ассемблер. Практикум. – СПб.: Питер, 2008. - 356 с.
4. Бен-Ари М. Языки программирования: Практический сравнительный анализ: Учебник. – М.: Мир. – 2007.
5. Молчанов А.Ю. Системное программное обеспечение: Лабораторный практикум. – М.; СПб; Нижн.Новг.; Минск, 2009.

Лабораторная работа № 8 Операторы цикла for и while

Цель работы: научиться работать с записью и чтением данных на файл.

Оборудование и материалы: для выполнения данного практического занятия необходим компьютер с установленной операционной системой Windows 7 и программными продуктами: Visual Studio.

Указания по технике безопасности:

К выполнению практических занятий допускаются студенты, ознакомившиеся с правилами работы в лаборатории, прошедшие инструктаж безопасности.

Задания: для выполнения практической работы необходимо выполнить следующее:

1. Изучить рекомендуемую литературу.
2. Выполнить практическую работу.
3. Ответить на контрольные вопросы.
4. Оформить отчет.

Содержание отчета: отчет по практическому занятию должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman, интервал – полуторный, поля левое – 3 см., правое – 1,5 см., верхнее и нижнее – 2 см. Абзацный отступ – 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой практической работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись (Рисунок 1 – Создание подсистемы) и ссылку на него в тексте.

Контрольные вопросы:

1. Указатели и их назначение
2. Указатели на объекты. Определение. Инициализация.
3. Типы указателей. Указатель NULL.
4. Доступ к переменной по указателю.
5. Доступ к элементу массива по указателю
6. Указатели на функции
7. Выделение памяти под хранение данных в динамическом массиве **Повышенный уровень:**
8. Связанный список
9. Односвязный список
10. Двусвязный список
11. Односвязный циклический список
12. Двусвязный циклический список **Список литературы, рекомендуемый к использованию по данной теме:**

1. Лафоре Р. Объектно-ориентированное программирование в C++. – М. – СПб.:
Питер, 2008 г. – 922 с.
2. Юров В. Ассемблер. Учебник для вузов. – СПб.: Питер, 2008.- 624 с.
3. Юров В. Ассемблер. Практикум. – СПб.: Питер, 2008. - 356 с.

4. Бен-Ари М. Языки программирования: Практический сравнительный анализ: Учебник. – М.: Мир. – 2007.

5. Молчанов А.Ю. Системное программное обеспечение: Лабораторный практикум.

– М.; СПб; Нижн.Новг.; Минск, 2009.

Лабораторная работа № 9. Введение в ООП

Цель работы: Ознакомиться с основными понятиями ООП.

Оборудование и материалы: для выполнения данного практического занятия необходим компьютер с установленной операционной системой Windows 7 и программными продуктами: Visual Studio.

Указания по технике безопасности:

К выполнению практических занятий допускаются студенты, ознакомившиеся с правилами работы в лаборатории, прошедшие инструктаж безопасности.

Задания: для выполнения практической работы необходимо выполнить следующее:

1. Изучить рекомендуемую литературу.
2. Выполнить практическую работу.
3. Ответить на контрольные вопросы.
4. Оформить отчет.

Содержание отчета: отчет по практическому занятию должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman, интервал – полуторный, поля левое – 3 см., правое – 1,5 см., верхнее и нижнее – 2 см. Абзацный отступ – 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой практической работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись.

Контрольные вопросы:

1. Потоковый класс IOS.
2. Иерархия классов.
3. Библиотека fstream. Ввод и вывод данных в файл.

Список литературы, рекомендуемый к использованию по данной теме:

1. Лафоре Р. Объектно-ориентированное программирование в C++. – М. – СПб.:

Питер, 2008 г. – 922 с.

2. Юров В. Ассемблер. Учебник для вузов. – СПб.: Питер, 2008.- 624 с.

3. Юров В. Ассемблер. Практикум. – СПб.: Питер, 2008. - 356 с.

4. Бен-Ари М. Языки программирования: Практический сравнительный анализ: Учебник. – М.: Мир. – 2007.

5. Молчанов А.Ю. Системное программное обеспечение: Лабораторный практикум.

– М.; СПб; Нижн.Новг.; Минск, 2009.

Лабораторная работа № 10. Классы. Связные списки

Цель работы: Ознакомиться с членами и методами класса, конструкторами и деструкторами.

Оборудование и материалы:

для выполнения данного практического занятия необходим компьютер с установленной
операционной системой Windows 7 и программными
продуктами: Visual Studio.

Указания по технике безопасности:

К выполнению практических занятий допускаются студенты, ознакомившиеся с правилами работы в лаборатории, прошедшие инструктаж безопасности.

Задания: для выполнения практической работы необходимо выполнить следующее:

1. Изучить рекомендуемую литературу.
2. Выполнить практическую работу.
3. Ответить на контрольные вопросы.
4. Оформить отчет.

Содержание отчета: отчет по практическому занятию должен быть выполнен редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman, интервал – полуторный, поля левое – 3 см., правое – 1,5 см., верхнее и нижнее – 2 см. Абзацный отступ – 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой практической работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись и ссылку на него в тексте.

Контрольные вопросы: 1.
Конструкторы и деструкторы

2. Статические члены класса.
3. Статические методы класса.
4. Методы класса, возвращающие значения. Методы класса, не возвращающие значения
5. Конструкторы, их назначение.
6. Синтаксис описания.
7. Деструкторы, их назначение.
8. Синтаксис описания. **Список литературы, рекомендуемый к использованию по данной теме:**

1. Лафоре Р. Объектно-ориентированное программирование в С++. – М. – СПб.:
Питер, 2008 г. – 922 с.

2. Юров В. Ассемблер. Учебник для вузов. – СПб.: Питер, 2008.- 624 с.
3. Юров В. Ассемблер. Практикум. – СПб.: Питер, 2008. - 356 с.
4. Бен-Ари М. Языки программирования: Практический сравнительный анализ: Учебник. – М.: Мир. – 2007.
5. Молчанов А.Ю. Системное программное обеспечение: Лабораторный практикум. – М.; СПб; Нижн.Новг.; Минск, 2009.

Лабораторная работа №11. Перегрузка методов и конструкторов

Цель работы: Ознакомиться с понятием перезагрузки системы

Оборудование и материалы: для выполнения данного практического занятия необходим компьютер с установленной операционной системой Windows 7 и программными продуктами: Visual Studio.

Указания по технике безопасности:

К выполнению практических занятий допускаются студенты, ознакомившиеся с правилами работы в лаборатории, прошедшие инструктаж безопасности.

Задания: для выполнения практической работы необходимо выполнить следующее:

1. Изучить рекомендуемую литературу.
2. Выполнить практическую работу.
3. Ответить на контрольные вопросы.
4. Оформить отчет.

Содержание отчета: отчет по практическому занятию должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию:

Шрифт TimesNewRoman, интервал – полуторный, поля левое – 3 см., правое – 1,5 см., верхнее и нижнее – 2 см. Абзацный отступ – 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой практической работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись (Рисунок 1 – Создание подсистемы) и ссылку на него в тексте. **Контрольные вопросы:**

1. Перегрузка функций
2. Примеры перегрузки
3. Перегрузка методов класса. Примеры перегрузки методов.
4. Перегрузка конструкторов. Примеры перегрузки.
5. Перегрузка операций: алгебраические операции.
6. Перегрузка операций: логические операции.
7. Операции с объектами класса. Методы, возвращающие объекты класса. **Список литературы, рекомендуемый к использованию по данной теме:**

1. Лафоре Р. Объектно-ориентированное программирование в C++. – М. – СПб.:
Питер, 2008 г. – 922 с.
2. Юров В. Ассемблер. Учебник для вузов. – СПб.: Питер, 2008.- 624 с.

3. Юров В. Ассемблер. Практикум. – СПб.: Питер, 2008. - 356 с.

4. Бен-Ари М. Языки программирования: Практический сравнительный анализ: Учебник. – М.: Мир. – 2007. 5. Молчанов А.Ю. Системное программное обеспечение: Лабораторный практикум. – М.; СПб; Нижн.Новг.; Минск, 2009.

Лабораторная работа № 12. Перегрузка операторов

Цель работы: Ознакомиться с понятием наследование.

Оборудование и материалы: для выполнения данного практического занятия необходим компьютер с установленной операционной системой Windows 7 и программными продуктами: Visual Studio.

Указания по технике безопасности:

К выполнению практических занятий допускаются студенты, ознакомившиеся с правилами работы в лаборатории, прошедшие инструктаж безопасности.

Задания: для выполнения практической работы необходимо выполнить следующее:

1. Изучить рекомендуемую литературу.
2. Выполнить практическую работу.
3. Ответить на контрольные вопросы.
4. Оформить отчет.

Содержание отчета: отчет по практическому занятию должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman, интервал – полуторный, поля левое – 3 см., правое – 1,5 см., верхнее и нижнее – 2 см. Абзацный отступ – 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой практической работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись (Рисунок 1 – Создание подсистемы) и ссылку на него в тексте.

Контрольные вопросы:

1. Наследование. Основные понятия.
2. Базовый и производный классы.
3. Конструкторы производного класса. Базовые функции класса.
4. Диаграммы UML. Средства Visual Studio для построения UML диаграмм.
5. Пример построение базового класса с двумя классами наследниками.
6. Иерархия классов.
7. Диаграммы UML. Средства Visual Studio для построения UML диаграмм.
8. Множественное наследование.

Список литературы, рекомендуемый к использованию по данной теме:

1. Лафоре Р. Объектно-ориентированное программирование в C++. – М. – СПб.:
Питер, 2008 г. – 922 с.

2. Юров В. Ассемблер. Учебник для вузов. – СПб.: Питер, 2008.- 624 с.

3. Юров В. Ассемблер. Практикум. – СПб.: Питер, 2008. - 356 с.
4. Бен-Ари М. Языки программирования: Практический сравнительный анализ: Учебник. – М.: Мир. – 2007.
5. Молчанов А.Ю. Системное программное обеспечение: Лабораторный практикум.
– М.; СПб; Нижн.Новг.; Минск, 2009.

Лабораторная работа №13. Перегрузка операторов

Цель работы: Ознакомиться с понятием виртуальной функции.

Оборудование и материалы: для выполнения данного практического занятия необходим компьютер с установленной операционной системой Windows 7 и программными продуктами: Visual Studio.

Указания по технике безопасности:

К выполнению практических занятий допускаются студенты, ознакомившиеся с правилами работы в лаборатории, прошедшие инструктаж безопасности.

Задания: для выполнения практической работы необходимо выполнить следующее:

1. Изучить рекомендуемую литературу.
2. Выполнить практическую работу.
3. Ответить на контрольные вопросы.
4. Оформить отчет.

Содержание отчета: отчет по практическому занятию должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman, интервал – полуторный, поля левое – 3 см., правое – 1,5 см., верхнее и нижнее – 2 см. Абзацный отступ – 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой практической работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись (Рисунок 1 – Создание подсистемы) и ссылку на него в тексте.

Контрольные вопросы:

1. Описание виртуальные функции.
2. Использование виртуальных функций
3. Дружественные функции.
4. Статические функции.
5. Инициализация копирования и присваивания. Указатель this.

Список литературы, рекомендуемый к использованию по данной теме:

1. Лафоре Р. Объектно-ориентированное программирование в C++. – М. – СПб.:
Питер, 2008 г. – 922 с.
2. Юров В. Ассемблер. Учебник для вузов. – СПб.: Питер, 2008.- 624 с.
3. Юров В. Ассемблер. Практикум. – СПб.: Питер, 2008. - 356 с.

4. Бен-Ари М. Языки программирования: Практический сравнительный анализ: Учебник. – М.: Мир. – 2007. 5. Молчанов А.Ю. Системное программное обеспечение: Лабораторный практикум. – М.; СПб; Нижн.Новг.; Минск, 2009.

Лабораторная работа №14. Множественное наследование

Цель работы: научиться работать с шаблонами и исключениями.

Оборудование и материалы: для выполнения данного практического занятия необходим компьютер с установленной операционной системой Windows 7 и программными продуктами: Visual Studio.

Указания по технике безопасности:

К выполнению практических занятий допускаются студенты, ознакомившиеся с правилами работы в лаборатории, прошедшие инструктаж безопасности.

Задания: для выполнения практической работы необходимо выполнить следующее:

1. Изучить рекомендуемую литературу.
2. Выполнить практическую работу.
3. Ответить на контрольные вопросы.
4. Оформить отчет.

Содержание отчета: отчет по практическому занятию должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman, интервал – полуторный, поля левое – 3 см., правое – 1,5 см., верхнее и нижнее – 2 см. Абзацный отступ – 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой практической работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись (Рисунок 1 – Создание подсистемы) и ссылку на него в тексте.

Контрольные вопросы:

1. Шаблоны функций.
2. Шаблоны классов.
3. Введение в STL.
4. Алгоритмы.
5. Последовательные контейнеры.
6. Итераторы. Ассоциированные контейнеры. **Список литературы, рекомендуемый к использованию по данной теме:**

1. Лафоре Р. Объектно-ориентированное программирование в C++. – М. – СПб.:
Питер, 2008 г. – 922 с.
2. Юров В. Ассемблер. Учебник для вузов. – СПб.: Питер, 2008.- 624 с.

3. Юров В. Ассемблер. Практикум. – СПб.: Питер, 2008. - 356 с.
4. Бен-Ари М. Языки программирования: Практический сравнительный анализ: Учебник. – М.: Мир. – 2007.
5. Молчанов А.Ю. Системное программное обеспечение: Лабораторный практикум.
– М.; СПб; Нижн.Новг.; Минск, 2009.

Лабораторная работа №15. Виртуальные функции

Цель работы: Ознакомиться с основными понятиями компонентно-ориентированного программирования.

Оборудование и материалы:

для выполнения данного практического занятия необходим компьютер с установленной
операционной системой Windows 7 и программными продуктами: Visual Studio.

Указания по технике безопасности:

К выполнению практических занятий допускаются студенты, ознакомившиеся с правилами работы в лаборатории, прошедшие инструктаж безопасности.

Задания: для выполнения практической работы необходимо выполнить следующее:

1. Изучить рекомендуемую литературу.
2. Выполнить практическую работу.
3. Ответить на контрольные вопросы.
4. Оформить отчет.

Содержание отчета: отчет по практическому занятию должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman, интервал – полуторный, поля левое – 3 см., правое – 1,5 см., верхнее и нижнее – 2 см. Абзацный отступ – 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой практической работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись (Рисунок 1 – Создание подсистемы) и ссылку на него в тексте.

Контрольные вопросы:

1. Современные подходы к программированию
2. Компонентный подход к программированию как расширение ООП. Обзор архитектурного решения .NET
3. Гетерогенное компонентное программирование в .NET
4. Среды программирования, использующие компонентно-ориентированный подход.
5. Введение в КОП.
6. Классы и объекты в компонентно-ориентированном программировании.
7. Понятие «компонент».

8. Реализация КОП в Delphi, C++ и C#.

Список литературы, рекомендуемый к использованию по данной теме:

1. Лафоре Р. Объектно-ориентированное программирование в C++. – М. – СПб.:
Питер, 2008 г. – 922 с.
2. Юров В. Ассемблер. Учебник для вузов. – СПб.: Питер, 2008.- 624 с.
3. Юров В. Ассемблер. Практикум. – СПб.: Питер, 2008. - 356 с.
4. Бен-Ари М. Языки программирования: Практический сравнительный анализ: Учебник. – М.: Мир. – 2007.
5. Молчанов А.Ю. Системное программное обеспечение: Лабораторный практикум.
– М.; СПб; Нижн.Новг.; Минск, 2009.

Лабораторная работа №16. Шаблоны и исключения

Цель работы: Научиться разрабатывать двухоконное Window-приложение.

Оборудование и материалы: для выполнения данного практического занятия необходим компьютер с установленной
операционной системой Windows 7 и программными продуктами: Visual Studio. **Указания по технике безопасности:**

К выполнению практических занятий допускаются студенты, ознакомившиеся с правилами работы в лаборатории, прошедшие инструктаж безопасности.

Задания: для выполнения практической работы необходимо выполнить следующее:

1. Изучить рекомендуемую литературу.
2. Выполнить практическую работу.
3. Ответить на контрольные вопросы.
4. Оформить отчет.

Содержание отчета: отчет по практическому занятию должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman, интервал – полуторный, поля левое – 3 см., правое – 1,5 см., верхнее и нижнее – 2 см. Абзацный отступ – 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой практической работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

Контрольные вопросы:

1. Основные методы формы, управляющие отображением формы на экране.
2. Вложенные элементы управления и контейнеры. **Список литературы, рекомендуемый к использованию по данной теме:**

1. Лафоре Р. Объектно-ориентированное программирование в C++. – М. – СПб.:
Питер, 2008 г. – 922 с.
2. Юров В. Ассемблер. Учебник для вузов. – СПб.: Питер, 2008.- 624 с.

3. Юров В. Ассемблер. Практикум. – СПб.: Питер, 2008. - 356 с.
4. Бен-Ари М. Языки программирования: Практический сравнительный анализ: Учебник. – М.: Мир. – 2007.
5. Молчанов А.Ю. Системное программное обеспечение: Лабораторный практикум.
– М.; СПб; Нижн.Новг.; Минск, 2009.

Лабораторная работа №17. Шаблоны STL

Цель работы: Научиться работать с основными методами форм, управляющими отображением формы на экране. **Оборудование и материалы:**

для выполнения данного практического занятия необходим компьютер с установленной
операционной системой Windows 7 и программными продуктами: Visual Studio.

Указания по технике безопасности:

К выполнению практических занятий допускаются студенты, ознакомившиеся с правилами работы в лаборатории, прошедшие инструктаж безопасности.

Задания: для выполнения практической работы необходимо выполнить следующее:

1. Изучить рекомендуемую литературу.
2. Выполнить практическую работу.
3. Ответить на контрольные вопросы.
4. Оформить отчет.

Содержание отчета: отчет по практическому занятию должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman, интервал – полуторный, поля левое – 3 см., правое – 1,5 см., верхнее и нижнее – 2 см. Абзацный отступ – 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой практической работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись (Рисунок 1 – Создание подсистемы) и ссылку на него в тексте.

Контрольные вопросы:

1. Провайдеры дополнительных свойств.
2. Назначение и виды меню.
3. Реализация в программе на C++/C# и Delphi.
4. Организация доступа к пунктам меню с помощью клавиш доступа и быстрых клавиш.
5. Управление доступностью пунктов меню. Создание нового меню во время выполнения программы.

Список литературы, рекомендуемый к использованию по данной теме:

1. Лафоре Р. Объектно-ориентированное программирование в C++. – М. – СПб.:

Питер, 2008 г. – 922 с.

2. Юров В. Ассемблер. Учебник для вузов. – СПб.: Питер, 2008.- 624 с.
3. Юров В. Ассемблер. Практикум. – СПб.: Питер, 2008. - 356 с.
4. Бен-Ари М. Языки программирования: Практический сравнительный анализ: Учебник. – М.: Мир. – 2007. 5. Молчанов А.Ю. Системное программное обеспечение: Лабораторный практикум. – М.; СПб; Нижн.Новг.; Минск, 2009.

Лабораторная работа №18. Введение в КОП. Объекты и классы в КОП

Цель работы: Ознакомиться **Оборудование и материалы:** для выполнения данного практического занятия необходим компьютер с установленной операционной системой Windows 7 и программными продуктами: Visual Studio. **Задания:**

для выполнения практической работы необходимо выполнить следующее:

1. Изучить рекомендуемую литературу.
2. Выполнить практическую работу.
3. Ответить на контрольные вопросы.
4. Оформить отчет.

Содержание отчета: отчет по практическому занятию должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman, интервал – полуторный, поля левое – 3 см., правое – 1,5 см., верхнее и нижнее – 2 см. Абзацный отступ – 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой практической работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись (Рисунок 1 – Создание подсистемы) и ссылку на него в тексте.

Контрольные вопросы:

1. Модальный режим формы.
2. События формы. Понятие события. События формы.
3. Работа с элементами управления и компонентами. **Список литературы, рекомендуемый к использованию по данной теме:**

1. Лафоре Р. Объектно-ориентированное программирование в С++. – М. – СПб.: Питер, 2008 г. – 922 с.
2. Юров В. Ассемблер. Учебник для вузов. – СПб.: Питер, 2008.- 624 с.
3. Юров В. Ассемблер. Практикум. – СПб.: Питер, 2008. - 356 с.
4. Бен-Ари М. Языки программирования: Практический сравнительный анализ: Учебник. – М.: Мир. – 2007.
5. Молчанов А.Ю. Системное программное обеспечение: Лабораторный практикум. – М.; СПб; Нижн.Новг.; Минск, 2009.

Лабораторная работа №19. Среды программирования, использующие компонентно-ориентированный подход

Цель работы: Научиться рисовать под управлением мыши.

Оборудование и материалы: для выполнения данного практического занятия необходим компьютер с установленной операционной системой Windows 7 и программными продуктами: Visual Studio.

Указания по технике безопасности:

К выполнению практических занятий допускаются студенты, ознакомившиеся с правилами работы в лаборатории, прошедшие инструктаж безопасности.

Задания: для выполнения практической работы необходимо выполнить следующее:

1. Изучить рекомендуемую литературу.
2. Выполнить практическую работу.
3. Ответить на контрольные вопросы.
4. Оформить отчет.

Содержание отчета: отчет по практическому занятию должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman, интервал – полуторный, поля левое – 3 см., правое – 1,5 см., верхнее и нижнее – 2 см. Абзацный отступ – 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой практической работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе. В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись (Рисунок 1 – Создание подсистемы) и ссылку на него в тексте. **Контрольные вопросы:**

1. Векторная графика
2. Цвет пикселя

Список литературы, рекомендуемый к использованию по данной теме:

1. Лафоре Р. Объектно-ориентированное программирование в C++. – М. – СПб.:
Питер, 2008 г. – 922 с.
2. Юров В. Ассемблер. Учебник для вузов. – СПб.: Питер, 2008.- 624 с.
3. Юров В. Ассемблер. Практикум. – СПб.: Питер, 2008. - 356 с.
4. Бен-Ари М. Языки программирования: Практический сравнительный анализ: Учебник. – М.: Мир. – 2007.
5. Молчанов А.Ю. Системное программное обеспечение: Лабораторный практикум.

Лабораторная работа №20. Разработка двухоконного Windows-приложения

Цель работы: Научиться сохранять и загружать данные файлов с использованием диалогов.

Оборудование и материалы: для выполнения данного практического занятия необходим компьютер с установленной
операционной системой Windows 7 и программными продуктами: Visual Studio.

Указания по технике безопасности:

К выполнению практических занятий допускаются студенты, ознакомившиеся с правилами работы в лаборатории, прошедшие инструктаж безопасности.

Задания: для выполнения практической работы необходимо выполнить следующее:

1. Изучить рекомендуемую литературу.
2. Выполнить практическую работу.
3. Ответить на контрольные вопросы.
4. Оформить отчет.

Содержание отчета: отчет по практическому занятию должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman, интервал – полуторный, поля левое – 3 см., правое – 1,5 см., верхнее и нижнее – 2 см. Абзацный отступ – 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой практической работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись (Рисунок 1 – Создание подсистемы) и ссылку на него в тексте.

Контрольные вопросы:

1. Загрузка и сохранение изображений
2. Диалог загрузки изображений
3. Диалог сохранения изображений

Список литературы, рекомендуемый к использованию по данной теме:

1. Лафоре Р. Объектно-ориентированное программирование в C++. – М. – СПб.: Питер, 2008 г. – 922 с.
2. Юров В. Ассемблер. Учебник для вузов. – СПб.: Питер, 2008.- 624 с.
3. Юров В. Ассемблер. Практикум. – СПб.: Питер, 2008. - 356 с.
4. Бен-Ари М. Языки программирования: Практический сравнительный анализ: Учебник. – М.: Мир. – 2007.
5. Молчанов А.Ю. Системное программное обеспечение: Лабораторный практикум.

Лабораторная работа №21. Основные методы формы, управляющие отображением формы на экране

Цель работы: Научиться работать с элементами управления компонентами.

Оборудование и материалы: для выполнения данного практического занятия необходим компьютер с установленной
операционной системой Windows 7 и программными продуктами: Visual Studio.

Указания по технике безопасности:

К выполнению практических занятий допускаются студенты, ознакомившиеся с правилами работы в лаборатории, прошедшие инструктаж безопасности.

Задания: для выполнения практической работы необходимо выполнить следующее:

1. Изучить рекомендуемую литературу.
2. Выполнить практическую работу.
3. Ответить на контрольные вопросы.
4. Оформить отчет.

Содержание отчета: отчет по практическому занятию должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman, интервал – полуторный, поля левое – 3 см., правое – 1,5 см., верхнее и нижнее – 2 см. Абзацный отступ – 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой практической работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись (Рисунок 1 – Создание подсистемы) и ссылку на него в тексте.

Контрольные вопросы:

1. Свойства TreeView
2. Обработчик событий, срабатывающий при выборе одного из узлов дерева
3. Свойства dataGridView

Список литературы, рекомендуемый к использованию по данной теме:

1. Лафоре Р. Объектно-ориентированное программирование в C++. – М. – СПб.:
Питер, 2008 г. – 922 с.
2. Юров В. Ассемблер. Учебник для вузов. – СПб.: Питер, 2008.- 624 с.
3. Юров В. Ассемблер. Практикум. – СПб.: Питер, 2008. - 356 с.
4. Бен-Ари М. Языки программирования: Практический сравнительный анализ: Учебник. – М.: Мир. – 2007.
5. Молчанов А.Ю. Системное программное обеспечение: Лабораторный практикум. – М.; СПб; Нижн.Новг.; Минск, 2009.

Лабораторная работа №22. Основные методы формы, управляющие отображением формы на экране

Цель работы: Научиться управлять графическими атрибутами.

Оборудование и материалы: для выполнения данного практического занятия необходим компьютер с установленной операционной системой Windows 7 и программными продуктами: Visual Studio.

Указания по технике безопасности:

К выполнению практических занятий допускаются студенты, ознакомившиеся с правилами работы в лаборатории, прошедшие инструктаж безопасности.

Задания: для выполнения практической работы необходимо выполнить следующее:

1. Изучить рекомендуемую литературу.
2. Выполнить практическую работу.
3. Ответить на контрольные вопросы.
4. Оформить отчет.

Содержание отчета: отчет по практическому занятию должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman, интервал – полуторный, поля левое – 3 см., правое – 1,5 см., верхнее и нижнее – 2 см. Абзацный отступ – 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой практической работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись (Рисунок 1 – Создание подсистемы) и ссылку на него в тексте.

Контрольные вопросы:

1. Элементы управления для изменения числовых характеристик
2. Преобразование типов

Список литературы, рекомендуемый к использованию по данной теме:

1. Лафоре Р. Объектно-ориентированное программирование в C++. – М. – СПб.:
Питер, 2008 г. – 922 с.
2. Юров В. Ассемблер. Учебник для вузов. – СПб.: Питер, 2008.- 624 с.
3. Юров В. Ассемблер. Практикум. – СПб.: Питер, 2008. - 356 с.
4. Бен-Ари М. Языки программирования: Практический сравнительный анализ: Учебник. – М.: Мир. – 2007. 5. Молчанов А.Ю. Системное программное обеспечение: Лабораторный практикум.
– М.; СПб; Нижн.Новг.; Минск, 2009.

Лабораторная работа №23. Рисование под управлением мыши

Цель работы: Ознакомиться с понятием многофайловые приложения.

Оборудование и материалы: для выполнения данного практического занятия необходим компьютер с установленной
операционной системой Windows 7 и программными продуктами: Visual Studio.

Указания по технике безопасности:

К выполнению практических занятий допускаются студенты, ознакомившиеся с правилами работы в лаборатории, прошедшие инструктаж безопасности.

Задания: для выполнения практической работы необходимо выполнить следующее:

1. Изучить рекомендуемую литературу.
2. Выполнить практическую работу.
3. Ответить на контрольные вопросы.
4. Оформить отчет.

Содержание отчета: отчет по практическому занятию должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman, интервал – полуторный, поля левое – 3 см., правое – 1,5 см., верхнее и нижнее – 2 см. Абзацный отступ – 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой практической работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись и ссылку на него в тексте.

Контрольные вопросы:

1. Web-приложения
2. Расширяемый язык разметки XML 3. Платформа Java 2 Enterprise Edition 4. Связь. Именование. Пример.

Список литературы, рекомендуемый к использованию по данной теме:

1. Лафоре Р. Объектно-ориентированное программирование в С++. – М. – СПб.: Питер, 2008 г. – 922 с.
2. Юров В. Ассемблер. Учебник для вузов. – СПб.: Питер, 2008.- 624 с.
3. Юров В. Ассемблер. Практикум. – СПб.: Питер, 2008. - 356 с.
4. Бен-Ари М. Языки программирования: Практический сравнительный анализ: Учебник. – М.: Мир. – 2007.
5. Молчанов А.Ю. Системное программное обеспечение: Лабораторный практикум.

Лабораторная работа №24. Сохранение (загрузка) данных файлов с использованием диалогов

Цель работы: Ознакомиться с основными понятиями низкоуровневого программирования.

Оборудование и материалы:

для выполнения данного практического занятия необходим компьютер с установленной операционной системой Windows 7 и программными продуктами: Visual Studio.

Указания по технике безопасности:

К выполнению практических занятий допускаются студенты, ознакомившиеся с правилами работы в лаборатории, прошедшие инструктаж безопасности.

Задания: для выполнения практической работы необходимо выполнить следующее:

1. Изучить рекомендуемую литературу.
2. Выполнить практическую работу.
3. Ответить на контрольные вопросы.
4. Оформить отчет.

Содержание отчета: отчет по практическому занятию должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию:

Шрифт TimesNewRoman, интервал – полуторный, поля левое – 3 см., правое – 1,5 см., верхнее и нижнее – 2 см. Абзацный отступ – 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой практической работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись (Рисунок 1 – Создание подсистемы) и ссылку на него в тексте.

Контрольные вопросы:

1. Структура языка, основные группы команд.
2. Синтаксис машинных команд.

Список литературы, рекомендуемый к использованию по данной теме:

1. Лафоре Р. Объектно-ориентированное программирование в С++. – М. – СПб.:
Питер, 2008 г. – 922 с.
2. Юров В. Ассемблер. Учебник для вузов. – СПб.: Питер, 2008.- 624 с.
3. Юров В. Ассемблер. Практикум. – СПб.: Питер, 2008. - 356 с.
4. Бен-Ари М. Языки программирования: Практический сравнительный анализ: Учебник. – М.: Мир. – 2007. 5. Молчанов А.Ю. Системное программное обеспечение: Лабораторный практикум. – М.; СПб; Нижн.Новг.; Минск, 2009.

Лабораторная работа №25. Разработка распределенного ПО

Цель работы: Ознакомиться с общей характеристикой языка Ассемблера.

Оборудование и материалы: для выполнения данного практического занятия необходим компьютер с установленной
операционной системой Windows 7 и программными продуктами: Visual Studio.

Указания по технике безопасности:

К выполнению практических занятий допускаются студенты, ознакомившиеся с правилами работы в лаборатории, прошедшие инструктаж безопасности.

Задания: для выполнения практической работы необходимо выполнить следующее:

1. Изучить рекомендуемую литературу.
2. Выполнить практическую работу.
3. Ответить на контрольные вопросы.
4. Оформить отчет.

Содержание отчета: отчет по практическому занятию должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman, интервал – полуторный, поля левое – 3 см., правое – 1,5 см., верхнее и нижнее – 2 см. Абзацный отступ – 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой практической работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание

к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись (Рисунок 1 – Создание подсистемы) и ссылку на него в тексте.

Контрольные вопросы:

1. Представление данных
2. Описание данных

Список литературы, рекомендуемый к использованию по данной теме:

1. Лафоре Р. Объектно-ориентированное программирование в С++. – М. – СПб.: Питер, 2008 г. – 922 с.
2. Юров В. Ассемблер. Учебник для вузов. – СПб.: Питер, 2008.- 624 с.
3. Юров В. Ассемблер. Практикум. – СПб.: Питер, 2008. - 356 с.
4. Бен-Ари М. Языки программирования: Практический сравнительный анализ: Учебник. – М.: Мир. – 2007.
5. Молчанов А.Ю. Системное программное обеспечение: Лабораторный практикум. – М.; СПб; Нижн.Новг.; Минск, 2009.

Лабораторная работа №26. Управление графическими атрибутами

Цель работы: Ознакомиться с общей характеристикой языка Ассемблера.

Оборудование и материалы:

для выполнения данного практического занятия необходим компьютер с установленной
операционной системой Windows 7 и программными продуктами: Visual Studio.

Указания по технике безопасности:

К выполнению практических занятий допускаются студенты, ознакомившиеся с правилами работы в лаборатории, прошедшие инструктаж безопасности.

Задания: для выполнения практической работы необходимо выполнить следующее:

1. Изучить рекомендуемую литературу.
2. Выполнить практическую работу.
3. Ответить на контрольные вопросы.
4. Оформить отчет.

Содержание отчета: отчет по практическому занятию должен быть выполнен в редакторе MS Word и оформлен согласно требованиям. Требования по форматированию: Шрифт TimesNewRoman, интервал – полуторный, поля левое – 3 см., правое – 1,5 см., верхнее и нижнее – 2 см. Абзацный отступ – 1,25. Текст должен быть выравнен по ширине.

Отчет должен содержать титульный лист с темой практической работы, цель работы и описанный процесс выполнения вашей работы. В конце отчеты приводятся выводы о проделанной работе.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним. Каждый рисунок должен располагаться по центру страницы, иметь подпись (Рисунок 1 – Создание подсистемы) и ссылку на него в тексте.

Контрольные вопросы:

1. Принципы выделения памяти под хранение переменных.

2. Регистры **Список литературы, рекомендуемый к использованию по данной теме:**

1. Лафоре Р. Объектно-ориентированное программирование в С++. – М. – СПб.: Питер, 2008 г. – 922 с.
2. Юров В. Ассемблер. Учебник для вузов. – СПб.: Питер, 2008.- 624 с.
3. Юров В. Ассемблер. Практикум. – СПб.: Питер, 2008. - 356 с.
4. Бен-Ари М. Языки программирования: Практический сравнительный анализ:

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Методические указания
по выполнению практических
работ
по дисциплине «ЯЗЫКИ ПРОГРАММИРОВАНИЯ»
для студентов специальности 10.05.01 Компьютерная безопасность
специализации «Информационно-аналитическая и техническая экспертиза компьютерных систем»

Ставрополь,
2023

Содержание

Практическое занятие №1 Введение в языки программирования. Общие принципы построения и использования языков программирования. Язык программирования C++

Цель работы: изучение основных типов данных, способов описания переменных различных типов, операторов присваивания и организации ввода – вывода.

Краткие теоретические сведения:

Алгоритм – это последовательность действий, выполняемых по строго определенным правилам, однозначно определяющая процесс решения задачи и заведомо приводящая к её решению за некоторое количество шагов. Алгоритмизация – разработка формального метода решения практической задачи с возможностью реализации в виде программы для ЭВМ.

Изображение алгоритма в виде схемы выполняется в соответствии с ГОСТ 19.701–90 Единой системы программной документации «Схемы алгоритмов, программ, данных и систем».

Структура программ для Microsoft Visual Studio.

```
// struct_program.cpp: определяет точку входа для консольного приложения.
#include "stdafx.h"
//здесь подключаем все необходимые препроцессорные директивы
int main() { // начало главной функции с именем main //здесь будет находится ваш программный код
}
```

В строке 1 говорится о точке входа для консольного приложения, это значит, что данную программу можно запустить через командную строку Windows указав имя программы, к примеру, такое struct_program.cpp. Строка 1 является однострочным комментарием, так как начинается с символов //. В строке 2 подключен заголовочный файл "stdafx.h". Данный файл похож на контейнер, так как в нем подключены основные препроцессорные директивы (те, что подключил компилятор, при создании консольного приложения), и вспомогательные (подключенные программистом).

include — директива препроцессора, т. е. сообщение препроцессору. Строки, начинающиеся с символа # обрабатываются препроцессором до компиляции программы.

заголовочные файлы:

Библиотека cmath определяет набор функций для выполнения общих математических операций и преобразований. Математические функции:

1) тригонометрические функции:

cos-вычисление косинуса угла, переведенного в радианы; sin-вычисление синуса угла, переведенного в радианы; tan-вычисление тангенса угла, переведенного в радианы; acos-вычисление арккосинуса, результат будет в радианах; asin-вычисление арксинуса, результат будет в радианах; atan-вычисление арктангенса, возвращаемый результат будет в радианах; atan2-вычисление арктангенса и квадранта по координатам x и y, возвращаемый результат будет в радианах; 2) гиперболические функции:

cosh-вычисление гиперболического косинуса; sinh-вычисление гиперболического синуса; tanh-вычисление гиперболического тангенса;

3) экспоненциальные и логарифмические функции: exp-вычисление экспоненты;

frexp-получить мантиссу и показатель степени двойки;

ldexp-генерация числа по значению мантиссы и показателю степени; log-натуральный логарифм; log10-десятичный логарифм;

modf-разделение вещественного значения на дробную и целую части; 4)
функции степени:
pow-возведение числа в степень. Пример использования функции; sqrt-корень квадратный;
5) округление, модуль и другие функции ceil-округление до наименьшего целого значения;
fabs-вычислить модуль значения; floor-округление до наибольшего целого значения; fmod-
остаток от деления числителя на
знаменатель.

Пример использования функций:

```
#include <iostream> // для оператора cout #include <cmath>        // для функции pow

int main()
{
std::cout << "5.0 ^ 4 = "        << pow (5.0, 4)          <<   std::endl;   std::cout   <<
"2.77 ^ 9 = "   << pow (2.77, 9)        << std::endl;
std::cout << "12.01 ^ 11.54 = " << pow (12.01, 11.54) << std::endl; std::cout << "sqrt(" << param
<< ") = "
<< sqrt(param)                                    // вычисляем корень квадратный
<< std::endl;
double param = 60.0;                        // угол 60 градусов std::cout << "Косинус "
     << param
<< " градусов = " << cos(param * PI / 180) // вычисляем косинус угла,
//переведённого в радианы <<
std::endl; double param = 0.5;
std::cout << "Арксинус " <<
param
<< " = " << (asin (param) * 180.0 / PI) // вычисляем арксинус
<< " градусов " << std::endl; double val = 5.5, result;
result = log (val);                                // вычисляем натуральный логарифм std::cout <<
"ln(" << val << ") = "
<< result << std::endl; return 0;
}
```

Файлы кода C++ (с расширением .cpp) не являются единственными файлами в проектах и программах. Есть еще один тип файлов, который называется **заголовочный файл** (файл заголовка, подключаемый файл или header file**)**. Они имеют расширение .h, но иногда их можно увидеть и с расширением .hpp или вообще без расширения. Целью заголовочных файлов является удобное хранение предварительных объявлений для использования другими файлами. Всё содержимое из заголовочного файла копируется в файл *.cpp, т.е. всё содержимое становится доступным для использования. Заголовочные файлы:

— cstdio (stdio.h)-заголовочный файл для выполнения операций ввода/вывода;

— cstring (string.h)-заголовочный файл для работы со строками;

— iostream-заголовочный файл с классами, функциями и переменными для организации ввода-вывода. Для удобства в библиотеке определены три стандартных объекта-потока:

cin – объект класса istream, соответствующий стандартному вводу. В общем случае он позволяет читать данные с терминала пользователя; cout – объект класса ostream, соответствующий стандартному выводу. В общем случае он позволяет выводить данные на терминал пользователя; cerr – объект класса ostream, соответствующий стандартному выводу для ошибок. В этот

поток мы направляем сообщения об ошибках программы.

Вывод осуществляется, как правило, с помощью перегруженного оператора сдвига влево (<<), а ввод – с помощью оператора сдвига вправо (>>). Основные типы данных представлены в таблице 1.

Таблица 1 — Типы данных C++

Тип	айт	Диапазон принимаемых значений
целочисленный (логический) тип данных		
bool		0 / 255
целочисленный (символьный) тип данных		
char		0 / 255
целочисленные типы данных		
short int		-32 768 / 32 767
Unsigned short int		0 / 65 535
int		-2 147 483 648 / 2 147 483 647
unsigned int		0 / 4 294 967 295
long int		-2 147 483 648 / 2 147 483 647
Unsigned long int		0 / 4 294 967 295
типы данных с плавающей точкой		
float		-2 147 483 648.0 / 2 147 483 647.0
long float		-9 223 372 036 854 775 808 .0/ 9 223 372 036 854 775 807.0
double		-9 223 372 036 854 775 808 .0 / 9 223 372 036 854 775 807.0

Способы ввода данных в языке возможно двумя способами: форматированные ввод-вывод или потоковый.

При форматированном способе используются операторы ввода scanf вывода printf. Синтаксис операторов имеет вид:

scanf(<строка описания форматов> [, <список ввода>]); printf(<строка описания форматов> [, <список вывода>]);

Строка описания форматов состоит из обычных символов, специальных управляющих последовательностей символов и спецификаций формата. Обычные символы и управляющие последовательности просто копируются в стандартный выходной поток в порядке их появления. Спецификации формата начинаются с символа % и заканчиваются символом, определяющим тип выводимого значения. Кроме того, спецификации формата могут содержать символы и цифры для управления видом выводимого значения (подробно см. ниже). Список вывода состоит из

переменных и/или констант, значения которых должны быть выведены. Количество спецификаций формата должно быть равно количеству выводимых значений, которые указываются в списке вывода.

К управляющим последовательностям относятся последовательности символов, представленных в таблице 2.

Таблица 2 — Управляющие символы

Последовательность	Действие
\a	Звуковой сигнал
\b	Удаление предыдущего символа
\n	Новая строка
\r	Возврат каретки
\t	Горизонтальная табуляция
\v	Вертикальная табуляция
\'	Апостроф
\"	Кавычки
\\	Обратный слеш
ooo	ASCII символ в восьмеричной нотации
\x*ooo*	ASCII символ в шестнадцатеричной нотации

Иногда при работе операторов используются спецификаторы форматов (таблица 3-5).

Таблица 3 — Спецификаторы формата для оператора printf

Символ	Назначение
%c	символ
%d	целое десятичное число
%i	целое десятичное число
%e	десятичное число в виде x.xx e+xx
%E	десятичное число в виде x.xx E+xx
%f	десятичное число с плавающей запятой xx.xxxx
%F	десятичное число с плавающей запятой
	xx.xxxx
%g	%f или %e, что короче
%G	%F или %E, что короче
%o	восьмеричное число
%s	строка символов
%u	беззнаковое десятичное число
%x	шестнадцатеричное число
%X	шестнадцатеричное число
%%	символ %
%p	указатель
%n	указатель

Кроме того, к командам формата могут быть применены модификаторы l и h (табл. 4).

Таблица 4

Обозначение	Назначение
%ld	печать long int
%hu	печать short unsigned
%Lf	печать long double

Таблица 5 — Спецификаторы формата для оператора scanf

Символы	Назначение
%c	чтение символа
%d	чтение десятичного целого
%i	чтение десятичного целого
%e	чтение числа типа float (плавающая запятая)
%h	чтение short int
%o	чтение восьмеричного числа
%s	чтение строки
%x	чтение шестнадцатеричного числа
%p	чтение указателя
%n	чтение указателя в увеличенном формате

Примеры использования спецификаторов и модификаторов в операторах ввода-вывода:

int m, n, x; double y; char c = '&'; char
str[] = "String";
scanf("%d%d", &m, &n); // Ввод десятичных целых чисел в переменные m и n

printf("m = %5d\nn = %5d\n", m, n); // Вывод переменных m и n в десятичном целом формате, используются как минимум 5 знаков scanf("%d", &x); // Ввод
десятичного целого числа в переменную x
printf("%#010x\n", x); // Вывод переменной x в шестнадцатеричной системе, используются 10 знаков,
// впереди добавляются нули и символы 0x
scanf("%lf", &y); // Ввод вещественного числа в переменную y
printf("y = %7.2lf\n", y); // Вывод вещественной переменной, используются как минимум 7 знаков, из них 2 – после точки printf("c = %c\n", c); // Вывод одного
символа printf("%.4s\n", str);
} Составим программу на языке C++ для умножения двух
чисел. *//Подключение стандартных библиотек:*
#include<iostream>

```
#include <math.h> // для работы с мат.функциями acos, log10, log, atan using
namespace std; // пространство имен, где определяются идентификаторы
void main()                                  //заголовок главной функции программы
{                                        //операторные скобки
setlocale(0, “Russian”); //установка вывода сообщений на русском языке float x,y,z;
                                             //описание переменных (имя и тип) system(“cls”);
                                           //функция очистки экрана
cout<<"Введите числа x,y: "<<endl; //вывод сообщения на экран cin>>x>>y; //считать 2 вещ.числа, записать их по адресу перем. x,y z=x*y;
cout<<"Произведение="<<z;          //вывести результат на экран system(“pause”);
}                                                                  //конец главной функции
```

Пример выполнения задания**.**

Задание: вычислить значение функции

$$y = \frac{\sin^3 c \cdot \cos^2 a}{5\sin^d b} + 2$$

При $A = 9{,}5$; $B = 1{,}365$; $C = 6{,}5$; $D = 5$. Использовать два варианта ввода исходных данных и вывода результатов: возможности библиотеки функций языка С и библиотеки классов языка С++.

Схема программы для задания представлена на рисунке 3:

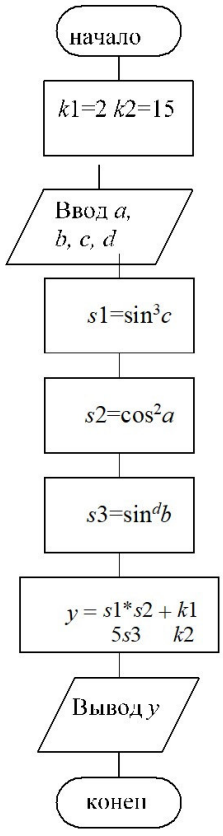


Рисунок 3

Текст программы с использованием библиотеки функций языка С (форматированный ввод-вывод) для ввода исходных данных и вывода результатов:

```
//Лабораторная работа 1
//с использованием библиотеки классов языка C++
// форматированный ввод-вывод # include <iostream>
# include <math.h> using namespace std; void main()
{
setlocale(0, “Russian”); //Вывод сообщений на русском языке const double k1=2.0; const
double k2= 15.0; double a, b, c, d, s1, s2, s3, y;
system(“cls”); //Очистка экрана printf(“Введите значения
переменных a, b, c, d:\n”); scanf(“%lf%lf%lf%lf”, &a, &b, &c, &d); s1=pow(sin(c), 3); s2=pow(cos(a),
2 );
s3=pow(sin(b), d ); y=(s1*s2)/(5*s3) + k1/k2; printf(“Искомое
значение y=%lf”, y); system(“pause”); //Пауза
}
```

Текст программы с использованием библиотеки классов языка C++ (потоковый ввод-вывод) для ввода исходных данных и вывода результатов:

```
//Лабораторная работа 1
//с использованием библиотеки классов языка C++
// потоковый ввод-вывод # include <iostream>
# include <math.h> using namespace std; void main()
{
setlocale(0, “Russian”); //Вывод сообщений на русском языке const double k1= 2.0; const
double k2= 15.0; double a, b, c, d, s1, s2, s3, y;
system(“cls”); //Очистка экрана cout<<“Введите значения
переменных a, b, c, d:”<<endl; cin>>a>>b>>c>>d; s1=pow(sin(c),
3);
s2=pow(cos(a), 2 );
s3=pow(sin(b), d ); y=(s1*s2)/(5*s3) + k1/k2; cout<<“Искомое значение y=”<<y;
system(“pause”); //Пауза
}
```

Контрольные вопросы

1. Структура программы на языке C++
2. Директивы препроцессора, заголовочные файлы, прототипы библиотечных функций, их вызовы.
3. Этапы обработки текста программы. Включение текстов из заголовочных файлов.
4. Главная функция программы. Структура функции, ее заголовок.
5. Определение переменных в программе. Типы переменных.
6. Функции форматного ввода-вывода в стиле C.
7. Ввод-вывод данных потоком в стиле C++.
8. Операции, выражения. Оператор присваивания.

Практическое занятие №2. Условные операторы if и switsh

Цель работы – изучение условного оператора и приобретение навыков программирования разветвляющихся алгоритмов.

Краткие теоретические сведения.

Синтаксис записи условного оператора if else:

— сокращенная запись: если условие истинно, т.е. выполняется, то выполняется и тело оператора выбора, иначе выполняется оператор, стоящий следом за if, т.е. оператор n.

```
if (/*проверяемое условие*/)
{
  /*тело оператора выбора 1*/;
}  оператор
n;
```

— полная запись: если проверяемое условие истинно, то выполняется тело оператора выбора 1, иначе, т. е. проверяемое условие ложно, выполняется тело оператора выбора 2.

```
if (/*проверяемое условие*/) {
/*тело оператора выбора 1*/;
} else
{
/*тело оператора выбора 2*/;
} Пример выполнения
задания.
```

Задание: по заданным координатам x и y определить, где находится точка (рисунок 6): внутри заштрихованной области; вне заштрихованной области; на границе этой области.

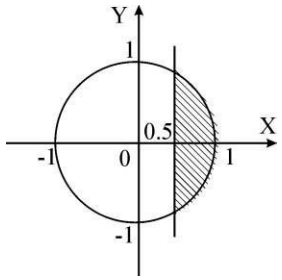


Рисунок 6 — Постановка задачи Метод решения задачи:

Для решения задачи будем использовать уравнение окружности $x^2+y^2=R^2$.

Так как R=1, то уравнение принимает вид $x^2+y^2=1$.

1. Определяем условие, при котором точка будет находиться внутри заштрихованной области: (x>0.5) и (x^2+y^2<1).
2. Определяем условие, при котором точка будет находиться вне заштрихованной области: (x<0.5) или (x^2+y^2>1). Текст программы:

```
#include <iostream> void main ()
{    double
x,y;
cout<<”Введите координаты точки: ”; cin>>x>>y; if ((x>0.5)&&(x*x+y*y<1)) cout<<”Точка
принадлежит области”; else if ((x<0.5)|
|(x*x+y*y>1)) cout<<”Точка не принадлежит области”; else cout<<”Точка лежит на границе
области”; }
```

Контрольные вопросы

1. В каких случаях используются условные операторы? Как изображаются условные операторы на схеме программы?
2. Условный оператор в языке С++. Форма записи. Правила выполнения.

3. Истинность и ложность выражений. Значение NULL.

4. Операции конъюнкции, дизъюнкции, отрицания. Знаки операций, их назначение. Какие знаки используются в операциях сравнения?

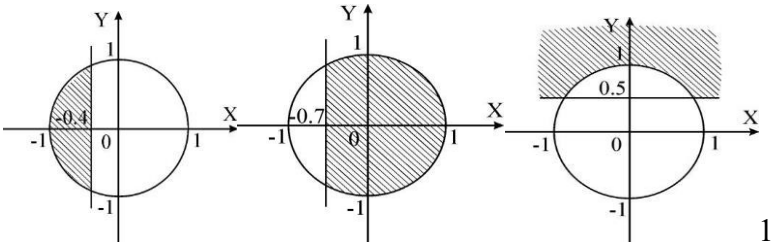
5. Использование составного оператора в языке C++. Отличие блока от составного оператора.

6. Вложенные операторы if–else.

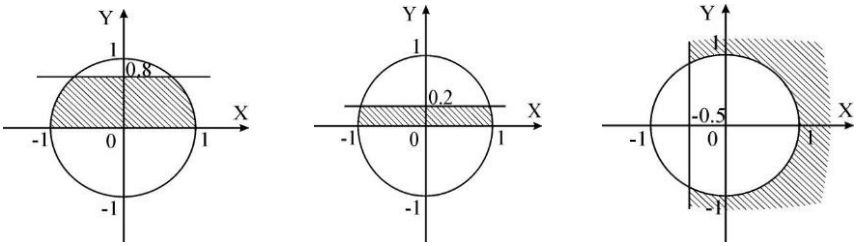
Варианты заданий

По заданным координатам точки определить, где находится точка:

1) внутри заштрихованной области; 2) вне заштрихованной области; 3) на границе этой области.

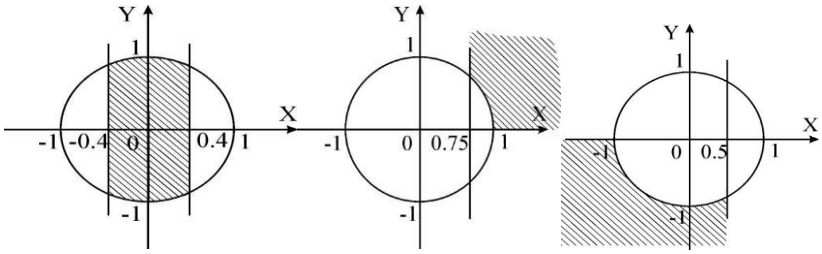


1. 2.



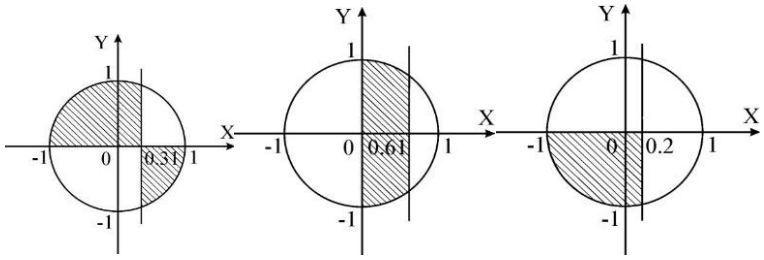
3.

4. 5. 6.



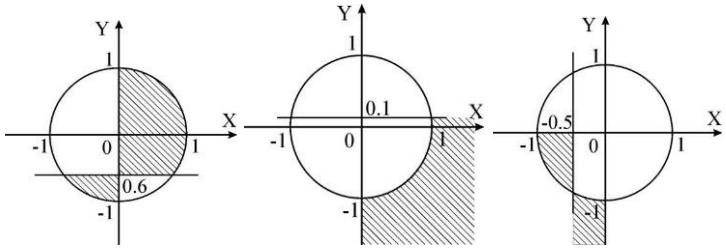
7. 8. 9.

11. 12. 13.

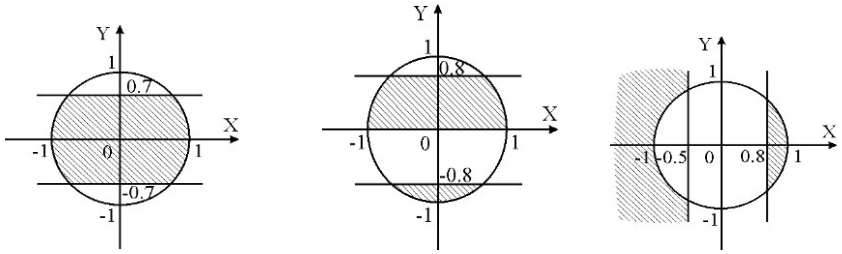


10.

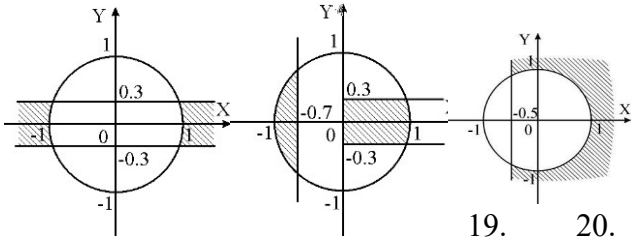
14. 15.



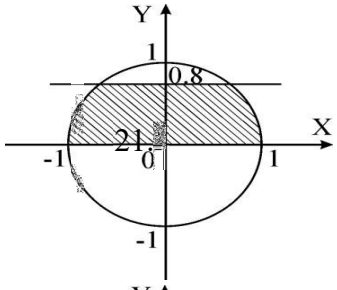
16.



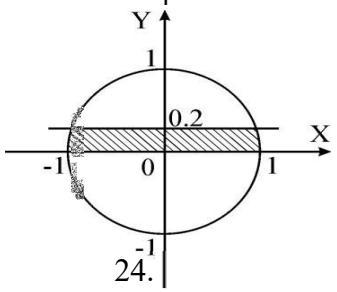
17. 18.



19. 20.



21.



24. 2

Практическое занятие №3 Программирование с использованием одномерных массивов

Цель работы: освоение средств языка C++ для описания итерационных циклов и закрепление навыков использования их при программировании.

Краткие теоретические сведения.

Иногда необходимо повторять одно и то же действие несколько раз подряд. Для этого используют циклы. Итерационные циклы — это циклы с заранее неизвестным числом шагов.

Различают циклы с пред и пост условием. Графически цикл с предусловием представлен на рисунке 8.

Цикл с предусловием будет выполняться, пока условие, указанное в круглых скобках будет истинным. Синтаксис цикла с предусловием:

```
while (Условие)
{
тело цикла;
}
```

Синтаксис цикла с постусловием:

```
// форма записи оператора цикла do while:
do // начало цикла do while {
/*блок операторов*/;
}
while (/*условие выполнения цикла*/); // конец
```

цикла do while

Цикл с постусловием do while отличается от цикла while тем, что в do while сначала выполняется тело цикла, а затем проверяется условие продолжения цикла. Из-за такой особенности do while называют циклом с постусловием. Таким образом, если условие do while заведомо ложное, то хотя бы один раз блок операторов в теле цикла do while выполнится. В итоге do while отличается от цикла while структурой. Если в while сначала выполняется проверка условия продолжения цикла, и если условие истинно, то только тогда выполняется тело цикла. Цикл do while работает с точностью до наоборот, сначала выполняется тело цикла, а потом проверяется условие, вот почему тело цикла do while, хотя бы раз, выполнится. Текст программы с использованием оператора **цикла с предусловием**:

```
//Программа вычисления sin x # include <iostream>
# include <math.h> void main ()
{
double x, eps;
cout<<”Введите значения аргумента и точности\n”;
cin>>x>>eps; double F=x, a=x; int n=2; while (fabs(a)>=eps)
{    a*=    –x*x/(n*(n+1));
F+=a;

n+=2;
}
cout<<”Приближенное значение sin x=”<<F<<endl; cout<<”Точное значение sin
x=”<<sin(x);
}
```

Текст программы с использованием оператора **цикла с постусловием**:

```
//Программа вычисления sin x # include <iostream.h>
# include <math.h> void main ()
{
```

```
double x, eps;
cout<<”Введите значения аргумента и точности\n”;
cin>>x>>eps; double F=x, a=x; int n=2; do {
a*= –x*x/(n*(n+1)); F+=a; n+=2;
}
while (fabs(a)>=eps)
cout<<”Приближенное значение sin x=”<<F<<endl; cout<<”Точное значение sin x=”<<sin(x);
}
```

Контрольные вопросы

1. Какие операторы языка С++ используются для организации итерационных циклов?
2. Синтаксис оператора цикла с предусловием.
3. Как выполняется оператор цикла с предусловием?
4. Синтаксис оператора цикла с постусловием.
5. Как выполняется оператор цикла с постусловием?
6. Чем отличаются операторы цикла с предусловием и с постусловием?
7. В каких случаях в операторе цикла используется составной оператор или блок?

Практическое занятие №4. Программирование с использованием многомерных массивов

Цель работы: освоение средств языка С++ для описания циклов с параметром и закрепление навыков использования их при программировании, изучение способов описания, ввода-вывода и обработки одномерных массивов.

Краткие теоретические сведения.

Цикл с параметром используется в том случае если известно число шагов обработки, т.е. если известно число итераций. Итерацией цикла называется один проход цикла.

Синтаксис оператора цикла имеет вид: for (действие до начала цикла; условие продолжения

цикла; действия в конце каждой итерации цикла) { инструкция

цикла; инструкция цикла 2; инструкция цикла N;

} Существует частный случай этой записи: for (счетчик = значение; счетчик < значение; шаг цикла)

{ тело цикла;

}

Описание синтаксиса:

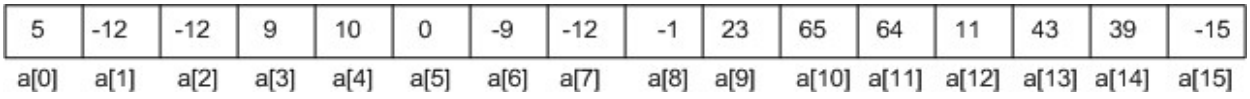
1) сначала присваивается первоначальное значение счетчику;

2) затем задается конечное значение счетчика цикла. Счетчик цикла — это переменная, в которой хранится количество проходов данного цикла. После того, как значение счетчика достигнет указанного предела, цикл завершится;

3) задаем шаг цикла. Шаг цикла — это значение, на которое будет увеличиваться или уменьшаться счетчик цикла при каждом проходе.

Одномерный массив — массив, с одним параметром, характеризующим количество элементов одномерного массива. Фактически одномерный массив — это массив, у которого

может быть только одна строка, и n-е количество столбцов. Столбцы в одномерном массиве — это элементы массива. На рисунке 12 показана структура целочисленного одномерного массива *a*. Размер этого массива — 16 ячеек.



Максимальный индекс одномерного массива *a* равен 15, но размер массива 16 ячеек, потому что нумерация ячеек массива всегда начинается с 0. Индекс ячейки – это целое неотрицательное число, по которому можно обращаться к каждой ячейке массива и выполнять какие-либо действия над ней (ячейкой). Синтаксис объявления одномерного массива в C++:

тип данных имя одномерного массива [размерность одномерного массива];
//пример объявления одномерного массива, изображенного на рисунке 12:
int a[16];

где, int — целочисленный тип данных; a — имя одномерного массива; 16 — размер одномерного массива, 16 ячеек.

Всегда сразу после имени массива идут квадратные скобочки, в которых задаётся размер одномерного массива, этим массив и отличается от всех остальных переменных.

Рассмотрим на примерах способы ввода-вывода одномерного массива:

1) размер массива можно не указывать только при его инициализации, при обычном объявлении массива обязательно нужно указывать размер массива

#include "stdafx.h" #include <iostream> using namespace std;

int main(int argc, char* argv[])
{
cout << "obrabotka massiva" << endl;
// объявление и инициализация одномерного массива
int array1[16] = { 5, -12, -12, 9, 10, 0, -9, -12, -1, 23, 65, 64, 11, 43, 39, -15 };
cout << "indeks" << "\t\t" << "element massiva" << endl; // печать заголовков for (int counter = 0; counter < 16; counter++) //начало цикла
{
//вывод на экран индекса ячейки массива, а затем содержимого этой ячейки cout << "array1[" << counter << "]" << "\t\t" << array1[counter] << endl;
}
system("pause"); return 0;
}

2) программа должна последовательно считывать десять введённых чисел с клавиатуры. Все введённые числа просуммировать, результат вывести на экран. #include "stdafx.h"
#include <iostream> using namespace std;

int main(int argc, char* argv[])
{

```
int array1[10]; // объявляем целочисленный массив cout << "Enter elementi massiva: " << endl; int sum = 0;
for ( int counter = 0; counter < 10; counter++ ) // цикл для считывания чисел cin >> array1[counter]; // считываем вводимые с клавиатуры числа cout
<< "array1 = {";
for ( int counter = 0; counter < 10; counter++ ) // цикл для вывода элементов массива cout << array1[counter] << " "; // выводим элементы массива на стандартное устройство
вывода
for ( int counter = 0; counter < 10; counter++ ) // цикл для суммирования чисел массива sum += array1[counter]; // суммируем элементы массива cout << "}\nsum = " << sum << endl; system("pause"); return 0;
}
```

Пример выполнения задания.

Задание: произвести следующую обработку 15 целых чисел: подсчитать сумму чисел, входящих в диапазон [-5..5] и количество нечетных чисел.

```
#include <iostream>

void main()
{
int x,sum=0,i,kol=0; printf("Enter numbers\n"); for (i=1;i<=15;i++)
{
scanf("%d",&x);
if ((x>=-5)&&(x<=5)) sum+=x; if (x%2!=0) kol++;
}
printf("Summa v diapazone [-5,5]=%d\n", sum);
printf("Kolichestvo nechetnih=%d", kol); }
```

Пример программы с использованием одномерного массива

```
#include<iostream> void main()
{ int a[15],sum=0,i,kol=0; printf("Enter numbers\n");
for (i=0;i<15;i++) scanf("%d",&a[i]); for
(i=0;i<15;i++)
{
if ((a[i]>=-5)&&(a[i]<=5)) sum+=a[i]; if (a[i]%2!=0)
kol++; }
printf("Summa v diapazone [-5,5]=%d\n", sum);
printf("Kolichestvo nechetnih=%d", kol); }
```

Контрольные вопросы

1. Массивы в языке С++: понятие массива в языке С++, описание массива в программе, представление элементов массива в памяти, обращение к элементам массива.
2. Оператор цикла for в языке С++. Форма записи. Правила выполнения.

Варианты заданий

1. Произвести следующую обработку 15 целых чисел: найти количество отрицательных чисел, количество нулевых и подсчитать сумму положительных чисел.

2. Произвести следующую обработку 15 целых чисел: найти количество четных чисел, а нечетные числа, входящие в диапазон [1..11] возвести в квадрат.

3. Произвести следующую обработку 15 вещественных чисел: найти количество отрицательных чисел, а числа, входящие в диапазон [0..10] возвести в квадрат.

4. Произвести следующую обработку 10 вещественных чисел: найти количество чисел, больших или равных 1,5, и подсчитать сумму отрицательных чисел.

5. Произвести следующую обработку 10 вещественных чисел: найти количество чисел, равных нулю, и найти сумму чисел, входящих в диапазон [- 15..15].

6. Произвести следующую обработку 15 целых чисел: найти количество отрицательных чисел и подсчитать разность положительных чисел.

7. Произвести следующую обработку 15 вещественных чисел: найти среднее арифметическое положительных чисел и подсчитать количество чисел, входящих в диапазон [- 15..5].

8. Произвести следующую обработку 10 целых чисел: найти количество отрицательных чисел и подсчитать сумму положительных чисел, делящихся без остатка на 3.

9. Произвести следующую обработку 10 целых чисел: найти количество отрицательных чисел, а числа, входящие в диапазон [0..10], умножить на 10.

10. Произвести следующую обработку 10 целых чисел: найти количество отрицательных чисел, а числа, входящие в диапазон [0..10], умножить на 3.

11. Произвести следующую обработку 15 вещественных чисел: найти среднее арифметическое отрицательных чисел и подсчитать количество чисел, входящих в диапазон [0..5].

12. Произвести следующую обработку 15 вещественных чисел: найти среднее арифметическое нечетных чисел и подсчитать сумму чисел, входящих в диапазон [- 15..5].

13. Произвести следующую обработку 10 вещественных чисел: найти количество чисел, равных нулю, и найти синус чисел, входящих в диапазон [-15..15].

14. Произвести следующую обработку 10 целых чисел: подсчитать сумму положительных чисел и определить номера отрицательных чисел.

15. Произвести следующую обработку 15 вещественных чисел: найти количество отрицательных чисел и номера нулевых чисел.

16. Произвести следующую обработку 12 целых чисел: подсчитать количество чисел, делящихся без остатка на 5, и сумму чисел, входящих в диапазон [-5..5].

17. Произвести следующую обработку 10 вещественных чисел: подсчитать количество чисел, отличающихся от числа 3 не более чем на 0.5, и сумму отрицательных чисел.

18. Произвести следующую обработку 15 целых чисел: подсчитать количество нулевых чисел и вычислить квадраты чисел, входящих в диапазон [-5..5].

19. Произвести следующую обработку 12 целых чисел: подсчитать количество нечетных чисел и сумму отрицательных чисел.

20. Произвести следующую обработку 15 вещественных чисел: подсчитать количество чисел, отличающихся от заданного не более чем на 0.5, и сумму положительных чисел.

21. . Произвести следующую обработку 10 вещественных чисел: найти количество отрицательных чисел, находящихся в диапазоне от -5 до 5 и подсчитать сумму положительных чисел.

22. Произвести следующую обработку 15 целых чисел: найти количество чисел, входящих в диапазон [1..11] и каждое число возвести в квадрат.

23. Произвести следующую обработку 15 вещественных чисел: найти количество чисел, равных 0, а числа, входящие в диапазон [-1..1] возвести в куб.

24. Произвести следующую обработку 10 вещественных чисел: найти количество чисел, больших или равных 1,5, и подсчитать сумму отрицательных чисел, входящих в диапазон [-1..0].

25. Произвести следующую обработку 10 вещественных чисел: найти количество чисел, меньших 15, и найти произведение чисел, входящих в диапазон [10..15].

Практическое занятие №5. Функции

Цель работы: изучение способов описания, ввода-вывода и обработки двумерных массивов, использование указателей при работе с массивами.

Краткие теоретические сведения.

Двумерный массив — это обычная таблица, со строками и столбцами. Фактически двумерный массив — это одномерный массив одномерных массивов. Структура двумерного массива, с именем *a*, размером *m* на *n* показана ниже, где, *m* — количество строк двумерного массива; *n* — количество столбцов двумерного массива; *m* x *n* — количество элементов массива.

Наиболее распространенный синтаксис объявления двумерного массива: /*тип данных*/ /*имя массива*/[/*количество строк*/][/*количество столбцов*/];

a[0][0]	a[0][1]	a[0][2]	a[0][3]	. . .	a[0][n]
a[1][0]	a[1][1]	a[1][2]	a[1][3]	. . .	a[1][n]
a[2][0]	a[2][1]	a[2][2]	a[2][3]	. . .	a[2][n]
. . .	. . .	. . .	. . .	. . .	. . .
a[m][0]	a[m][1]	a[m][2]	a[m][3]	. . .	a[m][n]

Примеры инициализации двумерного массива:

— способ 1:

```
int arr[5][3];
```

— способ 2:

```
int arr[5][3] = { {4, 7, 8}, {9, 66, -1}, {5, -5, 0}, {3, -3, 30}, {1, 1, 1} };
```

В последнем случае представление массива и обращение к элементу массива имеет вид, показанный на рисунке 18,19.

4 a[0][0]	7 [0][1]	8 [0][2]
9 a[1][0]	66 [1][1]	-1 [1][2]
5 a[2][0]	-5 [2][1]	0 [2][2]
3 a[3][0]	-3 [3][1]	30 [3][2]
1 a[4][0]	1 [4][1]	1 [4][2]

Рисунок 18 — Двумерный массив в C++

Ввод массива с клавиатуры и его вывод на экран выполняется следующим образом: int m,n,i,j;
cout <<"Введите размеры матрицы:"<<endl;
cin>>m; cin>>n; int elem; int ar [m][n]; for (i=0; i<m;
i++)
{for (j=0; j<n; j++){ cout<<"Введите элемент:"<<endl; cin>>elem; ar
[i][j]=elem;
} }
for (i=0; i<m; i++)
{for (j=0; j<n; j++){ cout<<ar[i][j]<<endl;
} }
system("pause"); return 0;}

При работе с массивами можно использовать указатели. Указатель – переменная, значением которой является адрес ячейки памяти. То есть указатель ссылается на блок данных из области памяти, причём на самое его начало. Указатель может ссылаться на переменную или функцию. Для этого нужно знать адрес переменной или функции. Так вот, чтобы узнать адрес конкретной переменной в C++ существует унарная операция взятия адреса &. Такая операция извлекает адрес объявленных переменных для того, чтобы его присвоить указателю.

Указатели используются для передачи по ссылке данных, что намного ускоряет процесс обработки этих данных (в том случае, если объём данных большой), так как их не надо копировать, как при передаче по значению, то есть, используя имя переменной. В основном указатели используются для организации динамического распределения памяти, например при объявлении массива, не надо будет его ограничивать в размере. Ведь программист заранее не может знать, какого размера нужен массив тому или иному пользователю, в таком случае используется динамическое выделение памяти под массив. Любой указатель необходимо объявить перед использованием, как и любую переменную:

/*тип данных*/ * /*имя указателя*/; Работу с указателями можно представить следующим образом:

#include "stdafx.h" #include <iostream>

using namespace std;

int main(int argc, char* argv[])

{

int var = 123; // инициализация переменной var числом 123

int *ptrvar = &var; // указатель на переменную var (присвоили адрес переменной указателю) cout << "&var = " << &var << endl;// адрес переменной var содержащийся в памяти, извлечённый операцией взятия адреса

cout << "ptrvar = " << ptrvar << endl;// адрес переменной var, является значением указателя ptrvar cout << "var = " << var << endl; // значение в переменной var cout << "*ptrvar = " << *ptrvar << endl; // вывод значения содержащегося в переменной var через указатель, операцией разименования указателя

system("pause"); return 0;
}

Пример выполнения задания.

Задание. Найти максимальную сумму элементов строк матрицы 3×5. Написать программы без использования указателей и с использованием указателей.

Текст программы:

```
#include <stdio.h> void main()
{
int a[3][5], i, j, s, max;
printf (“Введите 3 строки по 5 чисел”); for (i=0;i<3;i++)
for (j=0;j<5;j++) scanf("%d",&a[i][j]); printf (“Матрица a :\n”); for (i=0; i<3; i++) {for
(j=0; j<5; j++) printf (“%5d”, a[i][j]); printf (“\n”);
}
for(i=0;i<3;i++)
{s=0;
for (j=0;j<5;j++)
s+=a[i][j]; if (i==0) max=s;
else if (max<s) max=s;
}  printf("Максимальная  сумма  строки  =
%d",max);
}
```

Схема программы с использованием указателей представлена на рисунке 21:

Продолжение рисунка 21. Пример программы с использованием указателей:

```
#include <stdio.h> void main()
{
int a[3][5], *P, i, j, s, max; printf (“Введите 3 строки по 5
чисел”); for (i=0;i<3;i++)
for (j=0;j<5;j++) scanf("%d",&a[i][j]); printf (“Матрица a :\n”); for (i=0; i<3; i++)
{for (j=0; j<5; j++) printf (“%5d”, a[i][j]); printf (“\n”);
} P=&a[0][0]; for(i=0;i<3;i++)
{s=0; for
(j=0;j<5;j++) {s+=*P;
P++; } if (i==0)
max=s; else if
(max<s) max=s;
}  printf("Максимальная  сумма  строки  =
%d",max);
}
```

Контрольные вопросы

1. Правила организации вложенных циклов.
2. Особенности организации двумерных массивов: понятие массива в языке С++, описание массива в программе, представление элементов массива в памяти, обращение к элементам массива.
3. Указатели в языке С++: понятие указателя, описание указателя в программе.
4. Операции над указателями.
5. Связь массивов и указателей.

Варианты заданий

1. Вычислить сумму положительных элементов каждого столбца матрицы A(m×n).

2. Из матрицы *X* (m×n) построить матрицу *Y*, поменяв местами строки и столбцы.

3. Найти наименьший элемент матрицы *X* (m×n) и записать нули в ту строку и столбец, где он находится.

4. Переписать первые элементы каждой строки матрицы *A* (m×n), большие *C*, в массив *B***.** Если в строке нет элемента, большего *C*, то записать ноль в массив *B*.

5. Дана действительная матрица размера *m*×*n*. Найти сумму наибольших значений элементов ее строк.

6. В данной действительной матрице размера m×n поменять местами строку, содержащую элемент с наибольшим значением, со строкой, содержащей элемент с наименьшим значением. Предполагается, что такой элемент единственный.

7. В данной действительной квадратной матрице порядка *n* найти сумму элементов строки, в которой расположен элемент с наименьшим значением. Предполагается, что такой элемент единственный.

8. Дана действительная матрица размера *m*×*n*, все элементы которой различны. В каждой строке выбирается элемент с наименьшим значением, затем среди этих чисел выбирается наибольшее. Указать индексы элемента с найденным значением.

9. Дана целочисленная матрица размера m×n. Найти матрицу, получающуюся перестановкой столбцов (первого с последним, второго с предпоследним и т.д.).

10. Дана целочисленная матрица размера m×n. Найти матрицу, получающуюся перестановкой строк (первой с последней и т.д.).

11. Дана действительная матрица [a_{ij}], где i, j = 1..*n.* Получить действительную матрицу [b_{ij}], где i, j = 1..*n,* элемент b_{ij} которой равен сумме элементов данной матрицы, расположенных в области, определяемой индексами i, j (область заштрихована на рисунке 22):

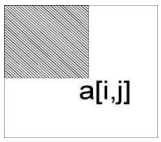


Рисунок 22

12. Дана действительная квадратная матрица порядка *n*. Преобразовать матрицу по правилу: строку с номером *n* сделать столбцом с номером *n,* а столбец с номером *n* сделать строкой с номером *n.*

13. Просуммировать элементы матрицы X (4,5), сумма индексов которых равна заданной константе K.

14. Дана матрица M (4×5). Вычислить вектор D, компоненты которого равны сумме элементов строк матрицы.

15. Дана матрица M (6×6). Вычислить сумму элементов главной диагонали.

16. Дана матрица N (6×5). Найти столбец с минимальной суммой элементов.

17. Дана матрица M (4×5) и константа C. Вычислить матрицу D, равную произведению элементов матрицы M на константу.

18. Дана матрица M (4×6). Вычислить вектор D, компоненты которого равны сумме элементов столбцов матрицы.

19. Дана действительная квадратная матрица порядка *n*, все элементы которой различны. Найти наибольший элемент, среди стоящих на главной и побочной диагоналях и поменять его местами с элементом, стоящим на пересечении этих диагоналей.

20. Дана действительная квадратная матрица порядка n. Найти наибольшее из значений элементов, расположенных в заштрихованной части матрицы (рисунок 23):

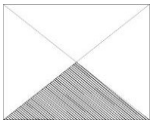


Рисунок 23

21. Дана матрица M (2×5), определить максимальный и минимальный элементы. Поменять местами максимальный и минимальный элементы.

22. В матрице A(n×n) вычислить сумму элементов матрицы (n-2×n-2) и определить максимальный элемент в ней.

23. Дана матрица M (6×6). Вычислить произведение элементов главной диагонали с константой C.

24. Дана матрица N (6×5). Найти строку с минимальной суммой элементов, а элемент с номером n_{ij} возвести в квадрат.

25. Дана матрица вещественных чисел A (m×n). В сроке m определить максимальный элемент, а в столбце n количество элементом, меньших порога k.

Практическое занятие №6. Структуры

Цель работы: изучение правил описания, ввода-вывода и основных функций обработки символьных данных.

Краткие теоретические сведения.

Строки в C++ позволяют работать с символьными данными. С помощью строк возможно осуществить чтение с клавиатуры текста, его обработка и вывод.

В C++ существует 2 типа строк: 1. Массив переменных типа char, заканчивающийся нуль-терминатором \0. Символьные строки состоят из набора символьных констант заключённых в двойные кавычки. При объявлении строкового массива необходимо учитывать наличие в конце строки нуль-терминатора, и отводить дополнительный байт под него.

char string[10]; // string – имя строковой переменной, 10 – размер массива, (в строке может поместиться 9 символов, последнее место отводится под нуль-терминатор) 2. Специальный класс string

Для его работы необходимо в начале программы подключить заголовочный файл string: *#include <string>*

Для создания строки необходимо в начале программы написать *using namespace std;*

Теперь чтоб создать строку достаточно написать: *string s;* Для записи в строку можно использовать оператор = s="Hello";

Доступ к i-му элементу строки **s** типа string осуществляется стандартным образом s[i]. Над строками типа string определены следующие операции:

- присваивания, например s1=s2;
- объединения строк (s1+=s2 или s1=s1+s2) — добавляет к строке s1 строку s2, результат храниться в строке s1, пример объединения строк:
- сравнения строк: s1=s2, s1!=s2, s1<s2, s1>s2, s1<=s2, s1>=s2 — результатом будет логическое значение.

Существует множество функций для работы со строками (таблица 7). Таблица 7 – Функции для работы со строками

Функция	Объяснение
s.append(str)	добавляет в конец строки строку str
s.assign(str)	присваивает строке s значение строки str
s.clear()	отчищает строку, т.е. удаляет все элементы в ней
s.compare(str)	сравнивает строку s со строкой str и возвращает 0 в случае совпадение
s.copy(C, I, N)	копирует из строки s в строку C(может быть как строка типа string, так и строка типа char) I символов, начиная с N-го символа
bool b=s.empty()	если строка пуста, возвращает true, иначе false
s.erase(I,N)	удаляет N элементов с I-го символа
s.find(str,I)	ищет строку str начиная с I- го символа
s.insert(pos, s1)	вставляет строку **s1** в строку **s**, начиная с позиции **pos**
int len=s.length()	записывает в len длину строки
s.push_back(symbol)	добавляет в конец строки символ
s.replace(index, n,str)	берет n первых символов из str и заменяет символы строки s на них, начиная с позиции index
str=s.substr(n,m)	возвращает m символов начиная с позиции n
s.swap(str)	меняет содержимое s и str местами.
s.size()	возвращает число элементов в строке.

Функции для работы со строками, прототипы которых описаны в заголовочном файле string.h:

1. Сравнение строк. Для сравнения строк используются функции strcmp и strncmp. Функция

int strcmp (const char *str1, const char *str2);

лексикографически сравнивает строки str1, str2 и возвращает –1, 0 или 1, если строка str1 соответственно меньше, равна или больше строки str2. Функция

int strncmp (const char *str1, const char *str2, size_t n); лексикографически сравнивает не более чем n первых символов из строк str1 и str2. Функция возвращает –1, 0 или 1, если первые n символов из строки str1 соответственно меньше, равны или больше первых n символов из строки str2.

Пример:

```
// пример сравнения строк #include #include int main() { char str1[] = "aa bb"; char str2[] = "aa aa"; char str3[] = "aa bb cc"; printf("%d\n", strcmp(str1, str3)); // печатает: -1 printf("%d\n", strcmp(str1, str1)); // печатает: -0 printf("%d\n", strcmp(str1, str2)); // печатает: 1 printf("%d\n", strncmp(str1, str3, 5)); // печатает: 0 return 1; }
```

2. Копирование строк. Для копирования строк используются функции strcpy и strncpy. Функция char *strcpy (char *str1, const char *str2);

копирует строку str2 в строку str1. Строка str2 копируется полностью, включая завершающий нулевой байт. Функция возвращает указатель на str1. Если строки перекрываются, то результат непредсказуем.

Функция

char *strncpy (char *str1, const char *str2, size_t n);

копирует n символов из строки str2 в строку str1. Если строка str2 содержит меньше чем n символов, то последний нулевой байт копируется столько раз, сколько нужно для расширения строки str2 до n символов. Функция возвращает указатель на строку str1.

Пример: char str1[80]; char str2 = "Copy string."; strcpy (str1, str2);
printf (str1); // печатает: Copy string.

4. Соединение строк. Для соединения строк в одну строку используются функции strcat и strncat. Функция char* strcat (char *str1, const char *str2);

присоединяет строку str2 к строке str1, причем завершающий нулевой байт строки str1 стирается. Функция возвращает указатель на строку str1.

Функция

char* strncat (char *str1, const char *str2, size_t n);

присоединяет n символов из строки str2 к строке str1, причем завершающий нулевой байт строки str1 стирается. Функция возвращает указатель на строку str1. если длина строки str2 меньше n, то присоединяются только символы, входящие в строку str2. После соединения строк к строке str1 всегда добавляется нулевой байт. Функция возвращает указатель на строку str1.

Пример:

```
#include <stdio.h>
#include <string.h> int main ( )
{
 char str1[80] = "String "; char str2 = "catenation "; char str3 = "Yes
No"; strcat ( str1, str2 ); printf ("%s\n", str1 ); // печатает: String catenation
strncat ( str1, str3, 3 ); printf ("%s\n", str1 ); // печатает: String catenation
Yes return
1;
}
```

5. Поиск символа в строке. Для поиска символа в строке используются функции strchr, strrchr, strspn, strcspn и strpbrk. Функция char*

strchr (const char *str, int c);

ищет первое вхождение символа, заданного параметром c, в строку str. В случае успеха функция возвращает указатель на первый найденный символ, а в случае неудачи – NULL.

Функция char* strrchr (const

char *str, int c);

ищет последнее вхождение символа, заданного параметром c, в строку str. В случае успеха функция возвращает указатель на последний найденный символ, а в случае неудачи – NULL.

Пример:

```
#include <stdio.h> #include <string.h> int main ( )
{
char str[80] = "Char search";
printf ("%s\n", strchr ( str, 'r' )); // печатает: r search printf
("%s\n", strrchr ( str, 'r' )); // печатает: rch return 1; }
```

Функция

size_t strspn (const char *str1, const char *str2);

возвращает индекс первого символа из строки str1, который не входит в строку str2.

Функция size_t strcspn (const char *str1, const char *str2); возвращает индекс первого символа из строки str1, который входит в строку str2. Пример:

```
int main ( )
{
char str[80] = "123 abc";
printf ("n = %d\n", strspn ( str, "321" ); // печатает: n = 3 printf
("n = %d\n", strcspn ( str, "cba" ); // печатает: n = 4 return 1; }
```

Функция

char* strpbrk (const char *str1, const char *str2); находит первый символ в строке str1, который равен одному из символов в строке str2.

В случае успеха функция возвращает указатель на этот символ, а в случае неудачи – NULL.

Пример. char str[80] = "123 abc";

```
printf ("%s\n", strpbrk ( str, "bca" )); // печатает: abc
```

6. Сравнение строк. Для сравнения строк используются функция strstr.

Функция

char* strstr (const char *str1, const char *str2);

находит первое вхождение строки str2 (без конечного нулевого байта) в строку str1. В случае успеха функция возвращает указатель на найденную подстроку, а в случае неудачи – NULL. Если указатель str1 указывает на строку нулевой длины, то функция возвращает указатель str1.

Пример: char str[80] = "123 abc 456;

```
printf ("%s\n", strstr ( str, "abc"); // печать: abc 456
```

7. Разбор строки на лексемы. Для разбора строки на лексемы используется функция strtok. Функция char* strtok (char *str1, const char *str2

);

возвращает указатель на следующую лексему (слово) в строке str1, в которой разделителями лексем являются символы из строки str2. В случае если лексемы закончились, то функция возвращает NULL. При первом вызове функции strtok параметр str1 должен указывать на строку, которая разбирается на лексемы, а при последующих вызовах этот параметр должен быть установлен в NULL. После нахождения лексемы функция strtok записывает после этой лексемы на место разделителя нулевой байт.

Пример.

```
#include <stdio.h> #include <string.h> int main( )
{ char str[ ] = "12 34 ab cd"; char
*p; p = strtok ( str, " " ); while (p)
{ printf ( "%s\n", p ); // печатает в столбик значения: 12 34 ab cd p =
strtok ( NULL, " " ); } return 1; }
```

8. Определение длины строки. Для определения длины строки используется функция strlen. Функция size_t strlen (const char *str);

возвращает длину строки, не учитывая последний нулевой байт.

Например, char str[] = "123"; printf ("len = %d\n", strlen (str)); // печатает: len = 3 Пример выполнения задания.

Задание. Найти слова во введенной с клавиатуры строке, вывести их на экран и подсчитать их количество.Пример программы:

```
#include<stdio.h> #include<string.h>

vid main()
{ char s[100],d[100]; int i=0,j=0,bw,ew,len; gets(s);
 len=strlen(s);
while (i<len)
 { while((s[i]==' ')&&(i<len)) i++; bw=i;
   while((s[i]!=' ')&&(i<len)) i++; ew=i;
   strncpy(d,&s[bw],ew-bw+1); d[ew-
   bw+1]=0; if (bw<len)
    { j++; printf("%s\n",d);}
 } printf("Vsego slov %d\n",
j); }
```

Контрольные вопросы

1. Строки в языке С++: понятие строки, описание строк в программе, обращение к элементам строки.
2. Три способа ввода строк в С++.
3. Три способа вывода строк в С++.
4. Способы инициализации строк (задание значений в программе).
5. Стандартные функции для обработки строк.

Варианты заданий

Дана последовательность символов S_1, …, S_N. Группы символов, разделенные пробелом (одним или несколькими) и не содержащие пробелов внутри себя, будем называть словами.

1. Определить число символов в самом длинном слове строки. Слова отделяются знаком “/”.
2. В произвольном тексте выделить и отпечатать слова, начинающиеся с буквы А.
3. В произвольном тексте вставить между первым и вторым словом новое слово.
4. В произвольном тексте найти и отпечатать слова, содержащие букву Е.
5. Отпечатать второе и третье слова произвольного текста.
6. В произвольном тексте вставить между вторым и третьим словом новое слово.
7. В произвольном тексте найти и отпечатать все слова длиной 5 символов.
8. В произвольном тексте найти самое короткое слово.
9. В последовательности из 10 пятибуквенных слов найти и поменять местами пару слов, у которых первые три буквы одного совпадают с последними тремя буквами другого.
10. Упорядочить в алфавитном порядке последовательность из 10 пятибуквенных слов.
11. В строке из 50 символов отдельные слова разделены пробелом. Упорядочить строку так, чтобы каждое следующее слово было не короче предыдущего.

12. Расположить слова строки в порядке, обратным исходному.

13. Подсчитать количество букв ‘а’ в последнем слове строки.

14. Найти количество слов, у которых первый и последний символы совпадают между собой.

15. Исключить из строки слова, расположенные между скобками (,). Сами скобки должны быть исключены.

16. В произвольном тексте найти и отпечатать слова, содержащие букву А.

17. Отпечатать первое и второе слова произвольного текста.

18. В произвольном тексте вставить после первого слова новое слово.

19. В произвольном тексте найти и отпечатать все слова длиной 4 символа.

20. В произвольном тексте найти самое длинное слово.

21. Выполнить сравнение двух строк s и d. Результат вывести в виде сообщения «идентичны» или «не идентичны».

22. Добавить в конец строки новое слово, длинною 5 символов, иначе выдать сообщение об ошибке.

23. Добавить в начало строки новое слово, начинающееся с буквы *а,* иначе, если слово начинается с другой буквы вывести сообщение о невозможности добавления.

24. Посчитать какое количество раз встречается буква n (задается при каждом выполнении алгоритма).

25. Проанализировать массив символов, состоящий из n символов. Если массив состоит из n-5 символов, добавить в конец набор символов rrrrr.

Практическое занятие №7. Указатели

Цель работы: изучение правил составления и использования функций в программах на C++.

Краткие теоретические сведения.

В практике программирования часто складываются ситуации, когда одну и ту же группу операторов, реализующих определённую цель, требуется повторить без изменений в нескольких местах программы. Для избавления от столь нерациональной траты времени была предложена концепция подпрограммы.

Подпрограмма — именованная, логически законченная группа операторов языка, которую можно вызвать для выполнения любое количество раз из различных мест программы. В языке C++ подпрограммы реализованы в виде функций.

Функция — это поименованный набор описаний и операторов, выполняющих определённую задачу. Функция может принимать параметры и возвращать значение.

Информация, передаваемая в функцию для обработки, называется параметром, а результат вычисления функции её значением. Обращение к функции называют вызовом.

Описание функции состоит из заголовка и тела функции:

```
тип имя_функции(список_переменных)
{
тело_функции
}
```

Заголовок функции содержит:

– тип возвращаемого функцией значения, он может быть любым; если функция не возвращает значения, указывают тип void;

– имя_функции;

– список_переменных — перечень передаваемых в функцию величин (аргументов), которые отделяются друг от друга запятыми; для каждой переменной из списка указывается тип и имя; если функция не имеет аргументов, то в скобках указывают либо тип void, либо ничего.

Тело функции представляет собой последовательность описаний и операторов, заключённых в фигурные скобки.

В общем виде структура программы на C++ может иметь вид: директивы компилятора

```
тип имя_1(список_переменных)
{
тело_функции_1;
}
тип имя_2(список_переменных)
{
тело_функции_2;
}
...
тип имя_n(список_переменных)
{
тело_функции_n;
} int main
(список_переменных)
{
//Тело функции может содержать операторы вызова
//функций имя_1, имя_2, ..., имя_n тело_основной_функции;
}
```

Однако допустима и другая форма записи программного кода : директивы компилятора

```
тип имя_1(список_переменных); тип имя_2(список_переменных);
...
тип имя_n(список_переменных); int main (список_переменных)
{
//Тело функции может содержать операторы вызова
// функций имя_1, имя_2, ..., имя_n тело_основной_функции;
}
тип имя_1(список_переменных)
{
тело_функции_1;
}
тип имя_2(список_переменных)
{
тело_функции_2;
}
...
```

```
тип имя_n(список_переменных)
{
тело_функции_n;
}
```

Здесь функции описаны после функции main(), однако до неё перечислены заголовки всех функций. Такого рода опережающие заголовки называют прототипами функций. Прототип указывает компилятору тип данных, возвращаемых функцией, тип переменных, выступающих в роли аргументов, и порядок их следования. Прототипы используются для проверки правильности вызова функций в основной программе.

Вызвать функцию можно в любом месте программы. Для вызова функции необходимо указать её имя и в круглых скобках, через запятую перечислить имена или значения аргументов, если таковые имеются:

```
имя_функции(список_переменных);
```

Рассмотрим пример. Создадим функцию f(), которая не имеет входных значений и не формирует результат. При вызове этой функции на экран выводится строка символов "С Новым Годом, ".

```
#include <iostream> using namespace std; void
f ( ) //Описание функции.
{ cout << "С Новым Годом,
";
}
int main ( )
{ f ( ); //Вызов функции. cout <<"Студент!" << endl; f ( ); //Вызов функции.
cout <<"Преподаватель!" << endl; f ( ); //Вызов функции.
cout <<"Народ!" << endl; }
```

Результатом работы программы будут три строки:

С Новым Годом, Студент!

С Новым Годом, Преподаватель! С Новым Годом, Народ!

Далее приведён пример программы, которая вычисляет значение выражения $\sin^2(\alpha)+\cos^2(\alpha)$ при заданном значении α. Здесь функция radian выполняет перевод градусной меры угла в радианную1.

```
#include <iostream> #include <math.h> #define PI 3.14159 using namespace std; double
radian ( int deg, int min, int sec )
{
return ( deg * PI/180+min* PI /180/60+ sec * PI /180/60/60);
}
int main ( )
{
int DEG, MIN, SEC; double RAD;
//Ввод данных.
cout<<" Input :"<<endl; //Величина угла: cout<<" DEG ="; cin >>DEG; //градусы, cout<<"
MIN ="; cin >>MIN; //минуты, cout<<" SEC ="; cin >>SEC; //секунды.
//Величина угла в радианах.
RAD=radian (DEG, MIN, SEC); //Вызов функции. cout << " Value in radian A="<<RAD <<
endl;
//Вычисление значения выражения и его вывод. cout << " sin (A) ^2+ cos (A) ^2=
"; cout << pow( sin (RAD), 2 )+pow( cos (RAD), 2 ) << endl; return 0;
```

}

Переменные, описанные внутри функции, а также переменные из списка аргументов, являются локальными. Например, если программа содержит пять разных функций, в каждой из которых описана переменная N, то для C++ это пять различных переменных. Область действия локальной переменной не выходит за рамки функции. Значения локальных переменных между вызовами одной и той же функции не сохраняются.

Переменные, определённые до объявления всех функций и доступные всем функциям, называют глобальными. В функции глобальную переменную можно отличить, если не описана локальная переменная с теми же именем. Глобальные переменные применяют для передачи данных между функциями, но это затрудняет отладку программы. Для обмена данными между функциями используют параметры функций и значения, возвращаемые функциями.

Пример выполнения задания.

Задание. Найти максимальную сумму элементов строк матрицы 3×5 с использованием функций.

Пример программы:

```
#include <stdio.h> const int m=3, n=5;
//функция ввода void
inparr(int a[m][n])
{
int i,j;
for (i=0;i<m;i++) for
(j=0;j<n;j++)
scanf("%d",&a[i][j]); }
//функция вывода void
outarr (int a[m][n])
{
int i,j; printf("Matrica:\n"); for (i=0; i<m; i++)
{
for (j=0; j<n; j++) printf("%5d", a[i][j]);
printf("\n"); } }
int processarr(int a[m][n])
{
int i,j,s,max; for(i=0;i<m;i++)
{ s=0;
for (j=0;j<n;j++)
s+=a[i][j]; if (i==0) max=s;
else if (max<s) max=s;
}
return max;
}
void main()
{
int b[m][n]; inparr(b); outarr(b);
printf("Maximalnaya summa stroki = %d", processarr(b)); }
```

Контрольные вопросы

1. Структура функции в языке C++, ее заголовок.

2. Вызов функции.
3. Способы передачи параметров.
4. Оператор return (2 формы записи).
5. Описание функции (прототип).

Практическое занятие №8. Реализация классов

Цели:

Изучение структуры класса, механизм создания и использования, описание членов- данных класса и методов доступа к ним.

1. Краткие теоретические сведения Компоненты класса

Класс - это определяемый пользователем тип. Описание класса очень похоже на описание структуры в Си. В этом смысле класс является расширением понятия структуры. В простейшем случае класс можно определить с помощью конструкции:

тип_класса имя_класса {список_членов_класса};

, где

тип_класса – одно из служебных слов **class, struct, union**;

имя_класса – идентификатор;

список_членов_класса – определения и описания типизированных данных и принадлежащих классу функций.

Функции – это методы класса, определяющие операции над объектом.

Данные – это поля объекта, образующие его структуру. Значения полей определяет состояние объекта.

Рассмотрим реализацию понятия даты с использованием struct для того, чтобы определить представление даты и множества функций для работы с переменными этого типа: struct date {

int month, day, year; // дата: месяц, день, год void set(int, int, int); void get(int*, int*, int*}; void next();

// ...

};

Функции, описанные таким образом, называются функциями-членами и могут вызываться только для специальной переменной соответствующего типа с использованием стандартного синтаксиса для доступа к членам структуры. Например:

```
date today;          // сегодня void f()
{
today.set(18,1,1985); today.next();
}
```

Поскольку разные структуры могут иметь функции члены с одинаковыми именами, при определении функции члена необходимо указывать имя структуры:

```
void date::next()
{
if ( ++day > 28 ) {
```

```
// делает сложную часть работы
…
}
}
```

В функции-члене имена членов структуры могут использоваться без явной ссылки на объект. В этом случае имя относится к члену того объекта, для которого функция была вызвана.

Описание date в предыдущем подразделе дает множество функций для работы с date, но не указывает, что эти функции должны быть единственными для доступа к объектам типа date. Это ограничение можно наложить, используя вместо struct class:

```
class date { int month, day, year; public:
void set(int, int, int); void next();
};
```

Методы доступа к полям (геттеры и сеттеры) class

```
TPen
{           private:
string FColor;  public:


};

string getColor ();
void setColor ( string newColor);
```

Получить значение:

```
string TPen::getColor () {   return FColor;          }   Записать   значение:   void
TPen::setColor ( string newColor )
{         if ( newColor.length()!= 6 )
FColor = "000000";
else

}

FColor = newColor;
```

Для описания объекта класса (экземпляра класса) используется конструкция ***имя_класса имя_объекта;***

Для изменения видимости компонент в определении класса можно использовать спецификаторы доступа: **public, private, protected**.

Общедоступные (public) компоненты класса доступны в любой части программы. Они могут использоваться любой функцией как внутри данного класса, так и вне его. Доступ извне осуществляется через имя объекта:

имя_объекта.имя_члена_класса ссылка_на_объект.имя_члена_класса
указатель_на_объект->имя_члена_класса

Собственные (private) компоненты класса локализованы в классе и не доступны извне. Они могут использоваться функциями – членами данного класса и функциями – “друзьями” того класса, в котором они описаны.

Защищенные (protected) компоненты доступны внутри класса и в производных классах.

В том, что доступ к структуре данных ограничен явно описанным списком функций, есть несколько преимуществ. Любая ошибка, которая приводит к тому, что дата принимает недопустимое значение (например, Декабрь 36, 1985), должна быть вызвана кодом функциичлена, поэтому первая стадия отладки, локализация, выполняется еще до того, как программа будет запущена.

Защита закрытых данных связана с ограничением использования имен членов класса. В функции-члене на члены объекта, для которого она была вызвана, можно ссылаться непосредственно. Например: class x { int m; public: int readm() { return m; }

}; x aa; x
bb; void
f(){

}

int a = aa.readm(); int b = bb.readm(); // ... В первом вызове члена readm() m относится к aa.m, а во втором - к bb.m.

Указатель на объект, для которого вызвана функция-член, является скрытым параметром функции. В каждой функции класса x указатель this неявно описан как x* this; и инициализирован так, что он указывает на объект, для которого была вызвана функция член. this не может быть описан явно, так как это ключевое слово.

2. Задание

2.1 Общее задание (50%)

1. Создайте консольный проект. В теле функции main будет выполняться демонстрация работы объекта.

2. Добавьте в проект новый класс и назовите этот класс Worker. В класс добавьте два общедоступных поля: имя и возраст и одно скрытое поле: вес:

class Worker { public:
int age; char* name; private:
float weight;
};

3. В теле функции main создайте объект класса Worker: Worker *wrk1 = new Worker (); wrk1->age = 34; wrk1->name = “Иванов”;

Добавьте оператор(функцию) вывода на экран созданного объекта.

4. Запустите программу на выполнение.

5. Попробуйте записать значение в поле weight. Почему данные не записались?

6. Для записи и чтения данных из скрытых полей используют методы. Добавим во внутрь класса Worker новый метод (действие) который будет отвечать за еду, если человек чегото там съест, то его вес должен будет увеличиться на количество съеденного.

в структуру класса: public:

…

void eat (float how_much);

после описания класса, но до функции main: void Worker::eat (float how_much){ weight = weight + how_much;
}

7. Если поле вес скрытое, то мы в него не только писать не можем, но и читать тоже не можем. Для чтения данных из скрытого поля необходимо использовать еще один метод:

в структуру класса: public:
…
float get_weight();

после описания класса, но до функции main: float Worker::get_weight(){ return weight; }

8. Почему в последних двух функциях после слова public идут различные слова? Что они обозначают и на что влияют?

9. Теперь эти два метода надо использовать в нашей программе. Заставьте рабочего съесть 2, а затем 3 кг пищи. Проверьте его вес.

wrk1->eat(2); wrk1->eat(3); float ves; ves = wrk1->get_weight(); Отобразите результат на экран.

10. Запустите программу на выполнение. Проверьте работоспособность. Добавьте комментарии.

11. Усовершенствуйте метод eat таким образом, что если рабочий за раз съедает более чем 10 кг, то его возраст увеличивается на год, а вес увеличивается только на половину от съеденного.

12. Попросите рабочего съесть 15 кг и посмотрите на результат работы программы.

13. Измените программу так, что бы имя рабочего и его первоначальный возраст вводились с клавиатуры и вносились в соответствующие переменные.

14. Запустите программу. Проверьте ее работоспособность.

15. Добавьте рабочему еще одно скрытое поле, которое будет отвечать за настроение и будет иметь первоначальное значение равное 10.

16. Добавьте три метода: гулять (метод должен увеличивать настроение на 1), танцевать (метод должен увеличивать настроение на 2) и работать (метод должен уменьшать настроение на 2).

17. Дополните основную программу так, что бы рабочий после еды два раза погулял и три раза потанцевал.

18. Добавьте в класс функцию, которая будет возвращать текущее настроение пользователя.

19. Добавьте в основную программу метод работать 9 раз (можно в цикле) и выведите настроение пользователя на экран.

20. Настроение получилось отрицательным? – ужасно. Измените метод работать таким образом, что бы настроение никогда не было меньше нуля (т.е. если настроение было 1 и человек поработал, то оно должно стать не меньше 0).

21. Проверьте заново работоспособность программы. **2.2 Индивидуальное задание (50%)**

- Реализовать пользовательский класс в соответствии с вариантом задания.
- При реализации классов поля должны быть скрытыми.

- Определить метод установки свойств (при недопустимых аргументах функции возвращать «false» и выдавать текст ошибки на экран).
- Определить метод чтения свойств.
- Написать демонстрационную программу, в которой показать использование объектов созданного класса.

3. Список рекомендуемой литературы

1. Объектно-Ориентированное программирование: учебник / Г. С. Иванова, Т. Н. Ничушкина ; под общ. ред. Г. С. Ивановой. — М. : Изд-во МГТУ им. Н. Э. Баумана, 2014. — 455.
2. Professional C++, 3rd Edition. Marc Gregoire. ISBN: 978-1-118-85805-9. Paperback 984 pages. September 2014

4. Контрольные вопросы

1. В определении класса члены класса с ключевым словом private доступны: а) любой функции программы;

б) в случае, если известен пароль; в) методам этого класса;

г) только открытым членам класса.

2. Напишите определение класса studentgroup, включающего одно закрытое поле типа int с именем number и одним открытым методом с прототипом voidadd( ).
3. Истинно ли следующее утверждение: поля класса должны быть закрытыми?
4. Для чего при работе с объектами применяется операция «точка»?
5. Для чего при работе с объектами применяется операция «стрелка»?
6. Методу класса всегда доступны данные:

а) объекта, членом которого он является; б) класса, членом которого он является

в) любого объекта класса, членом которого он является;

7. Что является единственным формальным различием между структурами и классами в C++?
8. Пусть определены три объекта класса. Сколько копий полей класса содержится в памяти? Сколько копий методов класса?
9. Для чего необходимо переопределять операции new и delete?
10. Что такое указатель this?

Практическое занятие №9. Конструкторы и деструкторы

Цели:

Изучение возможности инициализации объектов класса с помощью конструкторов и уничтожение их с помощью деструкторов

1. Краткие теоретические сведения

При создании объектов одной из наиболее широко используемых операций является инициализация элементов данных объекта. Единственным способом, с помощью которого вы можете обратиться к частным элементам данных, является использование функций класса. Чтобы упростить процесс инициализации элементов данных класса, C++ использует специальную функцию, называемую конструктором, которая запускается для каждого создаваемого вами объекта. Подобным образом C++ обеспечивает функцию, называемую

деструктором, которая запускается при уничтожении объекта. Основными концепциями конструктора и деструктора являются:

- Конструктор представляет собой метод класса, который облегчает инициализацию элементов данных класса.
- Конструктор имеет такое же имя, как и класс.
- Конструктор не имеет возвращаемого значения.
- Каждый раз, когда программа создает переменную класса, C++ вызывает конструктор класса, если конструктор существует.
- Многие объекты могут распределять память для хранения информации; когда вы уничтожаете такой объект, C++ будет вызывать специальный деструктор, который может освобождать эту память, очищая ее после объекта.
- Деструктор имеет такое же имя, как и класс, за исключением того, что вы должны предварять его имя символом тильды (~).
- Деструктор не имеет возвращаемого значения.

Представьте конструктор как функцию, которая помогает строить (конструировать) объект. Подобно этому, деструктор представляет собой функцию, которая помогает уничтожать объект. Деструктор обычно используется, если при уничтожении объекта нужно освободить память, которую занимал объект.

Создание простого конструктора

Конструктор представляет собой метод класса, который имеет такое же имя, как и класс. Например, если вы используете класс с именем employee, конструктор также будет иметь имя employee. Конструктор не возвращает никакого значения, несмотря на то, что он не объявляется как void.

```
class employee
{ public:
employee(char *, long, float); //Конструктор void
show_employee(void); int change_salary(float); long get_id(void);
private:
char name [64]; long employee_id; float salary;
};

employee::employee(char *name, long employee_id, float salary)
{
strcpy(employee::name, name) ; employee::employee_id = employee_id; if (salary <
50000.0) employee::salary = salary; else // Недопустимый оклад employee::salary = 0.0; }


void main(void)
{
employee worker("Happy Jamsa", 101, 10101.0); worker.show_employee(); }
```

Обратите внимание, что за объявлением объекта worker следуют круглые скобки и начальные значения, как и при вызове функции. Когда вы используете конструктор с параметрами, передавайте ему параметры при объявлении объекта:

```
employee worker("Happy Jamsa", 101, 10101.0);
```

Конструкторы и параметры по умолчанию

C++ позволяет указывать значения по умолчанию для параметров функции. Если пользователь не указывает каких-либо параметров, функция будет использовать значения по умолчанию. Конструктор не является исключением. Например, следующий конструктор employee использует по умолчанию значение оклада равным 10000.0, если программа не указывает оклад при создании объекта. Однако программа должна указать имя служащего и его номер:

```
employee::employee(char *name, long employee_id, float salary = 10000.00)
{
strcpy(employee::name, name); employee::employee_id = employee_id; if (salary <
50000.0) employee::salary = salary; else // Недопустимый оклад employee::salary = 0.0;
}
```

Перегрузка конструкторов

C++ позволяет вашим программам перегружать определения функций, указывая альтернативные функции для других типов параметров. C++ позволяет вам также перегружать конструкторы. Следующая программа перегружает конструктор employee. Первый конструктор требует, чтобы программа указывала имя служащего, номер служащего и оклад. Второй конструктор запрашивает пользователя ввести требуемый оклад, если программа не указывает его:

```
#include <iostream.h> #include <string.h>

class employee
{ public:
employee(char *, long, float); employee(char *, long); void
show_employee(void); int change_salary(float) ; long get_id(void);
private:
char name [64]; long employee_id; float salary;
};

employee::employee(char *name, long employee_id, float salary)
{
strcpy(employee::name, name); employee::employee_id = employee_id; if
(salary < 50000.0) employee::salary = salary; else // Недопустимый оклад
employee::salary = 0.0;
}

employee::employee(char *name, long employee_id)
{
strcpy(employee::name, name); employee::employee_id = employee_id; do
{
cout << "Введите оклад для " << name << " меньше $50000: "; cin >> employee::salary;
}
while (salary >= 50000.0);
}

void employee::show_employee(void)
{
cout << "Служащий: " << name << endl;
```

```
cout << "Номер служащего: " << employee_id << endl; cout << "Оклад: " << salary << endl; }

void main(void)
{
employee worker("Happy Jamsa", 101, 10101.0); employee manager("Jane Doe", 102);
worker.show_employee(); manager.show_employee();
}
```

Если вы откомпилируете и запустите эту программу, на вашем экране появится запрос ввести оклад для Jane Doe. Когда вы введете оклад, программа отобразит информацию об обоих служащих.

Представление о деструкторе

Деструктор автоматически запускается каждый раз, когда программа уничтожает объект.

При завершении программ C++ уничтожает объекты. Если определен деструктор, C++ будет автоматически вызывать деструктор для каждого объекта, когда программа завершается (т.е. когда объекты уничтожаются). Подобно конструктору, деструктор имеет такое же имя, как и класс объекта. Однако в случае деструктора его имя предваряется символом тильды (~), как показано ниже:

```
~class_name (void) //          >указывает деструктор
{
// Операторы деструктора
}
```

В отличие от конструктора деструктору параметры не передаются. Деструктор для класса employee:

```
void employee::-employee(void)
{
cout << "Уничтожение объекта для " << name << endl;
}
```

В данном случае деструктор просто выводит на ваш экран сообщение о том, что C++ уничтожает объект. Программа автоматически вызывает деструктор, без какого-либо явного вызова функции деструктора.

Что нужно знать?

Конструкторы и деструкторы представляют собой специальные функции класса, которые программа автоматически вызывает при создании или уничтожении объекта.

Большинство программ используют конструктор для инициализации элементов данных класса.

Основные концепции:

- Конструктор представляет собой специальную функцию, которую ваша программа автоматически вызывает каждый раз при создании объекта. Конструктор имеет такое же имя, как и класс объекта.
- Конструктор не имеет возвращаемого значения, но вы не указываете ему тип void. Вместо этого вы просто не указываете возвращаемое значение вообще.

- Когда ваша программа создает объект, она может передать параметры конструктору во время объявления объекта.
- C++ позволяет вам перегружать конструкторы и разрешает использовать значения по умолчанию для параметров.
- Деструктор представляет собой специальную функцию, которую ваша программа вызывает автоматически каждый раз при уничтожении объекта. Деструктор имеет такое же имя, как и класс объекта, но его имя предваряется символом тильды (~)

2. Практическое задание (100%)

Пользовательский класс **MyString** должен содержать необходимые элементы- данные, которые создаются в динамической области памяти.

- Конструкторы (без параметров, с параметрами, копирования) для создания строк: **MyString (…);** • Деструктор: **~MyString();**
- Метод ввода исходной строки: **set();** • Метод изменения исходной строки согласно варианту (исходная и измененная строка должны сохраняться в файле): **update() ;** • Метод вывода на экран: **print(…);**
- Каждый вызов методов (в том числе конструкторов и деструктора) сопровождается выдачей соответствующего сообщения;

Код методов – вне пространства определения класса.

Написать демонстрационную программу, в которой показать использование объектов созданного класса

Варианты:

1. Длина L нечетная, то удаляется символ, стоящий посередине строки;
2. Длина L четная, то удаляются 2 первых и 2 последних символа;
3. Длина L кратна 2-м, то удаляются все цифры, которые делятся на 2;
4. Длина L кратна 3-м, то удаляются все цифры, делящиеся на 3;
5. Длина L >10, то удаляются все цифры;
6. Длина L >15, то удаляются все a..z;
7. Длина L=10, то удаляются все A..Z;
8. Длина L кратна 4-м, то первая часть строки меняется местами со второй;
9. Длина L кратна 5-и, то подсчитывается количество скобок всех видов;
10. Длина L >5-и, то выделяется подстрока до первого пробела;
11. Длина L >6-и, то выделяется подстрока { } скобках;
12. Длина L >10-и, то удаляется подстрока в [] скобках;
13. Длина L >12-и, то удаляется подстрока до первой (скобки;
14. Длина L кратна 4-м, то выделяется подстрока после последнего пробела;
15. Длина L >5, то удаляются все точки.
16. Длина L четная, то выделяется подстрока до первого пробела
17. Длина L четная, то удаляется подстрока до первого пробела
18. Длина L четная, то выделяется подстрока со второго пробела
19. Длина L нечетная, то выделяется подстрока после первого пробела
20. Длина L нечетная, то удаляется подстрока со второго пробела
21. Длина L кратна 3, то удаляется каждый 3-й символ
22. Длина L четная, то удаляется каждый 2-й символ
23. Длина L нечетная, то
24. Длина L четная, то выделяется подстрока до последнего пробела
25. Длина L нечетная, то выделяется подстрока от последней цифры

3. Контрольные вопросы

1. Что такое конструктор
2. Как задается имя конструктора?
3. Может ли класс иметь более одного конструктора?
4. Что такое деструктор?
5. В чем состоит преимущество определения конструктора со списком инициализации элементов?
6. Какие виды конструкторов создаются по умолчанию?
7. В каком порядке инициализируются поля в классе?
8. Какая ошибка в следующей реализации конструктора?
9. Может ли деструктор иметь аргументы?
10. Какая ошибка в следующей реализации деструктора?

Практическое занятие №10. Перегрузка операций

Цели: Изучить механизм перегрузки операций.

1. Краткие теоретические сведения

Перегрузка операций – это одна из самых мощных и полезных возможностей ООП. Её применение позволяет существенно упростить написание программ, сделать их понятнее за счет применения интуитивно понятных обозначений, таких как +, *, [ ] и т.д.

Перегрузка операций, по сути, дает возможность переопределить стандартный язык программирования. Используя классы для создания новых типов переменных и перегрузку для определения операций над этими типами, можно, по сути, создать новый, собственный язык программирования, заточенный под конкретную задачу, и, таким образом, перейти от общего к предметно-ориентированному программированию.

Благодаря перегрузке операций любой класс можно сделать таким, что он будет вести себя подобно встроенному типу данных. В классе можно перегрузить любые из следующих операций:

Нельзя перегружать операции

. -> .* :: ?:

Функции-операции, реализующие перегрузку операций, имеют вид *operator операция ([операнды]) ;*

Если функция является элементом класса, то первый операнд соответствующей операции будет самим объектом, для которого вызвана функция-операция. В случае одноместной операции список параметров будет пуст. Для двухместных операций функция будет иметь один параметр, соответствующий второму операнду. Если функция- операция не является элементом класса, она будет иметь один параметр в случае одноместной операции и два — в случае двухместной.

Для перегрузки операций существуют такие правила:

- Приоритет и правила ассоциации для перегруженных операций остаются теми же самыми, что и для операций над встроенными типами.

- Нельзя изменить поведение операции по отношению к встроенному типу.
- Функция-операция должна быть либо элементом класса, либо иметь один или несколько параметров типа класса.
- Функция-операция не может иметь аргументов по умолчанию.

В качестве примера приведем перегрузку операции сложения и умножения (как скалярного произведения) для класса, описывающего вектор в пространстве с координатами x, y, z.

```
class my_vector {// Объявляем класс - вектор длины 3
public: // с элементами-данными x, y и z - координатами вектора long double x; long double y; long double z;
//Переопределяем оператор умножения long double operator*(my_vector v2);
//Переопределяем оператор сложения my_vector operator+(my_vector v2);
};
// переопределяем оператор умножения * как
// скалярное произведение векторов long double my_vector::operator*(my_vector v2)
{ long double help;
help=x*v2.x + y*v2.y + z*v2.z; return help;
};

//переопределяем оператор сложения + как сумму векторов my_vector my_vector::operator+(my_vector v2)
{ my_vector help; help.x = x + v2.x; help.y = y + v2.y; help.z = z + v2.z; return help;
};
```

После этого в программе для объектов класса my_vector будут доступны операции сложения и умножения:

```
my_vector v1, v2, v3; long double a; v1 = v2 + v3; a = v1 * v2;
```

При перегрузке операций необходимо помнить следующее:

- C++ не умеет образовывать из простых операций более сложные. Например, в классе со сложением строк мы определили присваивание и сложение; но это не значит, что тем самым будет автоматически определено присвоение суммы (+=). Такую операцию нужно реализовывать отдельно.
- Невозможно изменить синтаксис перегруженных операций. Одноместные операции должны быть одноместными, а двухместные — двухместными.
- Нельзя изобретать новые обозначения операций. Возможные операции ограничиваются тем списком, что приведен в начале этого раздела.
- Желательно сохранять смысл перегружаемой операции. Например, конкатенация — естественная семантика сложения для строк.

Операции не обязательно объявлять членами класса. Скажем, предыдущий пример с перегрузкой операций сложения и умножения для класса *my_vector* можно реализовать и так:

```
class my_vector { // Объявляем класс - вектор длины 3
public: // с элементами-данными x, y и z - координатами вектора long double x; long double y; long double z;
```

```
};
//переопределяем оператор умножения * как
//скалярное произведение векторов long double
operator*( my_vector v1, my_vector v2)
{ long double help;
help=v1.x*v2.x + v1.y*v2.y +v1. z*v2.z; return help;
};

//переопределяем оператор сложения + как сумму векторов my_vector operator+(
my_vector v1, my_vector v2)
{ my_vector help; help.x = v1.x + v2.x; help.y = v1.y + v2.y; help.z = v1.z + v2.z; return
help;
};
```

Несколько слов о перегрузке унарных операций.

Имеется особенность синтаксиса при перегрузке операций с префиксной и постфиксной формой ++ (инкремент) и -- (декремент).

В случае перегрузки префиксной формы используют следующий синтаксис переопределения:

```
void operator++( );
```

Если требуется переопределить постфиксную форму, то прототип перегружаемой операции будет таким:

```
void operator++( int );
```

Различие состоит лишь в том, что у постфиксной формы в скобках стоит *int*. Здесь int не играет роль аргумента и не означает целое число. Это просто сигнал для компилятора, чтобы использовалась постфиксная версия оператора.

Имеется также особенность перегрузки операции [ ] – индексация массива. Так как данная операция часто используется не только для доступа на чтение, но и для доступа на запись (то есть что-то типа *x[i] = y*), то целесообразно сделать так, чтобы перегруженная функция operator [] возвращала свое значение по ссылке. Это будет выглядеть примерно следующим образом:

```
int& operator[](int i);
```

2. Практическое задание (100%)

Все классы следует наделить конструкторами, деструктором. Необходимо явно реализовать конструктор копирования и перегрузить оператор присваивания. Необходимо подготовить демонстрацию по работе перегруженных для класса операторов.

Варианты:

1. Создать класс ПРЯМОУГОЛЬНИК со сторонами параллельными осям координат (прямоугольная система координат OXY). Реализовать метод вывода на экран информации о прямоугольнике. Перегрузить бинарный оператор несимметрической разности двух прямоугольников (-); унарный оператор (-): симметричное отображение прямоугольника относительно оси координат OX и OУ. Следует учесть, что результатом выполнения оператора может быть пустой прямоугольник.

2. Создать класс, описывающий тип ВРЕМЯ. Класс должен включать в себя атрибуты, описывающие часы, минуты, секунды и миллисекунды и иметь метод для вывода времени на экран. Для данного класса перегрузить следующие бинарные операторы: суммы(+), разности (-).

3. Создать класс МНОГОЧЛЕН степени ***n*** от одной переменной ***x***, задаваемый массивом своих коэффициентов (массив должен храниться в динамической памяти и задаваться

внутри конструктора, используя датчик случайных чисел). Класс должен включать конструктор, которому в качестве параметра передается степень многочлена; деструктор; конструктор копирования, метод, который печатает уравнение на экран. Для данного класса перегрузить следующие бинарные операторы: суммы двух многочленов (+), разности двух многочленов (-), операцию присваивания (=).

4. Для пространства $\mathbf{R}^3$ (выбрана правая система декартовых прямоугольных координат {0, **i, j, k**}) создать класс ВЕКТОР, предусмотрев для него несколько видов конструкторов, метод для вывода на экран его координат. Для данного класса перегрузить следующие бинарные операторы: суммы(+), разности (-),“векторное произведение” (*).

5. Создать класс ОТРЕЗОК ЧИСЛОВОЙ ПРЯМОЙ (сегмент). Предусмотреть несколько конструкторов для создания объектов класса, реализовать метод вывода отрезка на экран. Перегрузить следующие бинарные операторы: дополнение одного сегмента другим (+), пересечение двух сегментов (*), несимметрическая разность сегментов (-) {[1,5]-[3,6]=[1,3]}. Следует учесть, что результатом выполнения оператора может быть пустой сегмент, в этом случае следует вернуть отрезок нулевой длины [0,0].

6. Создать класс КВАДРАТНАЯ МАТРИЦА 3Х3. Элементы матрицы следует хранить в динамической памяти и задавать внутри конструктора, используя датчик случайных чисел. Класс должен включать конструктор; деструктор; конструктор копирования; метод для вывода матрицы на экран. Перегрузить следующие бинарные операторы: сумма матриц (+), произведение матриц (*).

7. Создать класс ПРЯМОУГОЛЬНИК со сторонами параллельными осям координат (прямоугольная система координат OXY). Реализовать метод вывода на экран информации о прямоугольнике. Перегрузить следующие бинарные операторы: пересечение двух прямоугольников (*), объединение двух прямоугольников (+). Следует учесть, что результатом выполнения оператора может быть пустой прямоугольник.

8. Создать класс ОКРУЖНОСТЬ в прямоугольной системе координат OXY. Реализовать метод вывода на экран информации об окружности. Перегрузить бинарный оператор несимметрической разности двух окружностей (-); унарный оператор (-): симметричное отображение окружности относительно оси координат OX и OУ. Следует учесть, что результатом выполнения оператора может быть окружность нулевого радиуса.

9. *Создать класс СТРОКА. Объект класса должен характеризоваться следующими свойствами: длина строки, динамически выделяемый массив символов, заканчивающийся символом ‘\n’, для хранения элементов строки. Предусмотреть несколько конструкторов для создания объектов класса, в том числе и конструктор с параметром, который задает длину будущей строки. Для данного класса реализовать метод вывода строки на экран, перегрузить следующие бинарные операторы: сцепление строк (+), удаление подстроки из строки (-) {“qwert”-“we”=qrt”, {“qwerty”-“tu”=“qwerty”, “qwert”-“qwert”=”” }. Учесть тот случай, когда результатом операции может быть пустая строка (строка нулевой длины).

10. Создать класс СЕКТОР ЕДИНИЧНОГО КРУГА в прямоугольной системе координат OXY с центром в точке O(0,0), который определяется двумя различными точкам на окружности единичного радиуса. Реализовать несколько конструкторов, метод вывода на экран информации о секторе. Перегрузить бинарные операторы: пересечение двух секторов (*), объединение двух секторов (+), несимметрическая разность двух секторов (-). Следует учесть все специальные случаи.

11. *Создать класс МНОГОЧЛЕН степени n от одной переменной x, задаваемый массивом своих коэффициентов (массив должен храниться в динамической памяти и задаваться внутри конструктора, используя датчик случайных чисел). Класс должен включать конструктор, которому в качестве параметра передается степень многочлена; деструктор; конструктор

копирования, метод, который печатает уравнение на экран. Для данного класса перегрузить бинарный оператор произведения двух многочленов (*), унарный оператор (-).

12. *Создать класс ОКРУЖНОСТЬ в прямоугольной системе координат OXY. Реализовать метод вывода на экран информации об окружности. Перегрузить следующие бинарные операторы: пересечение двух окружностей (*), объединение двух окружностей (+). Следует учесть, что результатом выполнения оператора может быть окружность нулевого радиуса.

13. Создать класс КОЛЬЦО (геометрическая фигура на плоскости) в прямоугольной системе координат OXY с центром в точке O(0,0), которая определяется двумя радиусами (R1, R2, R2>R1>0). Реализовать метод вывода на экран информации о кольце. Перегрузить бинарные операторы: пересечение двух колец (*), объединение двух колец (+), несимметрическая разность двух колец (-). Следует учесть, что результатом выполнения оператора может быть получено кольцо с равными радиусами (R1=R2), в этом случае следует сообщить об ошибке.

14. Создать класс ПРЯМОУГОЛЬНИК СО СТОРОНОЙ НА ОСИ OX, для прямоугольной системы координат OXY (одна сторона которого расположена на оси OX). Реализовать несколько конструкторов, метод вывода на экран информации о прямоугольнике. Перегрузить бинарные операторы: пересечение двух прямоугольников (*), объединение двух прямоугольников (+), несимметрическая разность двух прямоугольников (-). Следует учесть все специальные случаи.

15. Создать класс КВАДРАТНАЯ МАТРИЦА 3X3. Элементы матрицы следует хранить в динамической памяти и задавать внутри конструктора, используя датчик случайных чисел. Класс должен включать конструктор; деструктор; конструктор копирования; метод для вывода матрицы на экран. Перегрузить бинарный оператор разности двух матриц, операцию присваивания, унарный оператор (-), который возвращает транспонированную матрицу.

3. Список рекомендуемой литературы

1. Объектно-Ориентированное программирование: учебник / Г. С. Иванова, Т. Н. Ничушкина ; под общ. ред. Г. С. Ивановой. — М. : Изд-во МГТУ им. Н. Э. Баумана, 2014. — 455.

2. Professional C++, 3rd Edition. Marc Gregoire. ISBN: 978-1-118-85805-9. Paperback 984 pages. September 2014

4. Контрольные вопросы

1. Для чего в C++ применяется перегрузка операций
2. Истинно ли следующее утверждение: операция >= может быть перегружена?
3. Сколько аргументов требуется для определения перегруженной унарной операции?
4. Сколько аргументов требуется для определения перегруженной бинарной операции?
5. Чем отличается действие операции ++ в префиксной форме от её действия в постфиксной форме?
6. Истинно ли следующее утверждение: перегруженная операция всегда требует на один аргумент меньше, чем количество операндов?
7. Когда перегружается операция арифметического присваивания, то результат

a. Передается объекту справа от операции
b. Передается объекту слева от операции
c. Передается объекту, вызвавшему операцию
d. Должен быть возвращен

8. Истинно ли следующее утверждение: компилятор не выдаст сообщение об ошибке, если вы перегрузите операцию * для выполнения деления?

9. Существуют ли операции, которые нельзя перегружать?

Практическое занятие №11. Наследование

Цели:

Изучить возможности наследования классов на языке С++.

1. Краткие теоретические сведения

Каждый объект одного и того же класса имеет собственную копию данных класса. Но существуют задачи, когда данные должны быть компонентами класса, и иметь их нужно только в единственном числе. Такие компоненты должны быть определены в классе как *статические* (**static**). Статические данные классов не дублируются при создании объектов, т.е. каждый статический компонент существует в единственном экземпляре.

Наследование является наиболее значимой возможностью ООП. Наследованием называется процесс создания новых классов, называемых наследниками, дочерними или производными классами из уже существующих – базовых или родительских классов. Производный класс получает все возможности базового класса, но имеет также и свои собственные.

Выигрыш от применения наследования состоит в том, что наследование позволяет использовать существующий код несколько раз. Имея написанный и отлаженный базовый класс его можно больше не модифицировать, а механизм наследования позволит приспособить его для новых задач путем порождения от него новых производных классов. Использование уже написанного и отлаженного кода увеличивает надёжность программ.

Наследование также является упрощением распространения библиотек классов. Программист может использовать классы, созданные кем-то другим, без модификации кода, просто создавая производные классы и добавляя к ним новые возможности.

Рисунок 1 - Сущность наследования

БАЗОВЫЙ И ПРОИЗВОДНЫЙ КЛАССЫ

Класс в C++ может *наследовать* элементы-данные и элементы-функции от одного или нескольких б*азовых классов.* Сам класс называется в этом случае *производным* по отношению к базовым классам или *классом-потомком.* В свою очередь, производный класс может являться базовым по отношению к другим классам.

Принцип наследования, или *порождения* новых классов, позволяет абстрагировать (инкапсулировать) некоторые общие свойства и поведение в одном базовом классе, которые будут наследоваться всеми его потомками.

Наследование позволяет также модифицировать поведение базового класса. Производный класс может переопределять некоторые функции-элементы базового класса, оставляя основные свойства класса в неприкосновенности. **Синтаксис производного класса следующий:**

```
class имя класса: ключ доступа имя_базового класса [, ...]
{
тело_объявления_класса
} ;
```

Ключ_доступа — это одно из ключевых слов *private, protected* или *public*. Для производного класса доступны разделы *protected* и *public* базового класса; раздел *private* строго недоступен вне области действия базового класса, раздел *protected* недоступен для функций, не принадлежащих к базовому либо дочернему классу.

Таблица 1 - Наследование и доступ

Спецификатор доступа	Доступ из самого класса	Доступ из производных классов	Доступ из внешних классов и функций
public	Есть	Есть	Есть
protected	Есть	Есть	Нет
private	Есть	Нет	Нет

Для доступа к элементам базового класса через производный можно сформулировать такое правило: права доступа, определяемые для них базовым классом, остаются неизменными, если они такие же или строже, чем специфицировано *ключом доступа*. В противном случае права доступа определяются ключом в определении производного класса. То есть ключ доступа, указанный при наследовании «сдвигает» права доступа в сторону «более строгих».

Например, при наследовании с ключом *public* права доступа к элементам базового класса остаются неизменными; при закрытом наследовании (ключ *private*) все элементы базового класса будут недоступны за пределами производного класса. При наследовании с ключом *protected,* элементы, которые были *public* становятся *protected*, остальные остаются без изменения.

При закрытом наследовании можно сделать некоторые открытые функции базового класса открытыми в производном, если переобъявить их имена в производном классе. Пример:

```
class First { public: void
FFunc(void) ;
//...
```

```
};

class Second: private First { public:
First::FFunc; // First::FFunc() открыта в классе Second.
//.. .
};
```

Как правило, в практических задачах применяется почти исключительно открытое наследование.

Язык C++ допускает простое и сложное наследование. При простом наследовании у класса может быть только один родитель, при сложном – два и более.

ПРОСТОЕ НАСЛЕДОВАНИЕ

Пример простого наследования:

```
class Time // Базовый класс - время.
{
int hr, min; public:
Time(int h=12, int m=0): hr(h), min(m) {} // Конструктор
void SetTime(int h, int m) {hr = h; min = m; } //Метод установки времени
void Show( ) { printf("%02d:%02d", hr, min) }; // Метод, чтобы показать время
// на консоли
};

class Alarm: public Time// Класс сообщений таймера – дочерний от Time.
{
char *msg; // Указатель на строку-сообщение public:
Alarm(char*); // Конструктор нового класса ~Alarm()
{ delete [] msg; } // Деструктор void SetMsg(char*); //
Метод установки сообщений void Show(); //
Переопределяет метод Show ( ).
};
Alarm::Alarm(char *str) // Конструктор класса-потомка
{
msg = new char[strlen (str) + 1]; strcpy(msg, str);
}

void Alarm::SetMsg(char *str) // Описание функции класса-
потомка SetMsg( )
{
delete [] msg;
msg = new char[strlen (str) + 1]; strcpy(msg, str);
}

void Alarm::Show( ) // Переопределение функции Show( )
{
Time::Show(); // Вызов базовой Show() printf(": %s\n", msg);
}
```

Теперь можно создавать переменные производного класса, например, таким образом:

```
int main ( ) {
Alarm a = "Test Alarm!!!"; // Время по умолчанию 12:00. a.Show ( );
a.SetTime(7, 40); // Функция базового класса a.Show ( ) ;
a.SetMsg("It's time! " ); // Функция производного класса. a.Show( ); return
0;
}
```

Несколько слов о наследовании конструкторов. При создании объекта производного класса компилятор перед вызовом конструктора производного класса автоматически вызывает сначала конструктор родительского класса, но только тот, который может быть вызван без параметров (то есть либо конструктор по умолчанию, либо конструктор без параметров, либо конструктор с параметрами по умолчанию – как в примере выше у класса *Time*). Но конструктор родительского класса с параметрами не наследуется производным в том смысле, что он не будет вызываться автоматически при создании объекта производного класса с такими параметрами.

Вместе с тем, нужный конструктор базового класса можно вызвать явно через список инициализации. Например, дочерний класс из предыдущего примера можно было задать и так:

```
class Alarm: public Time // Класс сообщений таймера.
{
char *msg; public:
Alarm(char*);
Alarm(char*, int, int);  // Новый конструктор.
~Alarm( ) { delete[] msg; } void SetMsg(char*) ;
void Show(); // Переопределяет Time:: Show ( ).
};

Alarm::Alarm(char *str, int h, int m): Time(h, m) - // явный вызов конструктора
                                                    //родительского класса
{
msg = new char[strlen(str) + 1]; strcpy(msg, str);
}
```

Деструкторы базовых классов всегда вызываются компилятором автоматически.

СЛОЖНОЕ НАСЛЕДОВАНИЕ

Язык C++ допускает не только простое, но и *сложное наследование,* т. е. наследование от двух и более непосредственных базовых классов. Это позволяет создавать классы, комбинирующие в себе свойства нескольких независимых классов- предков. Синтаксис производного класса Alarm при сложном наследовании от классов Time и Message:

```
class Alarm: public Time, public Message
{
…….
```

```
};
```

В определенных ситуациях могут появиться проблемы, связанные со сложным наследованием. Например, в обоих базовых классах могут существовать методы с одинаковыми именами, а в производном классе такой метод не переопределяется. Как в этом случае объект производного класса определит, какой из методов базовых классов выбрать? Одного имени метода недостаточно, поскольку компилятор не сможет вычислить, какой из двух методов имеется в виду. Проблема решается путем использования оператора разрешения области действия, определяющего класс, в котором находится метод. Пример:

```
class A
{
  public:
    void show ( ) { cout << "Класс A\n"; }
};
class B
{
  public:
    void show ( ) { cout << "Класс B\n"; }
};
class C : public A, public B
{
};
//////////////////////////////////////////////////////////
int main ( )
{
  C objC;           // объект класса C
  // objC.show ( ); // так делать нельзя - программа не скомпилируется
  objC.A::show ( ); // так можно
  objC.A::show ( ); // так можно
  return 0;
}
```

Другой вид неопределенности появляется, если создается производный класс от двух базовых классов, которые, в свою очередь, являются производными одного класса. Это создает дерево наследования в форме ромба. Компилятор не сможет решить, какой из методов использовать, и сообщит об ошибке.

Существует множество вариаций этой проблемы. Поэтому технология программирования предписывает избегать множественного наследования.

2. Практическое задание

Реализовать иерархию классов для простого (задание 1) и сложного (задание 2) наследования:

2.1. Индивидуальное задание. Простое наследование (50%)

1. Создать класс квадрат, члены класса – длина стороны. Предусмотреть в классе методы вычисления и вывода сведений о фигуре – диагональ, периметр, площадь. Создать производный класс – правильная квадратная призма с высотой H, добавить в класс метод определения объема фигуры, перегрузить методы расчета площади и вывода сведений о фигуре. Написать программу, демонстрирующую работу с этими классами: дано N квадратов и M призм, найти квадрат с максимальной площадью и призму с максимальной диагональю.

2. Создать класс треугольник, члены класса – длины 3-х сторон. Предусмотреть в классе методы проверки существования треугольника, вычисления и вывода сведений о фигуре – длины сторон, углы, периметр, площадь. Создать производный класс – равносторонний треугольник, перегрузить в классе проверку, является ли треугольник равносторонним и метод вывода сведений о фигуре. Написать программу,

демонстрирующую работу с классом: дано K треугольников и L равносторонних треугольников, найти среднюю площадь для K треугольников и наибольший равносторонний треугольник.

3. Создать класс окружность, член класса – радиус R. Предусмотреть в классе методы вычисления и вывода сведений о фигуре – площади, длины окружности. Создать производный класс – круглый прямой цилиндр с высотой h, добавить в класс метод определения объема фигуры, перегрузить методы расчета площади и вывода сведений о фигуре. Написать программу, демонстрирующую работу с классом: дано N окружностей и M цилиндров, найти окружность максимальной площади и средний объем цилиндров.

4. Создать класс квадрат, члены класса – длина стороны. Предусмотреть в классе методы вычисления и вывода сведений о фигуре – диагоналей, периметр, площадь. Создать производный класс – правильная пирамида с апофемой h, добавить в класс метод определения объема фигуры, перегрузить методы расчета площади и вывода сведений о фигуре. Написать программу, демонстрирующую работу с классом: дано N квадратов и M пирамид, найти квадрат с минимальной площадью и количество пирамид с высотой более числа a (a вводить).

5. Создать класс четырехугольник, члены класса – координаты 4-х точек. Предусмотреть в классе методы проверки существования четырехугольника вычисления и вывода сведений о фигуре – длины сторон, диагоналей, периметр, площадь. Создать производный класс – параллелограмм, предусмотреть в классе проверку, является ли фигура параллелограммом. Написать программу, демонстрирующую работу с классом: дано N четырехугольников и M параллелограммов, найти среднюю площадь N четырехугольников и параллелограммы наименьшей и наибольшей площади.

6. Создать класс треугольник, члены класса – координаты 3-х точек. Предусмотреть в классе методы проверки существования треугольника, вычисления и вывода сведений о фигуре – длины сторон, углы, периметр, площадь. Создать производный класс – равносторонний треугольник, предусмотреть в классе проверку, является ли треугольник равносторонним. Написать программу, демонстрирующую работу с классом: дано N треугольников и M равносторонних треугольников, вывести номера одинаковых треугольников и равносторонний треугольник с наименьшей медианой.

7. Создать класс прямоугольник, члены класса – длины сторон a и b. Предусмотреть в классе методы вычисления и вывода сведений о фигуре – длины сторон, диагоналей, периметр, площадь. Создать производный класс – параллелепипед с высотой c, добавить в класс метод определения объема фигуры, перегрузить методы расчета площади и вывода сведений о фигуре. Написать программу, демонстрирующую работу с классом: дано N прямоугольников и M параллелепипедов, найти количество прямоугольников, у которых площадь больше средней площади прямоугольников и количество кубов (все ребра равны).

8. Создать класс окружность, член класса – радиус R. Предусмотреть в классе методы вычисления и вывода сведений о фигуре – площади, длины окружности. Создать производный класс – конус с высотой h, добавить в класс метод определения объема фигуры, перегрузить методы расчета площади и вывода сведений о фигуре. Написать программу, демонстрирующую работу с классом: дано N окружностей и M конусов, найти количество окружностей, у которых площадь меньше средней площади всех окружностей, и наибольший по объему конус.

9. Создать класс четырехугольник, члены класса – координаты 4-х точек. Предусмотреть в классе методы вычисления и вывода сведений о фигуре – длины сторон, диагоналей, периметр, площадь. Создать производный класс – равнобочная трапеция, предусмотреть в классе проверку, является ли фигура равнобочной трапецией. Написать

программу, демонстрирующую работу с классом: дано N четырехугольников и M трапеций, найти максимальную площадь четырехугольников и количество четырехугольников, имеющих максимальную площадь, и трапецию с наименьшей диагональю.

10. Создать класс равносторонний треугольник, член класса – длина стороны. Предусмотреть в классе методы вычисления и вывода сведений о фигуре – периметр, площадь. Создать производный класс – правильная треугольная призма с высотой H, добавить в класс метод определения объема фигуры, перегрузить методы расчета площади и вывода сведений о фигуре. Написать программу, демонстрирующую работу с классом: дано N треугольников и M призм. Найти количество треугольников, у которых площадь меньше средней площади треугольников, и призму с наибольшим объемом.

11.Создать класс треугольник, члены класса – длины 3-х сторон. Предусмотреть в классе методы проверки существования треугольника, вычисления и вывода сведений о фигуре – длины сторон, углы, периметр, площадь. Создать производный класс – прямоугольный треугольник, предусмотреть в классе проверку, является ли треугольник прямоугольным. Написать программу, демонстрирующую работу с классом: дано N треугольников и M прямоугольных треугольников, найти треугольник с максимальной площадью и прямоугольный треугольник с наименьшей гипотенузой.

12.Создать класс четырехугольник, члены класса – координаты 4-х точек. Предусмотреть в классе методы вычисления и вывода сведений о фигуре – длины сторон, диагоналей, периметр, площадь. Создать производный класс – квадрат, предусмотреть в классе проверку, является ли фигура квадратом. Написать программу, демонстрирующую работу с классом: дано N четырехугольников и M квадратов, найти четырехугольники с минимальной и максимальной площадью и номера одинаковых квадратов.

13.Создать класс треугольник, члены класса – длины 3-х сторон. Предусмотреть в классе методы проверки существования треугольника, вычисления и вывода сведений о фигуре – длины сторон, углы, периметр, площадь. Создать производный класс – равнобедренный треугольник, предусмотреть в классе проверку, является ли треугольник равнобедренным. Написать программу, демонстрирующую работу с классом: дано N треугольников и M равнобедренных треугольников, найти среднюю площадь для N треугольников и равнобедренный треугольник с наименьшей площадью.

14. Создать класс квадрат, член класса – длина стороны. Предусмотреть в классе методы вычисления и вывода сведений о фигуре – периметр, площадь, диагональ. Создать производный класс – куб, добавить в класс метод определения объема фигуры, перегрузить методы расчета площади и вывода сведений о фигуре. Написать программу, демонстрирующую работу с классом: дано N1 квадратов и N2 кубов. Найти среднюю площадь квадратов и количество кубов с наибольшей площадью.

15. 15.Создать класс четырехугольник, члены класса – координаты 4-х точек. Предусмотреть в классе методы вычисления и вывода сведений о фигуре – длины сторон, диагоналей, периметр, площадь. Создать производный класс – ромб, предусмотреть в классе проверку, является ли фигура ромбом. Написать программу, демонстрирующую работу с этими классами: дано N четырехугольников и M ромбов, найти четырехугольник с минимальным периметром и среднюю площадь ромбов.

2.2 Индивидуальное задание. Множественное наследование (50%)

1. Используя родительский класс «ТРАНСПОРТ» породить производный класс

«АВТОМОБИЛЬ». Используя классы «ВОДИТЕЛЬ» и «АВТОМОБИЛЬ», описать класс

«ВОДИТЕЛЬ АВТОМОБИЛЯ». Расширить класс «ВОДИТЕЛЬ АВТОМОБИЛЯ» создав два производных класса «ВОДИТЕЛЬ СЛУЖЕБНОГО АВТОМОБИЛЯ» и «ВОДИТЕЛЬ ТАКСИ». Продумать для данной иерархии классов поля и методы (обязательно: вывод информации о водителе, автомобиле)

2. Используя родительский класс «СЛУЖАЩИЙ» породить производный класс

«ДИРЕКТОР». Используя классы «ФИРМА» и «ДИРЕКТОР», описать класс «РУКОВОДИТЕЛЬ ФИРМЫ». Расширить класс «РУКОВОДИТЕЛЬ ФИРМЫ» создав два производных класса «РУКОВОДИТЕЛЬ ГОС.УЧ.» и «РУКОВОДИТЕЛЬ ООО».

Продумать для данной иерархии классов поля и методы (обязательно: вывод информации о фирме и руководителе)

3. Используя родительский класс «НЕДВИЖИМОСТЬ» породить производный класс «ЗДАНИЕ». Используя классы «ЗДАНИЕ» и «ВЛАДЕЛЕЦ», описать класс

«ВЛАДЕЛЕЦ ЗДАНИЯ». Расширить класс «ВЛАДЕЛЕЦ ЗДАНИЯ» создав два производных класса «ВЛАДЕЛЕЦ ЧАСТНОГО ДОМА» и «ВЛАДЕЛЕЦ ОТЕЛЯ ». Продумать для данной иерархии классов поля и методы (обязательно: вывод информации о владельце и здании)

4. Используя родительский класс «ТРАНСПОРТ» породить производный класс

«САМОЛЕТ». Используя классы «ПИЛОТ» и «САМОЛЕТ», описать класс «ПИЛОТ САМОЛЕТА». Расширить класс «ПИЛОТ САМОЛЕТА» создав два производных класса «ПИЛОТ ГРАЖДАНСКОГО САМОЛЕТА» и «ПИЛОТ ВОЕННОГО САМОЛЕТА».

Продумать для данной иерархии классов поля и методы (обязательно: вывод информации о пилоте, самолете)

5. Используя родительский класс «РАБОТНИК» породить производный класс

«РЕЖИССЕР». Используя классы «ФИЛЬМ» и «РЕЖИССЕР», описать класс «РЕЖИССЕР ФИЛЬМА». Расширить класс «РЕЖИССЕР ФИЛЬМА» создав два производных класса «РЕЖИССЕР ХУДОЖЕСТВЕННОГО ФИЛЬМА» и «РЕЖИССЕР ДОКУМЕНТАЛЬНОГО ФИЛЬМА». Продумать для данной иерархии классов поля и методы (обязательно: вывод информации о фирме и руководителе)

3. Список рекомендуемой литературы

1. Павловская Т. А.C/C++. Программирование на языке высокого уровня : для магистров и бакалавров : учеб. для вузов / Т. А. Павловская. - Гриф МО. - Санкт-Петербург: Питер, 2013. - 460 с. : ил.

2. Professional C++, 3rd Edition. Marc Gregoire. ISBN: 978-1-118-85805-9. Paperback 984 pages. September 2014

4. Контрольные вопросы

1. В чем состоит назначение наследования?
2. Когда программисту-разработчику целесообразно прибегнуть к наследованию?
3. Напишите первую строку описания класса Child, который является public-производным от класса Parent.
4. Верно ли утверждение: создание производного класса требует коренных изменений в базовом классе?
5. Члены базового класса, для доступа к ним методов производного класса должны быть объявлены как public или _.

6. Пусть базовый класс содержит метод basefunc( ), а производный класс не имеет такого метода. Может ли объект производного класса иметь доступ к методу basefunc( ).

7. Истинно ли следующее утверждение: если конструктор производного класса не определен, то объекты этого класса будут использовать конструкторы базового класса?

8. Как используется оператор разрешения области действия для разрешения неопределенностей?

9. Истинно ли следующее утверждение: иногда полезно создать класс, объектов которого никогда не будет создано?

10. Пусть класс Derv является дочерним от класса Base. Пусть определена переменная типа Derv, расположенная в функции main( ). Через эту переменную можно получить доступ к

а) членам класса Derv, объявленным как public;

б) членам класса Derv, объявленным как protected; в) членам класса Derv, объявленным как private; г) членам класса Base, объявленным как public;

д) членам класса Base, объявленным как protected; е) членам класса Base, объявленным как private;

11. Пусть существует класс Derv, производный от класса Base. Напишите объявление конструктора производного класса, принимающего один аргумент и передающего его в конструктор базового класса.

12. Истинно ли следующее утверждение: невозможно сделать объект одного класса членом другого класса?

Практическое занятие №12. Программирование динамических и виртуальных методов

Цели:

Изучить возможности виртуальных функций при наследовании классов на языке C++, абстрактные классы.

1. Краткие теоретические сведения ВИРТУАЛЬНЫЕ ФУНКЦИИ

Виртуальные функции являются важной частью реализации механизма полиморфизма (то есть выполнения разных действий с объектами разного типа). Использование виртуальных функций позволяет выполнить нужные действия в том случае, если доступ к объекту осуществляется не по значению, а по указателю.

Пусть имеется класс «Фигура», и имеются его классы-наследники «Круг», «Квадрат» и «Треугольник». Пусть в каждом из этих классов есть функция draw( ), которая прорисовывает фигуры на экране.

Теперь пусть имеется указатель X на объект класса «Фигура». По правилам языка этот указатель может хранить адрес как самого класса «Фигура», так и адрес любого из его потомков, т.е. и «Круга» и «Квадрата» и «Треугольника». Теперь необходимо реализовать функцию draw( ) так, чтобы при обращении к ней через указатель X->draw( ) вызывалась бы именно нужная функция draw( ), то есть именно того класса, адрес которого и хранится в указателе X. Понятно, что у круга, квадрата и треугольника это будут разные функции. Чтобы обеспечить такую возможность для функции draw( ) её необходимо объявить виртуальной в базовом классе.

Ниже приводятся примеры, демонстрирующие возможности виртуальной функции. Первый пример показывает, что бывает, когда базовый и производный классы содержат

функции с одним и тем же именем, и к ним обращаются с помощью указателей, но без использования виртуальных функций.

```
class Base                    //Базовый класс
{ public:
void show()                    //Обычная функция
{ cout << "Base\n"; }
};

class Derv1 : public Base //Производный класс 1
{
public:
void show()
{ cout << "Derv1\n"; }
};

class Derv2 : public Base //Производный класс 2
{
public:
void show()
{ cout << "Derv2\n"; }
};

int main()


        {
        Derv1 dv1; //Объект производного класса 1 Derv2 dv2; //Объект
    производного класса 2 Base* ptr; //Указатель на базовый класс

        ptr = &dv1; //Адрес dv1 занести в указатель ptr->show();
                //Выполнить show()
        ptr = &dv2; //Адрес dv2 занести в указатель ptr->show();
                //Выполнить show()
        return 0;
        }
```

Поскольку указатели на объекты дочерних классов совместимы по типу с указателями на объекты базового класса присвоение *ptr = &dv1;* и *ptr = &dv2;* вполне корректно. В результате выполнения приведенной программы будет дважды вызвана функция *show( )* именно базового класса, так как компилятор в данном случае не смотрит на содержимое указателя *ptr* , а выбирает тот метод, который удовлетворяет типу указателя (см. рисунок).

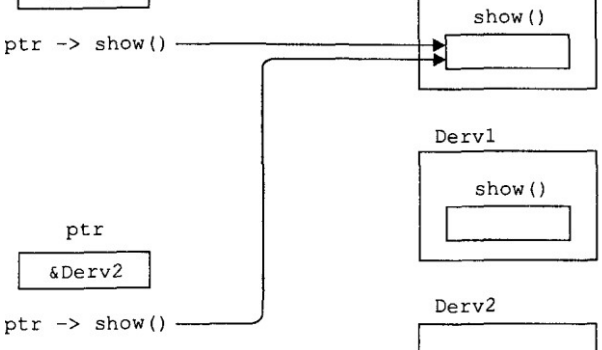


Рисунок 1 - Доступ через указатель без использования виртуальных функций

Теперь изменим обсуждаемую программу таким образом, чтобы функция *show( )* базового класса была бы объявлена как виртуальная функция (то есть добавим перед её объявлением в базовом классе ключевое слово virtual).

```
class Base     //Базовый класс
    {
    public:
      virtual void show() //Виртуальная функция
      { cout << "Base\n"; }
    };
    class Derv1 : public Base //Производный класс 1
    {
    public:
      void show()
      { cout << "Derv1\n"; }
    };

    class Derv2 : public Base //Производный класс 2
    {
    public:
      void show()
      { cout << "Derv2\n"; }
};

int main()
    {
    Derv1 dv1; //Объект производного класса 1 Derv2 dv2; //Объект производного класса 2 Base* ptr; //Указатель на базовый класс

    ptr = &dv1; //Адрес dv1 занести в указатель ptr->show();
          //Выполнить show()
    ptr = &dv2; //Адрес dv2 занести в указатель ptr->show();
          //Выполнить show()
    return 0;
    }
```

На выходе, в отличие от первого примера, будем иметь Derv1 Derv2 то есть при вызове функции *show( )* будут выполнены методы именно соответствующих производных классов, а не базового. То есть один и тот же вызов *ptr->show();* запускает на выполнение разные функции в зависимости от содержимого указателя *ptr*. Компилятор выбирает функцию, удовлетворяющую тому, что занесено в указатель, а не типу указателя, как в первом примере (см. рисунок).

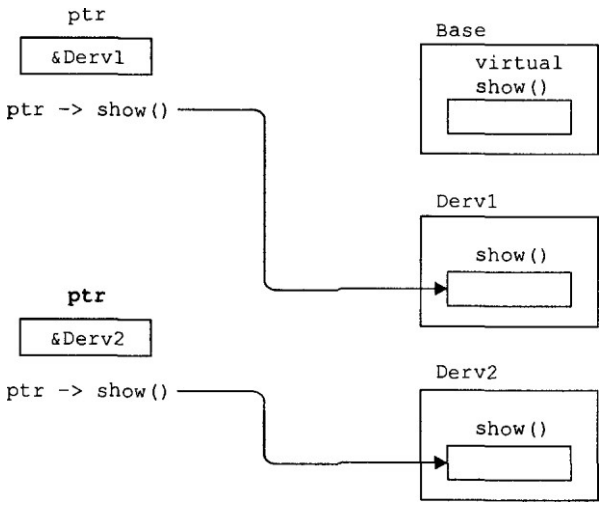


Рисунок 2 - Доступ через указатель к виртуальным функциям

АБСТРАКТНЫЕ КЛАССЫ И ЧИСТЫЕ ВИРТУАЛЬНЫЕ ФУНКЦИИ

Базовый класс, объекты которого никогда не будут созданы, называется абстрактным классом. Такой класс существует с единственной целью – быть родительским по отношению к производным классам, объекты которых уже будут реализованы. Если класс должен быть абстрактным, то от создания объектов этого класса его можно защитить программно. Для этого достаточно ввести в класс хотя бы одну чистую виртуальную функцию. Чистая виртуальная функция – это функция, после объявления которой добавлено выражение = 0. В следующем примере демонстрируется объявление и работа с чистыми виртуальными функциями.

```
class Base        //базовый класс
   {
     public:
     virtual void show() = 0; //чистая виртуальная функция
   };

class Derv1 : public Base         //порожденный класс 1
   {
   public:
     void show()
       { cout << “Derv1\n”; }
   };
class Derv2 : public Base         //порожденный класс 2
   {
   public: void
     show()
       { cout << “Derv2\n”; }
   };

int main()
   {
  // Base bad;  //невозможно создать объект //из абстрактного
    класса Base* arr[2];          //массив указателей на
                                  //базовый класс
```

```
    Derv1 dv1;   //Объект производного класса 1 Derv2 dv2;  //Объект
производного класса 2

    arr[0] = &dv1; //Занести адрес dv1 в массив arr[1] = &dv2; //Занести
адрес dv2 в массив

    arr[0]->show(); //Выполнить функцию show() arr[1]->show(); //над
обоими объектами return 0;
    }
```

Знак равенства при объявлении чистой виртуальной функции *virtual void show() = 0;* это не операция присваивания, это просто способ сообщить компилятору, что функция будет чистой виртуальной. Если теперь попытаться создать объект этого класса, компилятор выдаст ошибку.

На практике механизм виртуальных функций часто применяется для корректного вызова деструкторов. Технология программирования прямо предписывает, что деструкторы базовых классов должны быть виртуальными. Если деструктор базового класса не является виртуальным, то, будучи обычным методом, он будет вызываться независимо от того, данные какого типа в нем хранятся. Это приведет к тому, что может быть удалена только та часть объекта, которая относится к базовому классу. Конечно, если ни один из деструкторов (ни у базового, ни у производного класса) ничего особенного не делает, тогда их виртуальность перестает быть такой уж необходимой.

2. Практическое задание (100%)

1. Определить иерархию классов (в соответствии с вариантом).
2. Определить в классе статическую компоненту - указатель на начало связанного списка объектов и статическую функцию для просмотра списка.
3. Реализовать классы.
4. Написать демонстрационную программу, в которой создаются объекты различных классов и помещаются в список, после чего список просматривается.
5. Сделать соответствующие методы не виртуальными и посмотреть, что будет.
6. Реализовать вариант, когда объект добавляется в список при создании, т.е. в конструкторе.

Методические указания

1. Для определения иерархии классов связать отношением наследования классы заданного варианта. Из перечисленных классов выбрать один, который будет стоять во главе иерархии. Это абстрактный класс.
2. Определить в классах все необходимые конструкторы и деструктор.
3. Поля класса специфицировать как **protected**. 4. Пример определения статических компонентов:

```
static person* begin; // указатель на начало списка static void
print(); // просмотр списка
```

5. Статическое поле инициализировать вне определения класса, в глобальной области.
6. Для добавления объекта в список предусмотреть метод класса, т.е. объект сам добавляет себя в список. Например, a.Add() − объект **a** добавляет себя в список.

Включение объекта в список можно выполнять при создании объекта, т.е. поместить операторы включения в конструктор. В случае иерархии классов, включение объекта в список

должен выполнять **только** конструктор базового класса. Вы должны продемонстрировать оба этих способа.

7. Список просматривать путем вызова виртуального метода **Show** каждого объекта.

8. Статический метод просмотра списка вызывать не через объект, а через класс.

Варианты заданий

1) студент, преподаватель, персона, завкафедрой;
2) служащий, персона, рабочий, инженер;
3) рабочий, кадры, инженер, администрация;
4) деталь, механизм, изделие, узел;
5) организация, страховая компания, судостроительная компания, завод;
6) журнал, книга, печатное издание, учебник;
7) тест, экзамен, выпускной экзамен, испытание;
8) место, область, город, мегаполис;
9) игрушка, продукт, товар, молочный продукт;
10) квитанция, накладная, документ, чек;
11) автомобиль, поезд, транспортное средство, экспресс;
12) двигатель, двигатель внутреннего сгорания, дизель, турбореактивный двигатель;
13) республика, монархия, королевство, государство; 14) млекопитающие, парнокопытные, птицы, животное;
15) корабль, пароход, парусник, корвет.

3. Список рекомендуемой литературы

1. Павловская Т. А.C/C++. Программирование на языке высокого уровня : для магистров и бакалавров : учеб. для вузов / Т. А. Павловская. - Гриф МО. - Санкт-Петербург: Питер, 2013. - 460 с. : ил.

2. Professional C++, 3rd Edition. Marc Gregoire. ISBN: 978-1-118-85805-9. Paperback 984 pages. September 2014

4. Контрольные вопросы

1. Какие возможности перед программистом открывают виртуальные функции?
2. Истинно ли утверждение о том, что указатель на базовый класс может ссылаться на объекты порожденного класса?
3. Пусть указатель p ссылается на объекты базового класса и содержит адрес объекта порожденного класса. Пусть в обоих этих классах имеется невиртуальный метод ding(). Тогда выражение p->ding() вызовет метод ding() из …… класса.
4. Напишите описание для виртуальной функции dang(), возвращающей результат void и имеющей аргумент типа int.
5. Пусть указатель p ссылается на объекты базового класса и содержит адрес объекта порожденного класса. Пусть в обоих этих классах имеется виртуальный метод ding(). Тогда выражение p->ding() вызовет метод ding() из …… класса.
6. Напишите описание для чистой виртуальной функции aragorn(), не возвращающей значений и не имеющей аргументов.
7. Чистая виртуальная функция, это виртуальная функция, которая: а) делает свой класс абстрактным;

б) не возвращает результата;

в) используется в базовом классе; г) не имеет аргументов.

8. Напишите определение массива parr, содержащего 10 указателей на объекты класса dong.

9. Абстрактный класс используется тогда, когда

а) не планируется создавать порожденные классы;

б) есть несколько связей между двумя порожденными классами в) необходимо запретить создавать объекты класса

Практическое занятие №13. Оператор и конструктор преобразования

Цели:

Получить практические навыки применения операторов и конструкторов преобразования в программах C++.

1. Краткие теоретические сведения

Объекты класса могут быть преобразованы в другие типы данных с помощью операций приведения типа или конструкторов преобразования.

Конструктором преобразования называется функция-конструктор, имеющая в качестве параметра объект какого-либо класса. Если конструктор класса A имеет единственный параметр типа B, то объект класса B может быть неявно преобразован в объект класса A с помощью такого конструктора, то есть компилятор может сам вызвать такой конструктор и из B сделать A. Это происходит в следующем примере:

```
class Hold { char
*str; public:
Hold(const
char*);
//...
}; main
( )
{
Hold mainObj = "This is a local object in main.";
//. . . return 0;
}
```

Здесь объявленный в классе конструктор `Hold(const char*);` является, по сути, конструктором преобразования. Если вызовы конструктора с одним параметром в качестве конструктора преобразования необходимо запретить, то его нужно объявить с ключевым словом `explicit`. Пример:

```
class Hold { char
*str; public:
explicit Hold(const char*);
//. . .
}; main
( )
{
//Теперь неявное преобразование недопустимо:
// Hold mainObj = "This is a local object in main.";
```

```
    //Но можно вызвать конструктор с параметром явным образом: Hold
mainObj("This is a local object in main."); return 0;
    }
```

В классе можно определять элементы-функции, которые будут обеспечивать явное или неявное преобразование объектов данного класса в объекты других классов. Эти функции называют операциями приведения типа или процедурами преобразования. Они имеют следующий синтаксис:

```
operator имя_нового_типа( );
```

Процедуры преобразования характеризуются следующими правилами:

- У процедуры преобразования нет параметров;
- Для процедуры преобразования не специфицируется явно тип возвращаемого значения. Подразумевается тип, имя которого следует за ключевым словом operator.

В следующем примере процедура преобразования преобразует время, выраженное в часах, минутах и секундах в соответствующее число секунд.

```
class Time { int hr,
min, sec; public:
// Конструктор:
Time(int h, int m, int s): hr(h), min(m), sec(s) {}
// Процедура преобразования: operator int( );
};

Time::operator int( ) {
// Преобразует время в число секунд от начала суток: return
(3600*hr + 60*min + sec);
}
main
() {
int h = 7; int m = 40; int s = 10; int d;

Time t (h, m, s);
// Здесь будет неявно вызвана функция operator int( ): d = t;
}
```

Таким образом, преобразование от встроенного типа к определенному пользователем может быть выполнено только при создании объекта с помощью конструктора преобразования. А преобразование от типа, определенного пользователем, к встроенному типу выполняется с помощью определенных в классе операций преобразования типа. Преобразование типов может быть также выполнено с помощью переопределения операции присваивания.

2. Практическое задание (100%)

Разработайте и опишите классы из предложенного списка. Для каждого класса выполнить преобразование классов (ко всем оставшимся) двумя способами, т.е. используя оператор преобразования и конструктор. Перегрузите оператор присваивания.

Варианты заданий

Вариант 1.

- Угол в радианах
- Угол в градусах

- Угол в минутах ● Угол в секундах **Вариант 2.**
- Длина в м
- Длина в км
- Длина в ангстремах
- Длина в икс-единицах
- Длина в мкм ● Длина в нм **Вариант 3.**
- Площадь в м2 ● Площадь в км2 ● Площадь в дм2
- Площадь в см2
- Площадь в мм2 ● Площадь в барн **Вариант 4.**
- Объем в м3 ● Объем в дм3 ● Объем в см3 ● Объем в мм3 **Вариант 5.**
- Время в с
- Время в кс
- Время в мс
- Время в мкс ● Время в нс **Вариант 6.**
- Частота периодического процесса в Гц ● Частота периодического процесса в ТГц ● Частота периодического процесса в ГГц
- Частота периодического процесса в МГц ● Частота периодического процесса в кГц **Вариант 7.**
- Масса в кг
- Масса в Мг
- Масса в г
- Масса в мг ● Масса в мкг **Вариант 8.**
- Плотность массы в кг/м3
- Плотность массы в Мг/м3
- Плотность массы в кг/дм3 ● Плотность массы в г/см3 **Вариант 9.**
- Сила в Н (ньютон)
- Сила в МН
- Сила в кН
- Сила в мН
- Сила в мкН

Вариант 10.

- Единицы количества информации в Битах
- Единицы количества информации в Байтах
- Единицы количества информации в Кбайт
- Единицы количества информации в Мбайт ● Единицы количества информации в Гбайт.

Вариант 11.

- Давление в Па (паскаль)
- Давление в ГПа
- Давление в МПа
- Давление в кПа
- Давление в мПа ● Давление в мкПа **Вариант 12.**
- Энергия, работа в Дж (джоуль)
- Энергия, работа в ТДж ● Энергия, работа в ГДж
- Энергия, работа в МДж

- Энергия, работа в кДж ● Энергия, работа в мДж **Вариант 13.**
- Мощность в Вт (ватт) ● Мощность в ГВт
- Мощность в МВт
- Мощность в кВт
- Мощность в мВт
- Мощность в мкВт

3. Список рекомендуемой литературы

1. Павловская Т. А.C/C++. Программирование на языке высокого уровня: для магистров и бакалавров: учеб. для вузов / Т. А. Павловская. - Гриф МО. - Санкт-Петербург: Питер, 2013. - 460 с. : ил.

2. Professional C++, 3rd Edition. Marc Gregoire. ISBN: 978-1-118-85805-9. Paperback 984 pages. September 2014

4. Контрольные вопросы

1. Какой механизм преобразования от определенного пользователем класса к встроенному типу может быть использован в языке C++?
2. Какой механизм преобразования от встроенного типа данных к определенному пользователем может быть использован в языке C++?
3. Если объект objA принадлежит классу A, объект objB принадлежит классу B, и требуется записать objA = objB, поместив при этом функцию преобразования в класс A, то какую разновидность процедуры преобразования типа можно использовать?
4. Что такое конструктор преобразования?
5. Для чего используется ключевое слово explicit?

Практическое занятие №14. Применение шаблонов классов

Цели:

Получить практические навыки создания шаблонов и использования их в программах C++.

1. Краткие теоретические сведения Шаблон функции

Шаблон функции (иначе параметризированная функция) определяет общий набор операций (алгоритм), которые будут применяться к данным различных типов. При этом тип данных, над которыми функция должна выполнять операции, передается ей в виде параметра на стадии компиляции.

В C++ параметризированная функция создается с помощью ключевого слова **template**.

Формат шаблона функции:

template <class тип_данных> тип_возвр_значения имя_функции(список_параметров){тело_функции}

Основные свойства параметров шаблона функции

* Имена параметров шаблона должны быть уникальными во всем определении шаблона. * Список параметров шаблона не может быть пустым.
* В списке параметров шаблона может быть несколько параметров, и каждому из них должно предшествовать ключевое слово class.

* Имя параметра шаблона имеет все права имени типа в определенной шаблоном функции.

* Определенная с помощью шаблона функция может иметь любое количество непараметризованных формальных параметров. Может быть непараметризованно и возвращаемое функцией значение.

* В списке параметров прототипа шаблона имена параметров не обязаны совпадать с именами тех же параметров в определении шаблона.

* При конкретизации параметризованной функции необходимо, чтобы при вызове функции типы фактических параметров, соответствующие одинаково параметризованным формальным параметрам, были одинаковы.

Шаблон класса

Шаблон класса (иначе параметризованный класс) используется для построения родового класса. Создавая родовой класс, вы создаете целое семейство родственных классов, которые можно применять к любому типу данных. Таким образом, тип данных, которым оперирует класс, указывается в качестве параметра при создании объекта, принадлежащего к этому классу. Подобно тому, как класс определяет правила построения и формат отдельных объектов, шаблон класса определяет способ построения отдельных классов. В определении класса, входящего в шаблон, имя класса является не именем отдельного класса, а параметризованным именем семейства классов.

Общая форма объявления параметризованного класса: template <class тип_данных> class имя_класса { . . . };

Основные свойства шаблонов классов

* Компонентные функции параметризованного класса автоматически являются параметризованными. Их не обязательно объявлять как параметризованные с помощью template.

* Дружественные функции, которые описываются в параметризованном классе, не являются автоматически параметризованными функциями, т.е. по умолчанию такие функции являются дружественными для всех классов, которые организуются по данному шаблону.

* Если *friend*-функция содержит в своем описании параметр типа параметризованного класса, то для каждого созданного по данному шаблону класса имеется собственная *friend*функция.

* В рамках параметризованного класса нельзя определить *friend*-шаблоны (дружественные параметризованные классы).

* С одной стороны, шаблоны могут быть производными (наследоваться) как от шаблонов, так и от обычных классов, с другой стороны, они могут использоваться в качестве базовых для других шаблонов или классов.

* Шаблоны функций, которые являются членами классов, нельзя описывать как virtual. * Локальные классы не могут содержать шаблоны в качестве своих элементов.

Компонентные функции параметризованных классов

Реализация компонентной функции шаблона класса, которая находится вне определения шаблона класса, должна включать дополнительно следующие два элемента:

* Определение должно начинаться с ключевого слова template, за которым следует такой же список_параметров_типов в угловых скобках, какой указан в определении шаблона класса.

* За именем_класса, предшествующим операции области видимости (**::**), должен следовать список_имен_параметров шаблона.

*template<список_типов> тип_возвр_значения имя_класса <список_*имен_ параметров> :: имя_функции (список_параметров)

{ . . . }

2. Практическое задание (100%) Общая постановка:

1. Создать шаблон заданного класса. Определить конструкторы, деструктор, перегруженную операцию присваивания (“=”) и операции, заданные в варианте задания.

2. Определить пользовательский класс, который будет использоваться в качестве параметра шаблона. Определить в классе необходимые функции и перегруженные операции.

3. Написать программу, в которой демонстрируется использование шаблона для стандартного и пользовательского типа.

Методические указания

1. Класс АТД реализовать как динамический массив. Для этого определение класса должно иметь следующие поля:

− указатель на начало массива; −
максимальный размер массива; −
текущий размер массива.

2. Для ввода и вывода определить в классе функции **input** и **print**.

3. Чтобы у вас не возникало проблем, аккуратно работайте с константными объектами. Например:

*конструктор копирования следует определить так: MyTmp (**const** MyTmp& ob);

*операцию присваивания перегрузить так: MyTmp& operator = (**const** MyTmp& ob); 4. Для шаблонов множеств, списков, стеков и очередей в качестве стандартных типов использовать символьные, целые и вещественные типы. Для пользовательского типа взять класс из лабораторной работы № 2 (индивидуальное задание 1).

5. Для шаблонов массивов в качестве стандартных типов использовать целые и вещественные типы. Для пользовательского типа взять класс “комплексное число” complex.

```
class complex{
int re;        // действительная часть int
im;   // мнимая часть public;
// необходимые функции и перегруженные операции
};
```

6. Реализацию шаблона следует разместить вместе с определением в заголовочном файле. 8. Тестирование должно быть выполнено для всех типов данных и для всех операций.

Варианты заданий

1. Класс − одномерный массив. Дополнительно перегрузить следующие операции: [] − доступ по индексу;
= = − проверка на равенство;
!= − проверка на неравенство.

2. Класс − множество set**.** Дополнительно перегрузить следующие операции:

+ − добавить элемент в множество (типа set+item);
+ − объединение множеств;
* − пересечение множеств;

3. Класс − множество set**.** Дополнительно перегрузить следующие операции:
+ − добавить элемент в множество (типа item + set);
+ − объединение множеств;
= = − проверка множеств на равенство.

4. Класс − множество set**.** Дополнительно перегрузить следующие операции:
- − удалить элемент из множества (типа set-item); *
− пересечение множеств;
< − сравнение множеств.

5. Класс − множество set**.** Дополнительно перегрузить следующие операции:
- − удалить элемент из множества (типа set-item); >
− проверка на подмножество;
!= − проверка множеств на неравенство.

6. Класс − множество set. Дополнительно перегрузить следующие операции: + − добавить элемент в множество (типа set+item); * − пересечение множеств; int()− мощность множества.

7. Класс − множество set**.** Дополнительно перегрузить следующие операции:
() − конструктор множества (в стиле конструктора для множественного типа в языке Pascal);
+ − объединение множеств;
<= − сравнение множеств.

8. Класс − множество set**.** Дополнительно перегрузить следующие операции:
> − проверка на принадлежность (типа операции **in** множественного типа в языке Pascal);
* − пересечение множеств;
< − проверка на подмножество.

9. Класс − однонаправленный список list**.** Дополнительно перегрузить следующие операции:
+ − добавить элемент в начало (list+item); --
− удалить элемент из начала (--list);
= = − проверка на равенство.

10. Класс − однонаправленный список list**.** Дополнительно перегрузить следующие операции:
+ − добавить элемент в начало (item+list); --
− удалить элемент из начала (--list);
!= − проверка на неравенство.

11. Класс − однонаправленный список list**.** Дополнительно перегрузить следующие операции:

+ − добавить элемент в конец (list+item); --

− удалить элемент из конца (типа list--);

!= − проверка на неравенство.

12. Класс − однонаправленный список list**.** Дополнительно перегрузить следующие операции:

[] − доступ к элементу в заданной позиции, например:

Type c; int i; list L; c=L[i]; +

− объединить два списка;

= = − проверка на равенство.

13. Класс − однонаправленный список list**.** Дополнительно перегрузить следующие операции:

[] − доступ к элементу в заданной позиции, например: int i; Type c; list L; c=L[i];

+ − объединить два списка;

!= − проверка на неравенство.

14. Класс − однонаправленный список list**.** Дополнительно перегрузить следующие операции:

() − удалить элемент в заданной позиции, например: int i; list L; L(i);

() − добавить элемент в заданную позицию, например: int i; Type c; list L; L(c,i); != − проверка на неравенство.

15. Класс − стек stack**.** Дополнительно перегрузить следующие операции:

+ − добавить элемент в стек;

-- − извлечь элемент из стека; bool() − проверка, пустой ли стек..

3. Список рекомендуемой литературы

1. Павловская Т. А.C/C++. Программирование на языке высокого уровня : для магистров и бакалавров : учеб. для вузов / Т. А. Павловская. - Гриф МО. - Санкт-Петербург: Питер, 2013. - 460 с. : ил.

2. Professional C++, 3rd Edition. Marc Gregoire. ISBN: 978-1-118-85805-9. Paperback 984 pages. September 2014

4. Контрольные вопросы

1. В чем смысл использования шаблонов?
2. Каковы синтаксис/семантика шаблонов функций?
3. Каковы синтаксис/семантика шаблонов классов?
4. Напишите параметризованную функцию сортировки массива методом обмена.
5. Определите шаблон класса “вектор” − одномерный массив.
6. Что такое параметры шаблона функции?
7. Перечислите основные свойства параметров шаблона функции.

8. Как записывать параметр шаблона?
9. Можно ли перегружать параметризованные функции?
10. Перечислите основные свойства параметризованных классов.
11. Может ли быть пустым список параметров шаблона? Объясните.
12. Как вызвать параметризованную функцию без параметров?
13. Все ли компонентные функции параметризованного класса являются параметризованными?
14. Являются ли дружественные функции, описанные в параметризованном классе, параметризованными?
15. Могут ли шаблоны классов содержать виртуальные компонентные функции?
16. Как определяются компонентные функции параметризованных классов вне определения шаблона класса?
17. Объясните каждое из следующих определений шаблонов функций и укажите, нет ли среди них недопустимых. Исправьте все обнаруженные ошибки.

3. template <class T, U, typename V> void fl(T, U, V);
4. template <class T> T f2(int &T);
5. inline template <class T> T foo(T, unsigned int*);
6. template <class T> f4(T, T) ;
7. typedef char Ctype;
8. template <typename Ctype> Ctype f5(Ctype a);

18. Объясните, какие из следующих объявлений являются ошибочными и почему (если они есть).

2. template <class Type> Type bar(Type, Type); template <class Type> Type bar(Type, Type);
3. template <class Tl, class T2> void bar(Tl,T2);
4. template <class CI, typename C2> void bar(CI, C2);

Практическое занятие №15. Обработка исключений

Цели:

Создание консольного приложения, состоящего из нескольких файлов; разработка программы, обрабатывающей исключительные ситуации.

1. Краткие теоретические сведения Механизм обработки исключений.

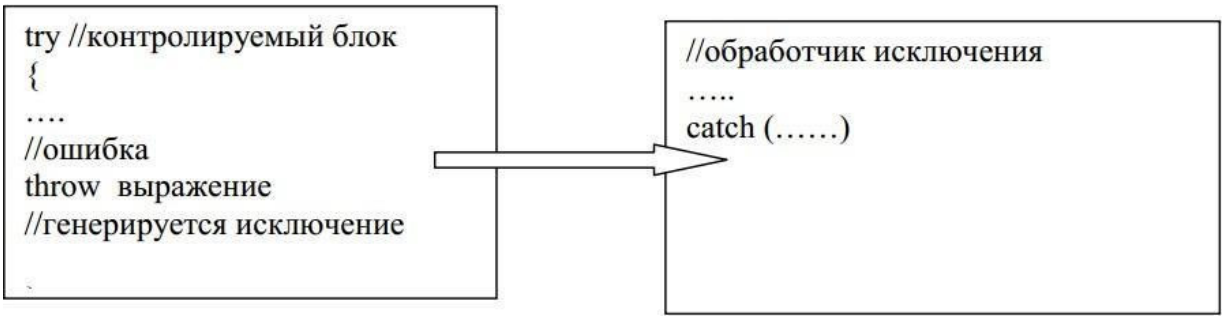
Исключение - это непредвиденное или аварийное событие.

В C++ исключение - это объект, который система должна генерировать при возникновении исключительной ситуации. Генерация такого объекта и создает исключительную ситуацию.

Исключения позволяют разделить вычислительный процесс на 2 части:

1) обнаружение аварийной ситуации (неизвестно как обрабатывать);
2) обработка аварийной ситуации (неизвестно, где она возникла). Достоинства такого подхода:

1) удобно использовать в программе, которая состоит из нескольких модулей;
2) не требуется возвращать значение в вызывающую функцию. Общая схема:



Исключение генерируется оператором
`throw <выражение>,`

где <выражение>- •
либо константа,

• либо переменная некоторого типа, •
либо выражение некоторого типа.

Тип объекта-исключения может быть как встроенным, так и определяемым пользователем. Для представления исключений часто используют пустой класс: `class ZeroDevide{}; class NegativeArg{};`

Генерация исключения будет выглядеть:

`throw ZeroDevide();//вызывается конструктор без параметров` или
`throw new ZeroDevide();`

Исключение надо перехватить и обработать. Для проверки возникновения исключения используется контролируемый блок `try { },` с которым связана одна или несколько секцийловушек `catch`. Все переменные, объявленные в этом блоке, являются локальными для этого блока. Форма записи секции-ловушки следующая:

`catch ( спецификация исключения )`, где спецификация исключения может иметь три формы:

1) (тип имя)
2) (тип)
3) ()

Тип - это встроенный тип или тип, определенный программистом.

Формы 1 и 2 обрабатывают конкретные исключения, а форма 3 перехватывает все исключения, такую ловушку надо помещать последней, тогда она будет обрабатывать все исключения, которые еще не были обработаны.

Форма 1 означает, что объект передается в блок обработки, чтобы его каким-то образом там использовать, например, для вывода информации в сообщении об ошибке.

Примеры:

```
catch( exception e) // по значению catch( exception &e) // по ссылке catch( const exception &e) // по константной ссылке catch( exception *e) //по указателю
```

Лучше всего передавать объект по ссылке, т. к. при этом не создается временный объект- исключение.

Для каждой функции, метода, конструктора или деструктора можно в заголовке указать спецификацию исключений. Если в заголовке спецификация исключений не указана, считается, что функция может порождать любое исключение, если указана, то считается, что функция генерирует те исключения, которые явно указаны в этом списке.

Примеры: `void fl()throw(int,double); void f2 ()throw(ZeroDivide);` Если спецификация имеет вид:

`void f ()throw();` то считается, что функция исключений не генерирует.

Наличие спецификаций исключения не является ограничением при реальной генерации исключений. Но если функция генерирует неспецифицированное исключение, то запускается стандартная функция `unexpected ()` , которая вызывает функцию `terminate ()` , что приводит к аварийному завершению программы.

При отсутствии подходящей секции-ловушки осуществляется вызов стандартной функции завершения `terminate` () . Эта функция вызывается из функции `unexpected` () при нарушении спецификации завершения.

Обе функции можно подменить собственными реализациями. Для этого необходимо

1. Подключить заголовок `#include <expection>`
2. Определить собственную функцию с прототипом `void F ()` для подмены стандартной функции `terminate () .`
3. Указать имя этой функции в вызове функции `set_terminate` (F) ;

После этого вместо `terminate ()` будет вызываться наша функция `F ()` . Такая функция не должна возвращать управление оператором return или генерировать исключение `throw ()` , она может только завершить программу функцией `exit ()` или `abort ()` . Аналогично реализуется подмена стандартной функции unexpected(): `set_ unexpected(F);`

Функция может сгенерировать неспецифицированное исключение, в этом случае, если в спецификации исключений не указано исключение `bad_exception,` вызывается функция `terminate ()` , в противном случае сгенерированное исключение подменяется на `bad exception` и начинается поиск его обработчика.

Стандартные исключения.

В составе стандартной библиотеки C++ реализован ряд стандартных исключений, которые организованы в иерархию классов.

Эта иерархия может служить основой для создания собственных классов исключений и иерархии исключений. Можно определять собственные исключения, унаследовав их от класса `exception.`

Класс `exception` определен в стандартной библиотеке следующим образом:

```
class exception {
public:
   exception () throw();//конструктор без параметров exception
(const exceptions) throw();//конструктор
   копирования
     exceptions operator= (const exceptions) throw();//оператор =
virtual ~exception() throw();//деструктор virtual const
   char*what() const throw();/*генерирует сообщение
```

```
  об ошибке */
};
```

Все конструкторы и методы имеют спецификацию, запрещающую генерацию исключений.

Предполагается, что исключения типа `logic_error` - ошибки в логике программы, например, невыполнение какого- либо

условия; `runtime_error` - ошибки возникают в результате непредвиденных обстоятельств при выполнении программы, например, переполнение при операциях с дробными числами;

Эти исключения программа должна генерировать самостоятельно оператором throw.

Пять стандартных исключений порождают различные механизмы C++. `bad_alloc` генерирует операция new, если не может быть выделена память `bad_cast и bad_typied` генерируются при динамической идентификации типов (RTTI)

`ios_base :: failure` генерируется системой ввода/вывода. `bad_exception` генерируется, если спецификация исключений содержит `bad_exception`

Создание собственной иерархии исключений

Для создания собственной иерархии исключений надо объявить свой базовый класс-исключение, например:

```
class BaseException{};
```

Остальные классы будут наследниками этого класса, аналогично тому, как это сделано в иерархии стандартных исключений:

```
class Child_Exception1rpublic   BaseException{};    class
Child_Exception2rpublic BaseException{};
```

Класс `BaseException` можно унаследовать от стандартного класса

```
exception class BaseException: public exception{};
```

Наследование от стандартных классов позволит использовать метод `what` для вывода сообщений об ошибках.

Иерархия классов-исключений позволяет вместо нескольких разных блоков-ловушек написать единственный блок с типом аргумента базового класса.

2. Задание

1. Реализовать класс, перегрузить для него операции, указанные в варианте.
2. Определить исключительные ситуации.
3. Предусмотреть генерацию исключительных ситуаций.

Ход работы **Пример**

Реализовать класс Вектор. Размер вектора ограничен значением MAX_SIZE=30. Перегрузить для него операции

- доступ по индексу([int i]),
- добавление элемента (+ int),
- удаление элемента из начала вектора (--). Предусмотреть генерацию исключительных ситуаций.

Исключительные ситуации генерируются:

1 - в конструкторе с параметром при попытке создать вектор больше максимального размера;

2, 3 - в операции [] - при попытке обратиться к элементу с номером меньше 0 или больше текущего размера вектора;

4 - в операции + - при попытке добавить элемент с номером больше максимального размера;

- в операции - при попытке удалить элемент из пустого вектора.

ВАРИАНТ РЕАЛИЗАЦИИ 1.

Информация об исключительных ситуациях передается с помощью стандартного типа данных.

1. Создать пустой проект.
2. Добавить в него класс Vector.
3. В файл Vector.h добавить описание класса Vector:

```
#pragma once #include <iostream> using namespace std; const
int MAX_SIZE=30;//максимальный размер вектора class
Vector { int size;//текущий
  размер
int *beg;//указатель на начало динамического массива public:
  Vector(){size=0;beg=0;}//конструктор без параметров
Vector(int s);//конструктор с параметром Vector(int s,int*
mas);//конструктор с параметром Vector(const
Vector&v);//конструктор копирования ~Vector();//деструктор
  const Vector& operator=(const Vector&v);//операция
присваивания int operator[](int i);//доступ по индексу
  Vector operator+(int a);//добавление элемента Vector
operator--();//удаление элемента
  //дружественные функции ввода-вывода
  friend ostream& operator<<(ostream& out,const Vector&v);
friend istream& operator» (istream& in, Vector&v);
};
```

4. В файл Vector.cpp добавить определение методов класса Vector:

```
Vector::Vector(int s)
{
  //если текущий размер больше максимального, то генерируется
исключение if(s>MAX_SIZE) throw 1; size=s; beg=new int [s]; for(int
i=0;i<size;i++) beg[i]=0;
}
Vector::Vector(const Vector &v)
{
  size=v.size; beg=new int [size]; for(int
     i=0;i<size;i++) beg[i]=v.beg[i];
}
Vector::~Vector()
```

```
    {
      if (beg!=0) delete[]beg;
    }
    Vector::Vector(int s, int *mas)
    {
      //если текущий размер больше максимального, то генерируется
исключение if(s>MAX_SIZE) throw 1; size=s;
      beg=new int[size]; for(int i=0;i<size;i++)
         beg[i]=mas[i];
    }
    const Vector& Vector::operator = (const Vector &v)
    {
      if(this==&v)return *this; if(beg!=0) delete []beg;
size=v.size; beg=new int [size]; for(int
      i=0;i<size;i++) beg[i]=v.beg[i];
      return*this;
    }
    ostream& operator<< (ostream& out, const Vector& v)
    {
      if(v.size==0) out<<"Empty\n"; else {for (int
         i=0;i<v.size;i++) out<<v.beg [i] <<" ";
      out<<endl;
      }
      return out;
    }
    istream& operator >>(istream& in, Vector& v)
    {
      for(int i=0;i<v.size;i++)
      {
         cout<<">"; in>>v.beg [i] ;
      }
      return in;
    }
    int Vector::operator [](int i)
    {
      if(i<0)throw 2;//если индекс отрицательный, то генерируется
исключение
      //если индекс больше размер вектора, то генерируется
исключение
      if(i>=size) throw error("Vector length more than
      MAXSIZE\n"); return beg[i];
    }
    Vector Vector::operator +(int a)
    {
      //если при добавлении элемента размер вектора станет больше
      //     максимального,     то     генерируется       исключение
if(size+1==MAX_SIZE) throw 4;
      Vector temp(size+1,beg); temp.beg[size]=a;
      return temp;
```

```
}

Vector Vector::operator --()
{
  //если вектор пустой, то удалить элемент нельзя
  //и генерируется исключение if(size==0) throw
  5; if (size==l) {//если в вектор один элемент
  size=0; delete[]beg; beg=0; return *this;
  };
  Vector temp(size,beg); delete[]beg; size--
  ;
  beg=new int[size]; for(int i=0;i<size;i++)
  beg[i]=temp.beg[i]; return*this;
}
```

5. Добавить в проект файл lab14_main.cpp. В файл записать функцию main(), создающую объекты класса Vector и позволяющую генерировать исключительные ситуации.

```
  #include "Vector.h" #include <iostream> using namespace std;
int main()
{
    //контролируемый блок try {
      Vector x(2);//вектор из двух элементов Vector y;//пустой
   вектор cout<<x;//печать вектора x cout«"Nomer?" ; int i;
   cin>>i;
  //вывод элемента с номером i, если номер больше 2 или меньше 0,
то //генерируется исключительная ситуация cout<<x[i] «endl ;
  //добавление элемента в вектор, если MAX_SIZE=2, то
генерируется //исключительная ситуация y=x+3; cout<<y;
  //удалить один элемент из вектора
        --x; cout<<x;
  //удалить один элемент из вектора
        --x;
        cout<<x;//вектор пустой
  //удалить один элемент из вектора
  //генерируется исключительная ситуация
        --x;
        }
     //обработчик исключения catch(int)
       {cout<<"ERROR!!! "<<endl;}//сообщение об ошибке return 0;
}
```

6. Выполнить тестирование программы с генерацией различных исключительных ситуаций.

ВАРИАНТ РЕАЛИЗАЦИИ 2.

Информация об исключительных ситуациях передается с помощью пользовательского класса.

1. Создать пустой проект.
2. Добавить в него файл error.h.

3. В файл error.h добавить описание класса error:

```
#pragma once #include <string> #include <iostream> using
namespace std; class error //класс ошибка
{
string str; public:
      //конструктор, инициирует атрибут str сообщением об
 ошибке error(string s){str=s;} void what(){cout<<str<<endl;}
 //выводит значение атрибута
  str
};
```

4. Добавить класс Vector. В файл Vector.h добавить описание класса Vector:

```
#pragma once #include <iostream> using namespace
std; const int MAX_SIZE=20; class Vector {int size;
       int *beg;
public:
      Vector(){size=0;beg=0;} Vector(int s); Vector(int s,int*
 mas); Vector(const Vector&v);
      ~Vector() ;
      const    Vector&    operator=(const    Vector&v);      int
 operator[](int i) ;
      Vector operator+(int a); Vector operator--(); friend
      ostream& operator<<(ostream& out, const Vector&
 v); friend istream& operator>>(istream& in, Vector& v);
};
```

5. В файл Vector.cpp добавить определение методов класса Vector:

```
#include "Vector.h" #include "Error.h" #include <iostream>
using namespace std; Vector::Vector(int
s)
{
            if(s>MAX_SIZE) throw error("Vector length more than
 MAXSIZE\n"); size=s; beg=new int [s];
  for(int i=0;i<size;i++) beg[ i]=0;
}
Vector::Vector(const Vector &v)
{
  size=v.size; beg=new int [size]; for(int
     i=0;i<size;i++) beg[i]=v.beg[i];
}
Vector::~Vector()
{
  if (beg!=0) delete[]beg;
}
Vector::Vector(int s, int *mas)
{
  if(s>MAX_SIZE) throw error("Vector length more than
```

```
MAXSIZE\n"); size=s; beg=new int[size];
     for(int i=0;i<size;i++) beg[i]=mas[i];
   }
   const Vector& Vector::operator =(const Vector &v)
   {

     if(this==&v)return *this; if(beg!=0) delete []beg;
size=v.size; beg=new int [size]; for(int
     i=0;i<size;i++) beg[i]=v.beg[i];
     return*this;
   }

   ostream& operator<<(ostream& out, const Vector& v) {
         if(v.size==0) out<<"Empty\n"; else {for (int
         i=0;i<v.size;i++) out<<v. beg [ i]<<" ";
         out<<endl;
         }
         return out;
   }
   istream& operator >>(istream& in, Vector& v)
   {
     for(int i=0;i<v.size;i++)
     {
        Cout<<">"; in>>v.beg [i] ;
     }
     return in;
   }
   int Vector::operator [](int i)
   {
     if (i<0) throw error("index <0"); if(i>=size) throw
error("index>size"); return beg[i];
   }
     Vector Vector::operator +(int a) if(size+l==MAX_SIZE) throw
4; Vector temp(size+1,beg); temp.beg[size]=a;
     return temp;
   }

   Vector Vector::operator --()
   {
     if(size==0) throw error("Vector is empty"); if (size==l)
     {
        size=0; delete[]beg; beg=0; return
        *this;
     };
     Vector temp(size, beg); delete[]beg;
      size--
     ;
     beg=new int[size]; for(int i=0;i<size;i++)
        beg[i]=temp.beg[i];
```

```
  return*this;
}
```

6. Добавить в проект файл lab14_main.cpp. В файл записать функцию main(), создающую объекты класса Vector и позволяющую генерировать исключительные ситуации.

```
  #include "Vector.h”
  #include "Error.h" #include <iostream> using namespace std;
int main()
{
  try
  {
     Vector x(2); Vector y; cout<<x; cout<<"Nomer?"; int i;
     cin>>i;
     cout<<x[i] <<endl ; y=x+3;
     cout<<y; --x; cout<<x; --
     x; cout<<x; --x;
  }

  catch(error e)
     {e.what () ; }
  return 0;
}
```

7. Выполнить тестирование программы с генерацией различных исключительных ситуаций.

ВАРИАНТ РЕАЛИЗАЦИИ 3

Информация об исключительных ситуациях передается с помощью иерархии пользовательских классов.

1. Создать пустой проект.
2. Добавить в него файл error.h.
3. В файл error.h добавить описание иерархии пользовательских классов для определения исключительных ситуаций.

```
  #pragma once #include <string> #include <iostream> using
namespace std; class Error//базовый класс
  {
  public:
        virtual void what(){};
  };
  class IndexError:public Error //ошибка в индексе вектора
  {
  protected:
  string msg; public:
        IndexError(){msg="Index   Error\n";}     virtual   void
   what(){cout<<msg;}
  };
```

```
class SizeError:public Error //ошибка в размере вектора
{
protected:
string msg; public:
      SizeError(){msg="size     error\n";}     virtual       void
 what(){cout<<msg;}
};
class MaxSizeError:public SizeError //превышение максимального
размера
{
protected:
string msg_; public:
      MaxSizeError(){SizeError();msg_="size>MAXSIZE\n";}
 virtual void what(){cout<<msg<<msg_;}
};
class EmptySizeError:public SizeError //удаление из пустого
вектора
{
protected:
string msg_; public:
      EmptySizeError(){SizeError(); msg_="Vector is empty\n";}
 virtual void what(){cout<<msg<<msg_;}
};
class IndexErrorl:public IndexError //индекс меньше нуля
{
protected:
string msg_; public:
      IndexErrorl(){IndexError();msg_="index <0\n";} virtual
 void what(){cout<<msg<<msg_;}
};
class IndexError2:public IndexError //индекс больше текущего
размера

{
protected:
string msg_; public:
      IndexError2(){IndexError();msg_="index>size\n";} virtual
 void what(){cout<<msg<<msg_;}
};
```

4. Добавить класс Vector. В файл Vector.h добавить описание класса Vector:

```
const int MAX_SIZE=20; class Vector {
      int size; int *beg;
public:
      Vector(){size=0;beg=0;} Vector(int s); Vector(int s,int*
 mas); Vector(const Vector&v);
      ~Vector();
      const    Vector&    operator=(const    Vector&v);      int
 operator[](int i);
```

```
      Vector operator+(int a); Vector operator--(); friend
      ostream& operator<<(ostream& out,const Vector& v);
 friend istream& operator>> (istream& in, Vector& v);
};
```

5. В файл Vector.cpp добавить определение методов класса Vector:

```
#include "Vector.h" #include "Error.h" #include <iostream>
using namespace std; Vector::Vector(int s)
{

      if (s>MAX_SIZE) throw MaxSizeError () ; size=s;
      beg=new int [s]; for(int i=0;i<size;i++)
      beg[i]=0;
}
Vector::Vector(const Vector &v)
{
      size=v.size; beg=new int [size]; for(int
          i=0;i<size;i++) beg[i]=v.beg[i];
}


Vector::~Vector()

{
      if (beg!=0) delete[]beg;
}
Vector::Vector(int s, int *mas)
{
      size=s;
      beg=new int [size]; for(int i=0;i<size;i++)
          beg[i]=mas[i] ;
}
const Vectors Vector::operator = (const Vector &v) {
      if(this==&v)return *this; if(beg!=0) delete []beg;
 size=v.size; beg=new int [size]; for(int
      i=0;i<size;i++) beg[i]=v.beg[i] ;
      return*this;
ostream&   operator<<(ostream&   out,   const   Vector&   v)   {
if(v.size==0) out<<"Empty\n"; else {for (int i=0;i<v.size;i++)
out<<v. beg [ i ] <<"
          "; out<<endl;
      }
      return out;
}
istream& operator >>(istream& in, Vector& v)
{
      for(int i=0;i<v.size;i++)
      {
            Cout<<">"; in>>v.beg [i] ;
```

```
        }
        return in;
    }
    int Vector::operator [](int i)
    {

        if(i<0)throw IndexErrorl () ; if(i>=size) throw
     IndexError2 () ; return beg[i];
    }

    Vector Vector::operator +(int a)
    {
        if(size+l==MAX_SIZE) throw MaxSizeError () ; Vector
     temp(size+1,beg); temp.beg[size]=a;
        return temp;
    }

    Vector Vector::operator --()
    {
        if(size==0) throw EmptySizeError(); if (size==l)
        {
            size=0; delete[]beg; beg=0;
            return *this;
        };
        Vector temp(size,beg); delete[]beg; size--;
        beg=new int[size]; for(int i=0;i<size;i++)
        beg[i]=temp.beg[i]; return*this;
    }
```

6. Добавить в проект файл lab14_main.cpp. В файл записать функцию main(), создающую объекты класса Vector и позволяющую генерировать исключительные ситуации.

```
    #include "Vector.h”
    #include "Error.h" #include <iostream> using namespace std;
int main()
    {

       try {
         Vector x(2); Vector y; cout<<x; cout<<"Nomer?" ; int i;
         cin>>i;
         cout<<x [ i ] <<endl ; y=x+3; cout«y;
         --x; cout<<x;
         --x;
         cout<<x;
         --x;
       }
    catch(Error &e)
       {
         e.what();
       }
```

```
return 0;
```

Выполнить тестирование программы с генерацией различных исключительных ситуаций.

2.1. Варианты индивидуального задания (100%)

№	Задание	Вариант реализации
1	Класс- контейнер ВЕКТОР с элементами типа int. Реализовать операции: [] - доступа по индексу; () - определение размера вектора; + число - добавляет константу ко всем элементам вектора; - n- удаляет n элементов из конца вектора.	1,2
2	Класс- контейнер ВЕКТОР с элементами типа int. Реализовать операции: []- доступа по индексу; int() - определение размера вектора; - n - удаляет n элементов из конца вектора; + n - добавляет n элементов в конец вектора.	2,3
3	Класс- контейнер ВЕКТОР с элементами типа int. Реализовать операции: [] - доступа по индексу; + +- добавляет элемент в вектор (постфиксная операция добавляет элемент в конец, префиксная в начало)	3,1
4	Класс- контейнер ВЕКТОР с элементами типа int. Реализовать операции: [] - доступа по индексу; () - определение размера вектора; --удаляет элемент из вектора (постфиксная операция удаляет элемент из конца вектора, префиксная - из начала)	1,2
5	Класс- контейнер ВЕКТОР с элементами типа int. Реализовать операции: [] - доступа по индексу; int() - определение размера вектора; * вектор - умножение элементов векторов a[i]*b[i]; + n - переход вправо к элементу с номером n .	2,3
6	Класс- контейнер МНОЖЕСТВО с элементами типа int. Реализовать операции: [] - доступа по индексу; () - определение размера множества; + - объединение множеств; ++ - добавление элемента в множество .	3,1
7	Класс- контейнер МНОЖЕСТВО с элементами типа int. Реализовать операции: [] - доступа по индексу;	1,2

int() - определение размера вектора;
* - пересечение множеств;
-- - удаление элемента из множества.

8	Класс- контейнер МНОЖЕСТВО с элементами типа int. Реализовать операции: [] - доступа по индексу; = - проверка на равенство; > число - принадлежность числа множеству; - n - переход влево к элементу с номером n.	2,3
9	Класс- контейнер МНОЖЕСТВО с элементами типа int. Реализовать операции: [] - доступа по индексу; != - проверка на неравенство; < число - принадлежность числа множеству; + n - переход вправо к элементу с номером n .	3,1
10	Класс- контейнер МНОЖЕСТВО с элементами типа int. Реализовать операции: [] - доступа по индексу; () - определение размера вектора; - разность множеств; --удаление элемента из множества.	1,2
11	Класс- контейнер СПИСОК с ключевыми значениями типа int. Реализовать операции: []- доступа по индексу; int() - определение размера списка; + вектор - сложение элементов списков a[i]+b[i]; - n - переход влево к элементу с номером n.	2,3
12	Класс- контейнер СПИСОК с ключевыми значениями типа int. Реализовать операции: [] - доступа по индексу; () - определение размера вектора; + число - добавляет константу ко всем элементам вектора; ++ - добавление элемента в конец списка.	1,3
13	Класс- контейнер СПИСОК с ключевыми значениями типа int. Реализовать операции: [] - доступа по индексу; + вектор - добавление списка b к списку a (a+b) + число - добавляет элемент в начало списка;	1,2
14	Класс- контейнер СПИСОК с ключевыми значениями типа int. Реализовать операции: [] - доступа по индексу; () - определение размера списка; * число - умножает все элементы списка на число; - n - переход влево к элементу с номером n.	2,3

15	Класс- контейнер СПИСОК с ключевыми значениями типа int. Реализовать операции: [] - доступа по индексу; int() - определение размера списка; * вектор - умножение элементов списков a[i]*b[i]; +n - переход вправо к элементу с номером n.	3,1

3. Список рекомендуемой литературы

1. Объектно-Ориентированное программирование: учебник / Г. С. Иванова, Т. Н. Ничушкина ; под общ. ред. Г. С. Ивановой. — М. : Изд-во МГТУ им. Н. Э. Баумана, 2014. — 455.

2. Professional C++, 3rd Edition. Marc Gregoire. ISBN: 978-1-118-85805-9. Paperback 984 pages. September 2014

4. Контрольные вопросы

1. Что представляет собой исключение в C++?
2. На какие части исключения позволяют разделить вычислительный процесс? Достоинства такого подхода?
3. Какой оператор используется для генерации исключительной ситуации?
4. Что представляет собой контролируемый блок? Для чего он нужен?
5. Что представляет собой секция-ловушка? Для чего она нужна?
6. Какие формы может иметь спецификация исключения в секции ловушке? В каких ситуациях используются эти формы?
7. Какой стандартный класс можно использовать для создания собственной иерархии исключений?
8. Каким образом можно создать собственную иерархию исключений?
9. Если спецификация исключений имеет вид: void fl()throw(int,double); то какие исключения может прождать функция f 1 ()?
10. Если спецификация исключений имеет вид: void fl()throw(); то какие исключения может прождать функция f 1 ()?
11. В какой части программы может генерироваться исключение?

Практическое занятие № 16. Работа с элементами управления в приложениях Windows Forms

1 ЦЕЛЬ РАБОТЫ

Учиться:

- создавать проекты Windows Forms;
- выполнять отладку приложений Windows Forms;
- работать с основными элементами управления.

ПОРЯДОК ВЫПОЛНЕНИЯ ЗАДАНИЯ

Данная работа выполняется в течение двух часов. Выполнение работы идет одновременно с изучением теоретической части. Для закрепления материала необходимо выполнить упражнения для самостоятельной работы.

3 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ. ТЕХНОЛОГИЯ ВЫПОЛНЕНИЯ РАБОТЫ

3.1 Общие сведения о Windows Forms

Windows Forms – интерфейс программирования приложений (API), отвечающий за графический интерфейс пользователя и являющийся частью Microsoft.NET Framework.

Классы из пространства имен System.Windows.Forms позволяют писать программы, похожие на привычные приложения Windows. Они позволяют использовать кнопки, списки, текстовые поля, меню, окна сообщений и множество других «элементов управления».

Элементы управления – это то, что вы помещаете в форму. Они нужны для вывода информации, например, текстовой (элемент управления Label) или графической (элемент управления PictureBox), либо для выполнения определенных действий, например, выбора значения или перехода к другой форме после нажатия кнопки. Все элементы управления помещаются на форму.

В Windows Forms форма является видимой поверхностью, на которой отображается информация для пользователя. Обычно приложение Windows Forms строится путем помещения элементов управления на форму и написанием кода для реагирования на действия пользователя, такие как щелчки мыши или нажатия клавиш.

При выполнении пользователем какого-либо действия с формой или одним из ее элементов управления, создается событие. Приложение реагирует на эти события с помощью кода и обрабатывает события при их возникно- вении.

Windows Forms включает широкий набор элементов управления, кото- рые можно добавлять на формы. Перечислим некоторые из них:

– Label (Надпись). – Button (Кнопка). – ListBox (Список). – CheckBox (Флажок).

– RadioButton (Переключатель). – MessageBox (Окно сообщений).

– MenuStrip (Меню).

– TabControl (Управление вкладками).

– ToolStrip (Панель инструментов). – TreeView (Дерево).

– DataGrid (Сетка данных). – PictureBox (Изображение).

– RichTextBox (Текстовое поле с поддержкой формата RTF).

Если существующий элемент управления не удовлетворяет потребностям, в Windows Forms можно создать собственные пользовательские элементы управления с помощью класса UserControl.

В состав Windows Forms входят элементы пользовательского интерфейса с расширенными функциями, соответствующими возможностям мощных приложений, таких как Microsoft Office. Используя элементы управления ToolStrip и MenuStrip, можно создавать панели инструментов и меню, содержащие текст и рисунки, отображающие подменю и

содержащие в себе другие элементы управления, такие как текстовые поля и поля с выпадающим списком.

С помощью конструктора Windows Forms Visual Studio, поддерживающего перетаскивание, можно легко создавать приложения Windows Forms: достаточно выделить элемент управления курсором и поместить его на нуж- ное место на форме. Конструктор предоставляет такие средства, как линии сетки и «привязка линий» для преодоления трудностей выравнивания элементов управления. И в случае использования Visual Studio или компиляции из командной строки можно использовать элементы управления FlowLayoutPanel, TableLayoutPanel и SplitContainer для создания продвинутых разметок формы за минимальное время и с минимальными усилиями.

Если необходимо создать свои собственные элементы пользователь ского интерфейса, пространство имен System.Drawing содержит широкий набор классов, необходимых для визуализации линий, кругов и других фигур непосредственно на форме.

3.2 Создание простейшей формы и изменение еѐ свойств

Форма, как и каждый элемент управления, имеет набор свойств, которые позволяют настраивать внешний вид формы и еѐ расположение на экране.

Упражнение 1

В этом упражнении вы создадите форму и измените еѐ свойства.

1. Запустите программу Ms Visual Studio 2010.
2. Выполните команду Файл – Создать – Проект. В появившемся диалого- вом окне Создать проект в списке приложений выберите Приложение Windows Forms, введите имя проекта **Form6_1** и укажите место его сохра- нения. В результате на экране появится окно проекта **Form6_1**. В рабочей области находится форма в режиме конструктора (рисунок 6.1).

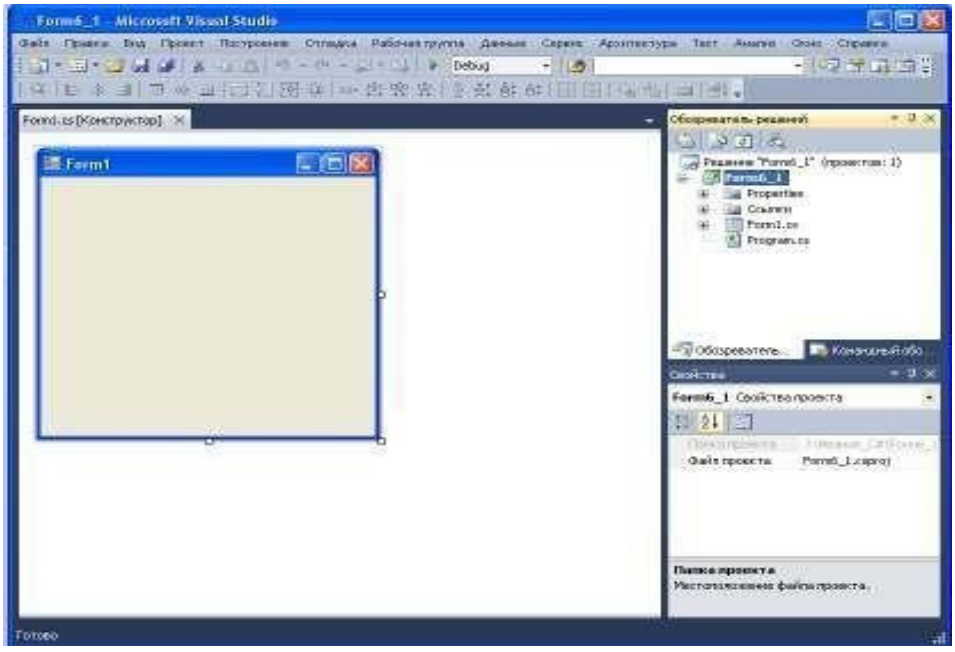


Рисунок 6.1 – Форма в режиме конструктора

3. Измените заголовок формы **Form1** на текст **Калькулятор**. Для этого вы полните щелчок правой кнопкой мышки на форме в режиме конструктора и в контекстном меню выберите команду Свойства. В правом нижнем углу экрана появится окно Свойства. Измените значение свойства Text с **Form1** на **Калькулятор** (рисунок 6.2).

Рисунок 6.2 – Окно Свойства в режиме редактирования свойств формы Form1

4. С помощью свойства Name измените имя формы на **MainForm**.

5. В окне Свойства измените значения свойства Size: ширина – 470, длина – 500.

6. Установите расположение формы после загрузки в центре экрана с по- мощью значения CenterOwner свойства StartPosition.

Свойство StartPosition – расположение формы при загрузке. Значения: – Manual – положение формы на экране, задается свойствами Left и Top.

– CenterScreen – в центре экрана.

– WindowsDefaultLocation – положение задается системой, исходя из количества открытых окон и расположения их на экране.

– WindowsDefaultBounds – форма отображается в месте и с размерами окна, заданными по умолчанию в Windows.

– CenterParent – расположение формы в центре родительской.

7. Измените цвет фона формы на белый с помощью свойства BackColor.

8. Сохраните изменения и закройте решение.

3.3 Добавление на форму элементов управления и выполнение расчетов

Упражнение 2

В этом упражнении вы добавите на форму проекта **Form6_1** элементы управления, которые позволят выполнять операции сложения, вычитания, умножения и деления двух чисел.

1. Откройте проект **Form6_1**.
2. Откройте панель элементов и разместите на форме элементы управления и укажите их идентификаторы (свойство Name) согласно рисунку 6.3.

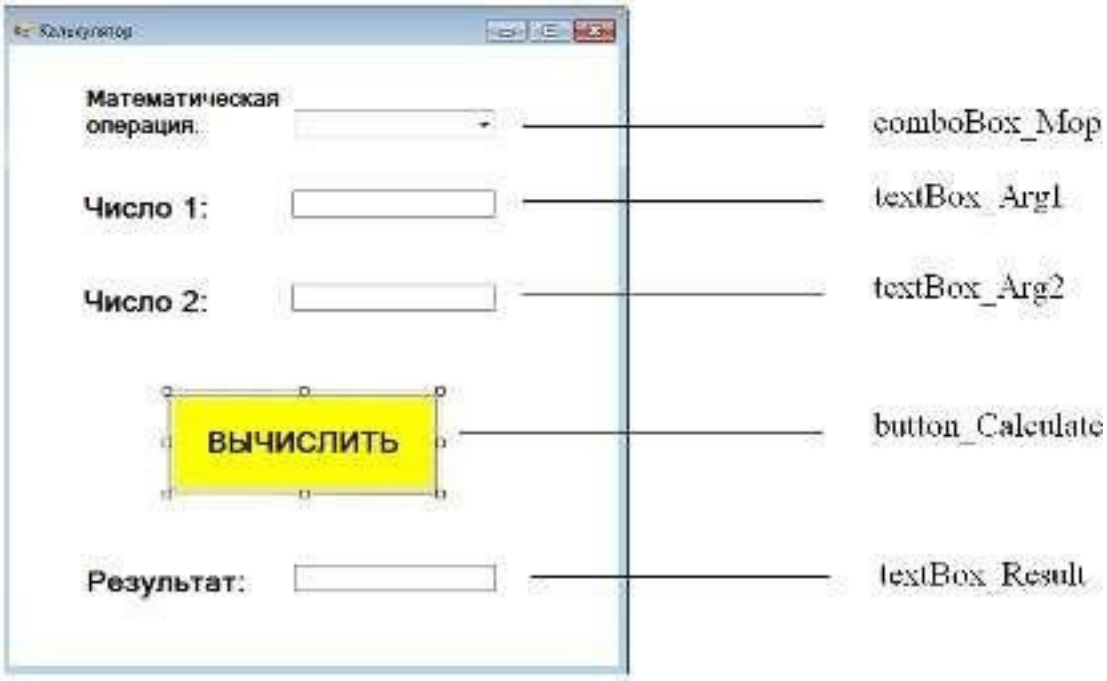


Рисунок 6.3 – Расположение элементов управления на форме к упражнению 2

3. Добавьте обработчик событий загрузки формы. Для этого щелкните по форме и в окне Свойства переключитесь на вкладку работы с событиями элементов управления, нажав на кнопку События  , и дважды кликните по событию Load – справа появится надпись **MainForm_Load** (рисунок 6.4), и в код класса **MainForm** будет добавлен метод-обработчик события загрузки формы с тем же именем (рисунок 6.5).

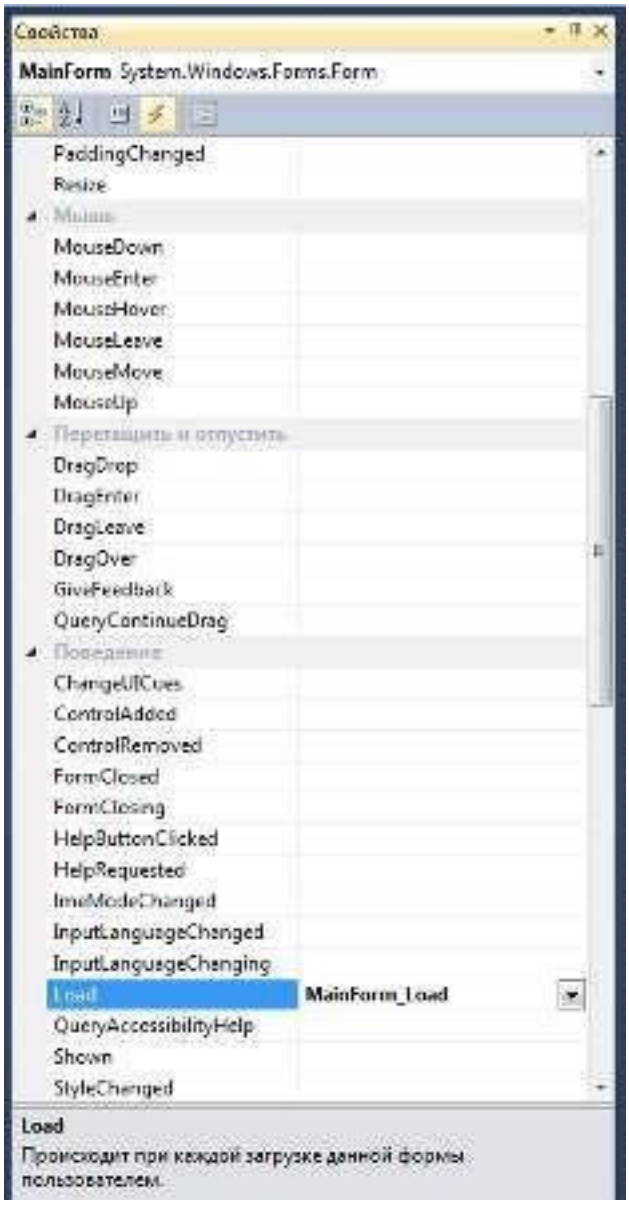


Рисунок 6.4 – Окно Свойства в режиме редактирования событий класса **MainForm** с добавленным обработчиком события Load – методом MainForm_Load

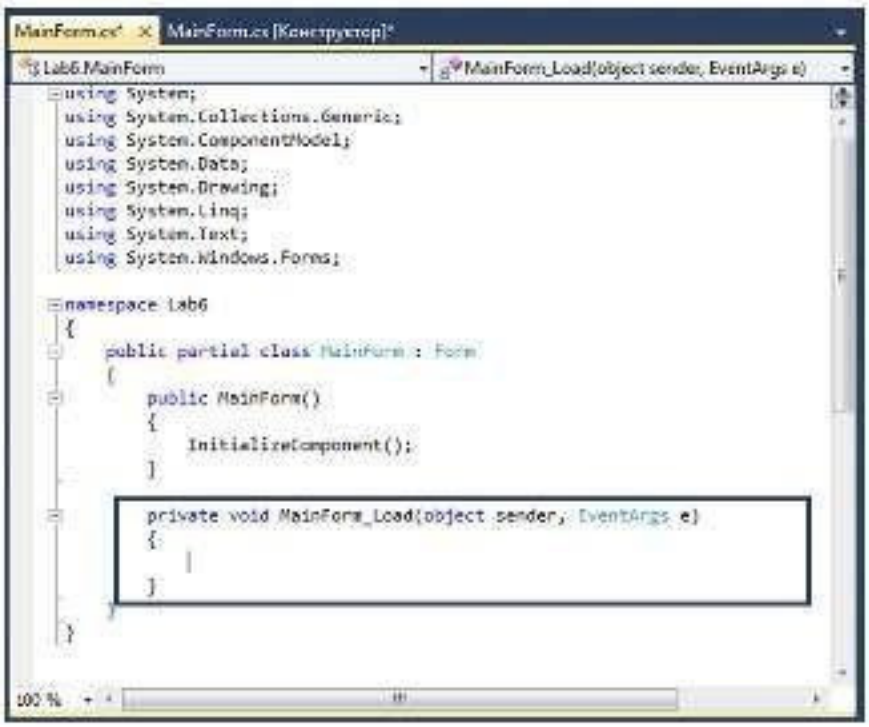


Рисунок 6.5 – Окно редактирования кода класса **MainForm** с добавленным обработчиком события Load – методом MainForm_Load

4. Отредактируйте метод MainForm_Load в соответствии с приведенным ниже кодом

```
using System;
using System.Collections.Generic; using System.ComponentModel; using System.Data; using System.Drawing; using System.Linq; using
System.Text; using System.Windows.Forms;

namespace Form6_1
{ public partial class MainForm : Form
   { public MainForm()
      {
         InitializeComponent();
      }

      private void MainForm_Load(object sender, EventArgs e) {
        // Добавляем элементы в список математических операций
comboBox1_Mop.Items.Add("+");      comboBox1_Mop.Items.Add("-");
comboBox1_Mop.Items.Add("*");
comboBox1_Mop.Items.Add("/");

        // Устанавливаем выбранный элемент с индексом 0
        comboBox1_Mop.SelectedIndex = 0;
      }
   }
}
```

5. Запустите проект на выполнение и убедитесь в заполнении списка мате- матических операций.

6. Дважды щелкните по кнопке Вычислить. Автоматически будет добавлен обработчик события по умолчанию для данного элемента управления (для кнопки это щелчок) – метод button_Calculate_Click. Добавьте в него следу- ющий код:

```
private void button_Calculate_Click(object sender, EventArgs e)
        {
          // Определяем текст, введенный пользователем
          // в поле "Число 1" string sx1 =
          textBox_Arg1.Text;

          // Преобразуем текст в число int
          x1 = Convert.ToInt32(sx1);

          // Определяем текст, введенный пользователем
          // в поле "Число 2" string sx2 =
          textBox_Arg2.Text;

          // Преобразуем текст в число int
          x2 = Convert.ToInt32(sx2);

        // Определяем, какую математическую операцию выбрал пользователь
          string mop = comboBox1_Mop.SelectedItem.ToString(); double result = 0; if
          (mop == "+")
          { result = x1 + x2;
          } else if (mop == "-
          ")
          { result = x1 - x2;
          }  else  if  (mop  ==
          "*")
          { result = x1 * x2;
          } else
          { result = x1 / x2;
          }

          // Выводим результат в текстовое поле
        textBox_Result.Text = " = " + result; }
```

7. Запустите программу на выполнение. Проверьте работу программы при выборе различных математических операций.

8. Запустите программу на выполнение. Выберите математическую опера- цию «деление». В качестве первого числа наберите 7, в качестве второго чис- ла – 0. Нажмите кнопку Выполнить. На экране появится сообщение об ошибке (рисунок 6.6).

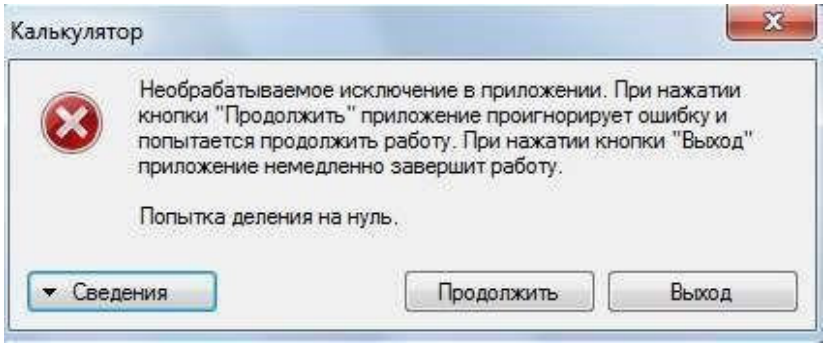


Рисунок 6.6 – Сообщение об ошибке «Деление на ноль»

9. Запустите программу на выполнение еще раз. Выберите любую матема- тическую операцию. В качестве первого числа наберите 5, в качестве второго числа – любую букву. Нажмите кнопку Выполнить. На экране появится со- общение об ошибке (рисунок 6.7).

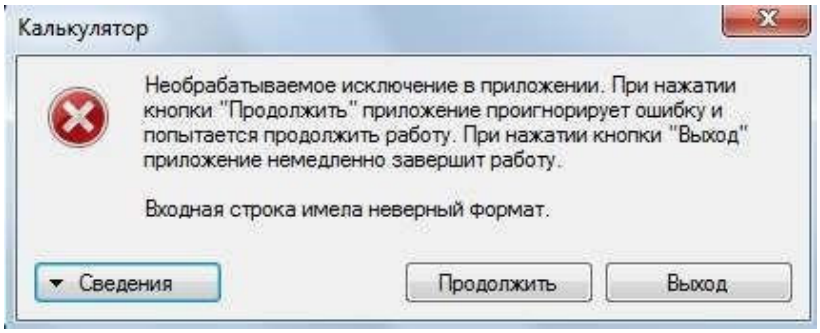


Рисунок 6.7 – Сообщение об ошибке «Неверный формат данных»

10. Сохраните решение и закройте программу.

3.4 Обработка исключительных ситуаций

Иногда при выполнении программы возникают ошибки, которые трудно предусмотреть или предвидеть. Например, при делении двух чисел необходимо предвидеть ситуацию деления на ноль (упражнение 2). Язык C# предоставляет возможности для обработки таких ситуаций.

Основу обработки исключительных ситуаций в C# составляет пара ключевых слов try и catch. Эти ключевые слова действуют совместно и не могут быть использованы порознь.

```
catch (ExcepType1 exOb)
{
   // Обработчик исключения типа ExcepType1.
}
catch (ExcepType2 exOb)
{
   // Обработчик исключения типа ExcepType2.
}
```

... где ExcepType – это тип возникающей исключительной ситуации.

Когда исключение генерируется оператором try, оно перехватывается со- ставляющим ему пару оператором catch, который затем обрабатывает это ис- ключение. В зависимости от типа исключения выполняется и соответствующий оператор catch. Так, если типы генерируемого исключения и того, что указывается в операторе catch, совпадают, то выполняется именно этот опе- ратор, а все остальные пропускаются.

Когда исключение перехватывается, переменная исключения exOb полу- чает свое значение. На самом деле указывать переменную exOb необязатель- но. Так, ее необязательно указывать, если обработчику исключений не требу- ется доступ к объекту исключения, что бывает довольно часто. Для обработ- ки исключения достаточно и его типа.

Следует, однако, иметь в виду, что если исключение не генерируется, то блок оператора try завершается как обычно, и все его операторы catch про- пускаются. Выполнение программы возобновляется с первого оператора, следующего после завершающего оператора catch. Таким образом, оператор catch выполняется лишь в том случае, если генерируется исключение.

Исключения в C# представлены классами. Все классы исключений могут быть унаследованы от встроенного класса исключений Exception, который является частью пространства имен System. Именно поэтому все исключе- ния представляют собой подклассы класса Exception.

Для исключения ошибок: деление на ноль и неправильный ввод дан- ных, добавим в программу «Калькулятор» обработку исключений. ***Упражнение 3***

В этом упражнении вы добавите в программу «Калькулятор» обработку исключений.

1. Откройте проект **Form6_1**.
2. Отредактируйте код программы в соответствии с приведенным ниже ко- дом:

```
using System; using System;
using System.Collections.Generic; using System.ComponentModel; using
System.Data; using System.Drawing; using System.Linq; using System.Text; using
System.Windows.Forms; namespace Form6_1
{
public partial class MainForm : Form
{
public MainForm()
{
InitializeComponent();
}
private void MainForm_Load(object sender, EventArgs e) {
// Добавляем элементы в список математических операций
comboBox1_Mop.Items.Add("+"); comboBox1_Mop.Items.Add("-");
comboBox1_Mop.Items.Add("*"); comboBox1_Mop.Items.Add("/");

// Устанавливаем выбранный элемент с индексом 0 comboBox1_Mop.SelectedIndex
= 0;
}
private void button_Calculate_Click(object sender, EventArgs e) {
```

```
try {
// Определяем текст, введенный пользователем
// в поле "Число 1" string sx1 =
textBox_Arg1.Text; //
Преобразуем текст в число int
x1 = Convert.ToInt32(sx1);
// Определяем текст, введенный пользователем //
в поле "Число 2"
string sx2 = textBox_Arg2.Text;
// Преобразуем текст в число int
x2 = Convert.ToInt32(sx2);

// Определяем, какую математическую операцию выбрал пользователь
string mop = comboBox1_Mop.SelectedItem.ToString(); double result = 0;
if (mop == "+")
{
result = x1 + x2;
}
else if (mop == "-")
{
result = x1 - x2;
}
else if (mop == "*")
{
result = x1 * x2;
}
else
{
result = x1 / x2; }
// Выводим результат в текстовое
поле textBox_Result.Text = " = " +
result; }
// Обрабатываем исключение возникающее при делении на ноль catch
(DivideByZeroException)
{
textBox_Result.Text = "Деление на 0!"; }
// Обрабатываем исключение при неккоректном вводе числа в консоль catch
(FormatException)
{
textBox_Result.Text = "Это НЕ число!";
}
Console.ReadLine();
}
}
}
```

3. Запустите проект на выполнение и убедитесь в обработке исключений.

4. Сохраните решение и закройте программу.

4 ВАРИАНТЫ ЗАДАНИЙ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. Написать программу, которая пересчитывает скорость ветра из «метров в секунду» в «километров в час».

2. Написать программу, которая пересчитывает скорость ветра из «метров в секунду» в «километров в час». Программа должна быть спроектирована таким образом, чтобы пользователь мог ввести в поле «Скорость» только целое положительное число. Написать программу, которая пересчитывает скорость ветра из «метров в секунду» в «километров в час». Вычисление должно выполняться как в результате щелчка на кнопке «Пересчет», так и при нажатии клавиши <*Enter*> после ввода последней цифры в поле «Ско-рость».

3. Написать программу, которая пересчитывает массу из фунтов в килограммы (1 фунт = 409,5 грамм). Программа должна быть спроектирована таким образом, чтобы кнопка «Пересчет» была доступна только в том случае, если пользователь ввел исходные данные.

4. Написать программу, которая пересчитывает массу из фунтов в килограммы (1 фунт = 409,5 грамм). Програм-ма должна быть спроектирована таким образом, чтобы пользователь мог ввести в поле «Масса» только положительное число (целое или дробное).

5. Написать программу, которая вычисляет скорость (км/час), с которой бегун пробежал дистанцию. Количе-ство минут задается целым числом, секунд – дробным.

6. Написать программу, которая вычисляет силу тока в электрической цепи. Программа должна быть спроектирована таким образом, чтобы кнопка «Вычислить» была доступна только в том случае, если пользователь ввел величину сопротивления.

7. Написать программу, которая вычисляет силу тока в электрической цепи. Цепь состоит из двух параллельно соединенных сопротивлений.

8. Написать программу, которая вычисляет стоимость по-купки с учетом скидки. Скидка 1% предоставляется, если сумма покупки больше 300 рублей, 2% - если сумма больше 500 рублей, 3% - если сумма больше 1000 рублей. Информация о предоставленной скидке (процент и величина) должна быть выведена в диалоговом окне.

9. Напишите программу, которая вычисляет стоимость по-купки. Пользователь должен вводить код товара и количество единиц. Программа должна формировать список товаров. Напишите программу, которая вычисляет доход по вкладу. Программа должна обеспечивать расчет про-стых и сложных процентов. Простые проценты начис-ляются в конце срока вклада, сложные – ежемесячно и прибавляются к первоначальной сумме вклада и в следующем месяце проценты начисляются на новую сумму.

10. Написать программу, которая вычисляет сопротивление электрической цепи, состоящей из двух сопротивлений. Сопротивления могут быть соединены последовательно или параллельно. Если величина сопротивления цепи превышает 1000 Ом, то результат должен быть выведен в килоомах.

11. Написать программу, которая вычисляет силу тока в электрической цепи. Цепь состоит из двух сопротивле-ний. Сопротивления могут быть соединены последовательно или параллельно.

12. Написать программу, которая, используя закон Ома, вычисляет силу тока, напряжение или сопротивление электрической цепи. В результате выбора переключателя «Ток», «Напряжение» или

13. «Сопротивление», текст, поясняющий назначение полей ввода должен меняться.

14. Написать программу, которая вычисляет стоимость поездки на автомобиле.

15. Напишите программу-калькулятор, выполняющую сложение, вычитание, умножение и деление двух заданных чисел.

16. Напишите программу «Электронные часы», на поверхности формы которой отображается текущее время.

17. Напишите программу «Электронные часы», в окне которой отображается текущее время и дата.

18. Напишите программу «Электронные часы», в окне которой отображается текущее время, дата и день недели.

Практическое занятие №17 Программирование с использованием многомерных массивов

Цель работы: изучить свойства компонента **dataGridView**. Написать программу с использованием двухмерных массивов.

1. *Двухмерные массивы*

Многомерные массивы имеют более одного измерения. Чаще всего используются двумерные массивы, которые представляют собой таблицы. Каждый элемент массива имеет два индекса, первый определяет номер строки, второй - номер столбца, на пересечении которых находится элемент. Нумерация строк и столбцов начинается с нуля. Объявить двумерный массив можно одним из предложенных способов:

тип **[,]** имя массива**;**

тип **[,]** имя массива **= new** тип **[**размер**1,** размер**2];**

тип **[,]** имя массива**={{**элементы **1-**ой строки**}, … ,** {элементы **n-**ой строки**}};** тип **[,]** имя_строки**}};** массива**= new** тип **[,]{{**элементы **1-**ой строки**}, … ,{**элементы **n-**ой строки**}};**

Пример кода использующего многомерные массивы:

```
// объявление и инициализация двухмерного массива
int[,] array2D = new int[,] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };
// Объявление такого массива с указанием размерности (кол-во строки столбцов) int[,] array2Da = new int[4, 2] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };
// Объявление двухмерного массива элементами, которого являются строки string[,] array2Db = new string[3, 2] { { "one", "two" }, { "three", "four" }, { "five", "six" } };

// Объявление трехмерного массива
int[, ,] array3D = new int[,,] { { { 1, 2, 3 }, { 4, 5, 6 } }, {
{ 7, 8, 9 }, { 10, 11, 12 } } };
// Объявление трехмерного массива с указанием размерности
int[, ,] array3Da = new int[2, 2, 3] { { { 1, 2, 3 }, { 4, 5, 6 } }, {
{ 7, 8, 9 }, { 10, 11, 12 } } };

// Доступ к элементам массива System.Console.WriteLine(array2D[0, 0]);
System.Console.WriteLine(array2D[0, 1]);
System.Console.WriteLine(array2D[1, 0]);
System.Console.WriteLine(array2D[1, 1]);
System.Console.WriteLine(array2D[3, 0]);
System.Console.WriteLine(array2Db[1, 0]);
```

```
System.Console.WriteLine(array3Da[1, 0, 1]); System.Console.WriteLine(array3D[1,
1, 2]);
// Результаты работы программы (выводятся в консоль):
// 1
// 2
// 3
// 4
// 7
// three //
8
// 12
```

2. Элемент управления **DataGridView**

При работе с двухмерными массивами ввод и вывод информации на экран удобно организовывать в виде таблиц. Элемент управления **DataGridView** может быть использован для отображения информации в виде двумерной таблицы. Для обращения к ячейке в этом элементе необходимо указать номер строки и номер столбца. Например: **dataGridView1.Rows[2].Cells[7].Value = "*";** данный код позволяте записать во вторую строку в 7 ячейку знак звездочка.

3. *Случайные числа*

Одним из способов задания массива является задание определение элементов через случайные числа. Для работы со случайными числами используются в основном два метода класса **Random: Random** и **Next.** Метод **Random** подготавливает работу со случайными числами, обеспечивая, надежный способ создания непредсказуемой последовательности чисел. Метод **Random.Next** создает случайное число в диапазоне значений от нуля до **Int32.MaxValue**. Для создания случайного числа в диапазоне от нуля до какого-либо другого положительного числа используется перегрузка метода **Random.Next(Int32)**. Для создания случайного числа в другом диапазоне используется перегрузка метода **Random.Next(Int32, Int32)**.

4. *Порядок выполнения задания*

Задание: Создать программу для определения целочисленной матрицы 15 на

15. Разработать обработчик для поиска минимального элемента на дополнительной диагонали матрицы. Результат, после нажатия кнопки типа **Button**, вывести в **textBox**.

Окно программы приведено на рис. 7.1.

Рис. 1. Окно программы для работы с двухмерным массивом

Текст обработчика события нажатия на кнопку приведен ниже.

```
private void button1_Click(object sender, EventArgs e)
{
dataGridView1.RowCount = 15; //Указываем количество строк dataGridView1.ColumnCount = 15;
                                                    //Указываем количество столбцов int[,] a
= new int[15,15]; //Инициализируем массив
int i,j;
//Заполняем матрицу случайными числами Random rand = new
Random(); for (i=0; i<15; i++) for (j=0; j<15; j++)
a[i,j] = rand.Next(-100,100);
//Выводим матрицу в dataGridView1 for (i=0; i<15; i++) for (j=0;
j<15; j++) dataGridView1.Rows[i].Cells[j].Value =
Convert.ToString(a[i,j]);
//производим поиск максимального элемента на дополнительной
диагонали int m = int.MinValue; for (i = 0; i < 15; i++) if (a[i, 14 - i] > m) m =
a[i, 14 - i];
// выводим результат
textBox1.Text = Convert.ToString(m);
}
```

5. Индивидуальные задания

1) Дана матрица A(3,4). Найти наименьший элемент в каждой строке матрицы. Вывести исходную матрицу и результаты вычислений.

2) Дана матрица A(3,3). Вычислить сумму второй строки и произведение первого столбца. Вывести исходную матрицу и результаты вычислений.

3) Дана матрица A(4,4). Найти наибольший элемент в главной диагонали. Вывести матрицу и наибольший элемент.

4) Дана матрица A(3,4). Найти сумму элементов главной диагонали и эту сумму поставить на место последнего элемента. Вывести исходную и полученную матрицу.

5) Дана матрица A(4,3). Вычислить наибольший элемент матрицы. Вывести исходную матрицу и наибольший элемент.

6) Дана матрица A(4,3).Найти количество положительных элементов.

7) Дана матрица A(3,4).Найти количество отрицательных элементов.

8) Даны матрицы X(15,15) и Y(15,15). Вычислить и вывести элементы новой матрицы $z_{ij}=12x_{ij}-0.85y_{ij}^2$.

9) Даны матрицы A(6,6),B(6,6) и C(6,6). Получить матрицу D(6,6), элементы которой вычисляются по формуле $d_{ij}=\max\left(a_{ij},(b_{ij}+c_{ij})\right)$. Матрицу D(6,6) вывести.

10) Вычислить сумму S элементов главной диагонали матрицы B(10,10). Если S>10, то исходную матрицу преобразовать по формуле $b_{ij}=b_{ij}+13.5$; если S≤10, то $b_{ij}=b_{ij}^2-1.5$. Вывести сумму S и преобразованную матрицу.

11) Дана матрица F(15,15). Вывести номер и среднее арифметическое элементов строки, начинающейся с 1. Если такой строки нет, то вывести сообщение “строки нет”.

12) Дана матрица F(7,7). Найти наименьший элемент в каждом столбце. Вывести матрицу и найденные элементы.

13) Найти наибольший элемент главной диагонали матрицы A(15,15) и вывести всю строку, в которой он находится.

14) Найти наибольшие элементы каждой строки матрицы Z(16,16) и поместить их на главную диагональ. Вывести полученную матрицу.

15) Вычислить суммы элементов матрицы Y(12,12) по столбцам и вывести их.

16) Найти наибольший элемент матрицы A(10,10) и записать нули в ту строку и столбец, где он находится. Вывести наибольший элемент, исходную и полученную матрицу.

17) Дана матрица R(9,9). Найти наименьший элемент в каждой строке и записать его на место первого элемента строки. Вывести исходную и полученную матрицы. 18) Определить количество положительных элементов каждой строки матрицы A(10,20) и запомнить их в одномерном массиве N. Массив N вывести.

19) Вычислить количество H положительных элементов последнего столбца матрицы X(5,5). Если H<3, то вывести все положительные элементы матрицы, если H≥3, то вывести сумму элементов главной диагонали матрицы.

20) Вычислить и вывести сумму элементов матрицы A(12,12), расположенных над главной диагональю матрицы.

Практическое занятие №18. Программирование с использованием средств для отображения графической информации

Цель работы: изучить возможности построения графиков с помощью компонента отображения графической информации C**hart**. Написать и отладить программу построения на экране графика заданной функции.

8.1. *Как строится график с помощью компонента **Chart***

Обычно результаты расчетов представляются в виде графиков и диаграмм. Библиотека .NET Framework имеет мощный элемент управления Chart для отображения на экране графической информации (рис. 8.1).

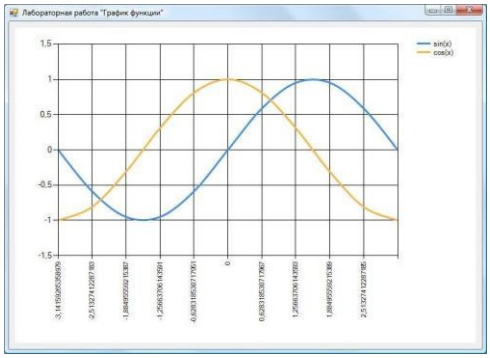


Рис 8.1. Окно программы с элементом управления.

Построение графика (диаграммы) производится после вычисления таблицы значений функции y=f(x) на интервале [Xmin, Xmax] с заданным шагом. Полученная таблица передается в специальный массив Points объекта Series компонента Chart с помощью метода DataBindXY. Компонент Chart осуществляет всю работу по отображению графиков: строит и размечает оси, рисует координатную сетку, подписывает название осей и самого графика, отображает переданную таблицу в виде всевозможных графиков или диаграмм. При необходимости компоненту Chart передаются данные о толщине, стиле и цвете линий, параметрах шрифта подписей, шагах разметки координатной сетки и другие настройки. В процессе работы программы изменение параметров возможно через обращение к соответствующим свойствам компонента Chart. Так, например, свойство AxisX содержит значение максимального предела нижней оси графика и при его изменении во время работы программы автоматически изменяется изображение графика.

8.2. *Пример написания программы*

Задание: составить программу, отображающую графики функций sin(x) и cos(x) на интервале [Xmin, Xmax]. Предусмотреть возможность изменения разметки координатных осей, а также шага построения таблицы.

Прежде всего, следует определить в коде класса все необходимые переменные и константы. Конечно, можно обойтись и без этого, вставляя значения в виде чисел прямо в формулы, но это, во-первых, снизит читабельность кода программы, а во вторых, значительно усложнит изменение каких-либо параметров программы, например, интервала построения графика.

```
/// <summary>
/// Левая граница графика
/// </summary>
private double XMin = -Math.PI;

/// <summary>
/// Правая граница графика
/// </summary>
private double XMax = Math.PI;

/// <summary>
/// Шаг графика ///
</summary>
private double Step = (Math.PI * 2) / 10;
```

```
// Массив значений X - общий для обоих графиков private
double[] x;

// Два массива Y - по одному для каждого графика private
double[] y1; private double[] y2;
```

Также в коде класса следует описать глобальную переменную типа Chart, к которой мы будем обращаться из разных методов:

```
Chart chart;
```

Поскольку данный класс не входит в пространства имен, подключаемые по умолчанию, следует выполнить дополнительные действия. Во-первых, в Обозревателе решений нужно щёлкнуть правой кнопкой по секции Ссылки и добавить ссылку на библиотеку визуализации (рис. 8.2):

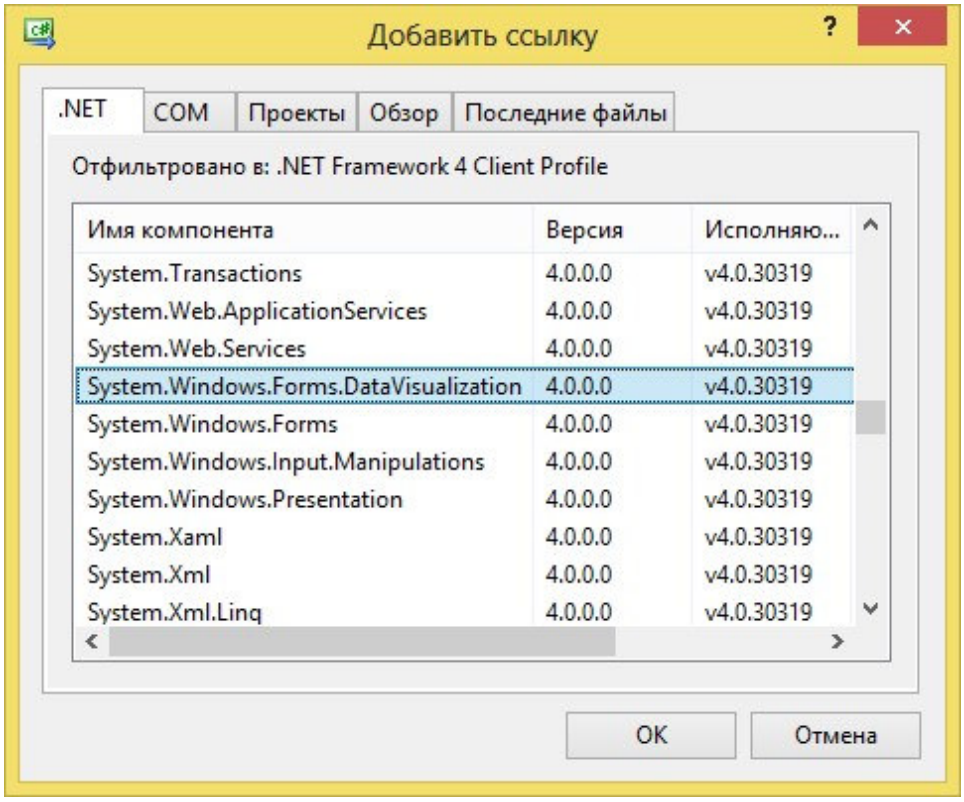


Рис. 8.2. Добавление ссылки на библиотеку визуализации.

Кроме того, следует подключить соответствующее пространство имен:

```
using System.Windows.Forms.DataVisualization.Charting;
```

Далее следует определить метод, который будет расчитывать количество шагов и вычислять значения функций в каждой точке, внося вычисленные значения в массивы *x*, *y1* и *y2*:

```
/// <summary>
/// Расчёт значений графика
/// </summary>
```

```
private void CalcFunction()
{
     // Количество точек графика
     int count = (int)Math.Ceiling((XMax - XMin) / Step) +
                  1;
     // Создаём массивы нужных размеров
     x = new double[count]; y1 = new double[count]; y2 =
     new
double[count];
     // Расчитываем точки для графиков функции for
     (int i = 0; i < count; i++)
     {
          // Вычисляем значение X

          x[i] = XMin + Step * i;
             // Вычисляем значение функций в точке X y1[i] =
     Math.Sin(x[i]); y2[i] =
          Math.Cos(x[i]);
     }
}
```

После расчёта значений нужно отобразить графики на форме с помощью элемента Chart. Элемент управления Chart нельзя выбрать с помощью панели элементов – его нужно создавать прямо в коде программы. Вторым шагом следует созать область отображения графика и настроить внешний вид осей:

```
/// <summary>
/// Создаём элемент управления Chart и настраиваем его
/// </summary>
private void CreateChart()
{
     // Создаём новый элемент управления Chart chart = new
Chart();
     // Помещаем его на форму
     chart.Parent = this; //
     Задаём размеры элемента
     chart.SetBounds(10, 10, ClientSize.Width - 20,
                       ClientSize.Height - 20);

     // Создаём новую область для построения графика
     ChartArea area = new ChartArea();
     // Даём ей имя (чтобы потом добавлять графики) area.Name
= "myGraph";
     // Задаём левую и правую границы оси X
area.AxisX.Minimum = XMin; area.AxisX.Maximum = XMax;
     // Определяем шаг сетки
     area.AxisX.MajorGrid.Interval = Step; //
     Добавляем область в диаграмму
     chart.ChartAreas.Add(area);

     // Создаём объект для первого графика
```

```
        Series series1 = new Series();
        // Ссылаемся на область для построения графика
        series1.ChartArea = "myGraph"; // Задаём тип
        графика - сплайны
        series1.ChartType = SeriesChartType.Spline;
        // Указываем ширину линии графика series1.BorderWidth
        = 3;

        // Название графика для отображения в легенде
        series1.LegendText = "sin(x)";
        // Добавляем в список графиков диаграммы
        chart.Series.Add(series1);
        // Аналогичные действия для второго графика Series
series2 = new Series(); series2.ChartArea = "myGraph";
series2.ChartType = SeriesChartType.Spline;
series2.BorderWidth = 3; series2.LegendText = "cos(x)";
chart.Series.Add(series2);

      // Создаём легенду, которая будет показывать названия
Legend legend = new Legend(); chart.Legends.Add(legend); }
```

Наконец, все эти методы следует откуда-то вызвать. Чтобы графики появлялись сразу после запуска программы, надо вызывать их в обработчике события Load формы:

```
private void Form1_Load(object sender, EventArgs e)
{
    // Создаём элемент управления CreateChart();

    // Расчитываем значения точек графиков функций
    CalcFunction();

    // Добавляем вычисленные значения в графики
chart.Series[0].Points.DataBindXY(x, y1);
chart.Series[1].Points.DataBindXY(x, y2); }
```

8.3. *Выполнение индивидуального задания*

Ниже приведено 15 вариантов задач. По указанию преподавателя выберите свое индивидуальное задание. Уточните условие задания, количество, наименование, типы исходных данных. В соответствии с этим установите необходимое количество окон TextBox, тексты заголовков на форме, размеры шрифтов, а также типы переменных и функции преобразования при вводе и выводе результатов.

Постройте графики функций для соответствующих вариантов. Таблицу данных получить путём изменения параметра X с шагом h. Самостоятельно выбрать удобные параметры настройки.

Практическое занятие №19. Программирование графики

Цель работы: изучить возможности Visual Studio по создание простейших графических изображений. Написать и отладить программу построения на экране разлчиных графических примитивов.

6.1. *Сообщение **WM_PAINT***

Прежде чем приступить к описанию способов рисования в окнах, применяемых приложениями .NET Frameworks, расскажем о том, как это делают «классические» приложения Microsoft Windows.

ОС Microsoft Windows следит за перемещением и изменением размера окон и при необходимости извещает приложения, о том, что им следует перерисовать содержимое окна. Для извещения в очередь приложения записывается сообщение с идентификатором **WM_PAINT**. Получив такое сообщение, функция окна должна выполнить перерисовку всего окна или его части, в зависимости от дополнительных данных, полученных вместе с сообщением **WM_PAINT**.

Для облегчения работы по отображению содержимого окна весь вывод в окно обычно выполняют в одном месте приложения — при обработке сообщения **WM_PAINT** в функции окна. Приложение должно быть сделано таким образом, чтобы в любой момент времени при поступлении сообщения **WM_PAINT** функция окна могла перерисовать все окно или любую его часть, заданную своими координатами.

Последнее нетрудно сделать, если приложение будет хранить где-нибудь в памяти свое текущее состояние, пользуясь которым функция окна сможет перерисовать окно в любой момент времени.

Здесь не имеется в виду, что приложение должно хранить образ окна в виде графического изображения и восстанавливать его при необходимости, хотя это и можно сделать. Приложение должно хранить информацию, на основании которой оно может в любой момент времени перерисовать окно.

Сообщение **WM_PAINT** передается функции окна, если стала видна область окна, скрытая раньше другими окнами, если пользователь изменил размер окна или выполнил операцию прокрутки изображения в окне. Приложение может передать функции окна сообщение **WM_PAINT** явным образом, вызывая функции программного интерфейса Win32 API, такие как **UpdateWindow**, **InvalidateRect** или **InvalidateRgn**.

Иногда ОС Microsoft Windows может сама восстановить содержимое окна, не посылая сообщение **WM_PAINT**. Например, при перемещении курсора мыши или значка свернутого приложения ОС восстанавливает содержимое окна. Если же ОС не может восстановить окно, функция окна получает от ОС сообщение **WM_PAINT** и перерисовывает окно самостоятельно.

6.2. *Событие **Paint***

Для форм класса **System.Windows.Forms** предусмотрен удобный объектно-ориентированный способ, позволяющий приложению при необходимости перерисовывать окно формы в любой момент времени. Когда вся клиентская область окна формы или часть этой области требует перерисовки, форме передается событие **Paint**. Все, что требуется от программиста, это создать обработчик данного события (рис. 8.1.), наполнив его необходимой функциональностью.

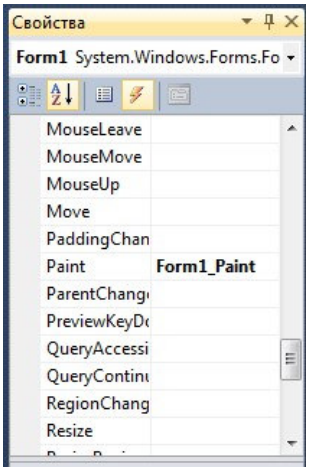


Рис. 6.1. Создание обработчика события Paint

*6.3. Объект **Graphics** для рисования*

Перед тем как рисовать линии и фигуры, отображать текст, выводить изображения и управлять ими в GDI необходимо создать объект **Graphics**. Объект Graphics представляет поверхность рисования GDI и используется для создания графических изображений. Ниже представлены два этапа работы с графикой.

1. Создание объекта Graphics. 2. Использование объекта Graphics для рисования линий и фигур, отображения текста или изображения и управления ими.

Существует несколько способов создания объектов **Graphics**. Одним из самых используемых является получение ссылки на объект **Graphics** через объект **PaintEventArgs** при обработке события **Paint** формы или элемента управления:

```
private void Form1_Paint(object sender, PaintEventArgs e)
   {
   Graphics g = e.Graphics; // Объявляется объект Graphics
   // Далее вставляется код рисования
   }
```

*6.4. Методы и свойства класса **Graphics***

Имена большого количества методов, определенных в классе Graphics, начинается с префикса Draw* и Fill*. Первые из них предназначены для рисования текста, линий и не закрашенных фигур (таких, например, как прямоугольные рамки), а вторые — для рисования закрашенных геометрических фигур. Мы рассмотрим применение только самых важных из этих методов, а полную информацию Вы найдете в документации.

Метод **DrawLine** рисует линию, соединяющую две точки с заданными координатами. Ниже мы привели прототипы различных перегруженных версий этого метода:

public void DrawLine(Pen, Point, Point); public void DrawLine(Pen, PointF PointF; public void DrawLine(Pen, int, int, int, int); **public void DrawLine(Pen, float, float, float, float);**

Первый параметр задает инструмент для рисования линии — перо. Перья создаются как объекты класса **Pen**, например:

Pen p = new Pen(Brushes.Black,2);

Здесь мы создали черное перо толщиной 2 пиксела. Создавая перо, Вы можете выбрать его цвет, толщину и тип линии, а также другие атрибуты.

Остальные параметры перегруженных методов DrawLine задают координаты соединяемых точек. Эти координаты могут быть заданы как объекты класса **Point** и **PointF**, а также в виде целых чисел и чисел с плавающей десятичной точкой.

В классах **Point** и **PointF** определены свойства **X** и **Y**, задающие, соответственно, координаты точки по горизонтальной и вертикальной оси. При этом в классе **Point** эти свойства имеют целочисленные значения, а в классе **PointF** — значения с плавающей десятичной точкой.

Третий и четвертый вариант метода **DrawLine** позволяет задавать координаты соединяемых точек в виде двух пар чисел. Первая пара определяет координаты первой точки по горизонтальной и вертикальной оси, а вторая — координаты второй точки по этим же осям. Разница между третьим и четвертым методом заключается в использовании координат различных типов (целочисленных **int** и с плавающей десятичной точкой **float**).

Чтобы испытать метод **DrawLine** в работе, создайте приложение DrawLineApp (аналогично тому, как Вы создавали предыдущее приложение). В этом приложении создайте следующий обработчик события **Paint**:

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
   Graphics g = e.Graphics; g.Clear(Color.White);
   for (int i = 0; i < 50; i++)
   {
      g.DrawLine(new Pen(Brushes.Black, 2), 10, 4 * i + 20,
 200, 4 * i + 20);
   }
}
```

Здесь мы вызываем метод **DrawLine** в цикле, рисуя 50 горизонтальных линий (рис. 9.2.).

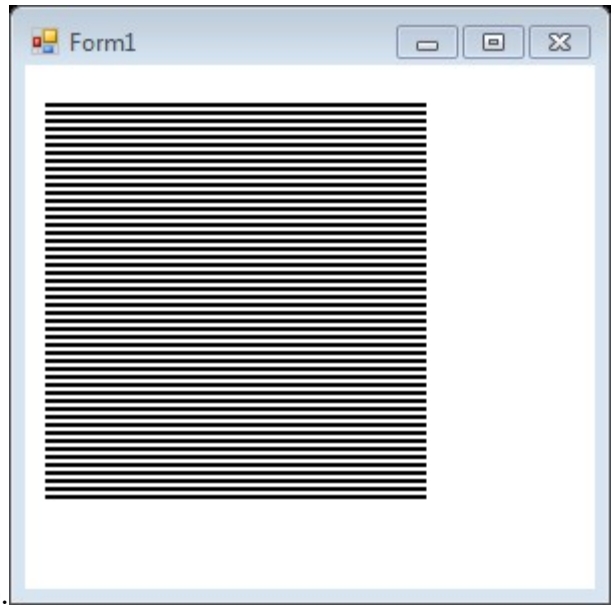


Рис. 6.2. Пример использования DrawLine

Вызвав один раз метод DrawLines, можно нарисовать сразу несколько прямых линий, соединенных между собой. Иными словами, метод **DrawLines** позволяет соединить между собой несколько точек. Координаты этих точек по горизонтальной и вертикальной оси передаются методу через массив класса **Point** или **PointF**:

public void DrawLines(Pen, Point[]); public void DrawLines(Pen, PointF[];

Для демонстрации возможностей метода **DrawLines** создайте приложение. Создайте кисть **pen** для рисования линий:

Pen pen = new Pen(Color.Black, 2);

а также массив точек **points**, которые нужно соединить линиями: Point[] points = new Point[50];

for(int i=0; i < 20; i++) { **int xPos; if(i%2 == 0)** { **xPos=10;** } **else** { **xPos=400;** }

points[i] = new Point(xPos, 10 * i); }

Код будет выглядеть следующим образом:

```
public partial class Form1 : Form
{
    Point[] points = new Point[50]; Pen pen = new Pen(Color.Black, 2);

    public Form1()
    {
        InitializeComponent();
    }

    private void Form1_Paint(object sender, PaintEventArgs e)
    {
        Graphics g = e.Graphics; g.DrawLines(pen, points); }

    private void Form1_Load(object sender, EventArgs e)
    { for (int i = 0; i < 20; i++)
        { int xPos; if (i %
            2 == 0)
            { xPos = 10;
            }
            else
            { xPos = 400;
            }
            points[i] = new Point(xPos, 10 * i); }
    }
}
```

Результат приведен на рис. 6.3.

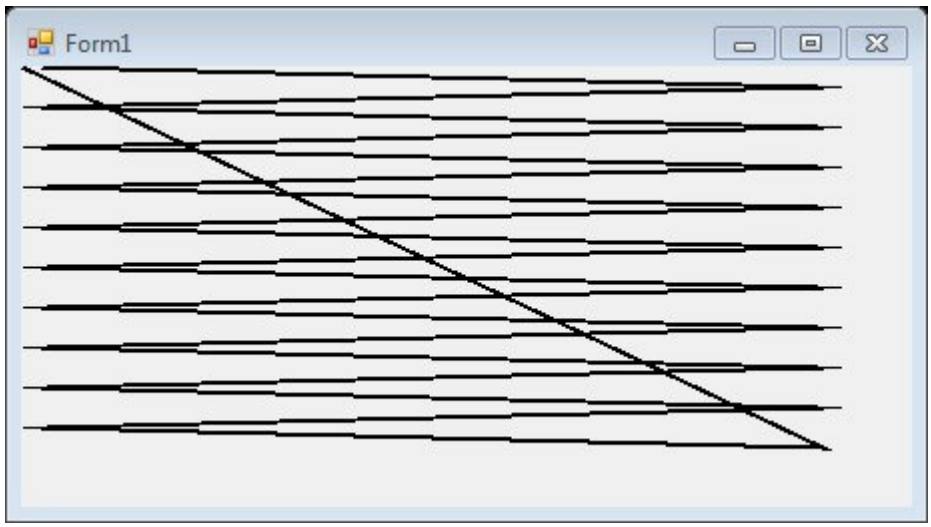


Рис. 6.3. Пример использования массива точек

Для прорисовки прямоугольников можно использовать метод **DrawRectangle(Pen, int, int, int, int);** В качестве первого параметра передается перо класса Pen. Остальные параметры задают расположение и размеры прямоугольника.

Для прорисовки многоугольников можно использовать метод DrawPolygon(Pen, Point[]);

Метод **DrawEllipse** рисует эллипс, вписанный в прямоугольную область, расположение и размеры которой передаются ему в качестве параметров. При помощи метода **DrawArc** программа может нарисовать сегмент эллипса. Сегмент задается при помощи координат прямоугольной области, в которую вписан эллипс, а также двух углов, отсчитываемых в направлении против часовой стрелки. Первый угол Angle1 задает расположение одного конца сегмента, а второй Angle2 — расположение другого конца сегмента (рис. 9.4.).

В классе **Graphics** определен ряд методов, предназначенных для рисования закрашенных фигур. Имена некоторых из этих методов, имеющих префикс Fill: **FillRectangle** (рисование закрашенного прямоугольника), **FillRectangles** (рисование множества закрашенных многоугольников), **FillPolygon** (рисование закрашенного многоугольника), **FillEllipse** (рисование закрашенного эллипса) **FillPie** (рисование закрашенного сегмента эллипса) **FillClosedCurve** (рисование закрашенного сплайна) **FillRegion** (рисование закрашенной области типа **Region**).

Есть два отличия методов с префиксом Fill от одноименных методов с префиксом Draw. Прежде всего, методы с префиксом Fill рисуют закрашенные фигуры, а методы с префиксом Draw — не закрашенные. Кроме этого, в качестве первого параметра методам с префиксом Fill передается не перо класса **Pen**, а кисть класса **Brush**.

Как видете платформа .Net содержит большое число классов со многими методами и свойствами. Нет смысла описывать все классы, методы в каком либо учебнике или в данном пособии, поскольку по любому методу или классу можно получить MSDN справку набрав наименование метода в среде Visual Studio и нажав на нем клавишу F1. Также, при наборе метода в редакотре кода среда показывает краткую справку о передаваемых параметрах.

6.5. Выполнение индивидуального задания

Изучите с помощью справки MSDN методы и свойства классов **Graphics**, **Color**, **Pen** и **SolidBrush**. Создайте собственное приложение выводящий на форму рисунок, состоящий из различных объектов (линий, многоугольников, эллипсов, прямоугольников и пр.), не закрашенных и закрашенных полностью. Используйте разные цвета и стили линий (сплошные, штриховые, штрих-пунктирные).

Практическое занятие №20.Обработка изображений

Цель работы: изучить возможности Visual Studio по открытию и сохранению фалйов. Написать и отладить программу для обработки изображений.

7.1. Отображение графических файлов

Обычно для отображения точечных рисунков, рисунков из метафайлов, значков, рисунков из файлов в формате BMP, JPEG, GIF или PNG используется объект **PictureBox,** т.е. элемент управления **PictureBox** действует как контейнер для картинок. Можно выбрать изображение для вывода, присвоив значение свойству **Image**. Свойство **Image** может быть установлено в окне Свойства или в коде программы, указывая на рисунок, который следует отображать.

Элемент управления **PictureBox** содержит и другие полезные свойства, в том числе: **AutoSize** определяющее, будет ли изображение растянуто в элементе **PictureBox**, и **SizeMode**, которое может использоваться для растягивания, центрирования или увеличения изображения в элементе управления **PictureBox**.

Перед добавлением рисунка к элементу управления **PictureBox** в проект обычно добавляется файл рисунка в качестве *ресурса*. После добавления ресурса к проекту можно повторно использовать его. Например, может потребоваться отображение одного и того же изображения в нескольких местах.

Необходимо отметить, что поле **Image** само является классом для работы с изображениями, у которого есть свои методы. Например, метод **FromFile** используется для загрузки изображения из файла. Кроме класса **Image** существует класс **Bitmap**, коотрый расширяет возможности класса **Image** за счет дополнительных методов для загрузки, сохранения и использования растровых изображений. Так метод **Save** класса **Bitmap** позволяет сохранять изображения в разных форматах, а методы **GetPixel** и **SetPixel** позволяют получить доступ к отдельным пикселям рисунка.

*7.2. Компоненты **OpenFileDialog** и **SaveFileDialog***

Компонент **OpenFileDialog** является стандартным диалоговым окном. Он аналогичен диалоговому окну «Открыть файл» операционной системы Windows. Компонент **OpenFileDialog** позволяет пользователям просматривать папки личного компьютера или любого компьютера в сети, а также выбирать файлы, которые требуется открыть. Для вызова диалогового окна для выбора файла можно использовать метод **ShowDialog()** который возвращет **true** при корректном выборе.

Диалоговое окно возвращает путь и имя файла, который был выбран пользователем в специальном свойстве **FileName**.

7.3. Простой графический редактор

Создайте приложение, реализующее простой графический редактор. Функциями этого редактора должны быть: открытие рисунка, рисование поверх него простой кистью, сохранение рисунка в другой файл. Для этого создайте форму и разместите на ней элементы управления **button** и **picturebox** (рис 11.1).

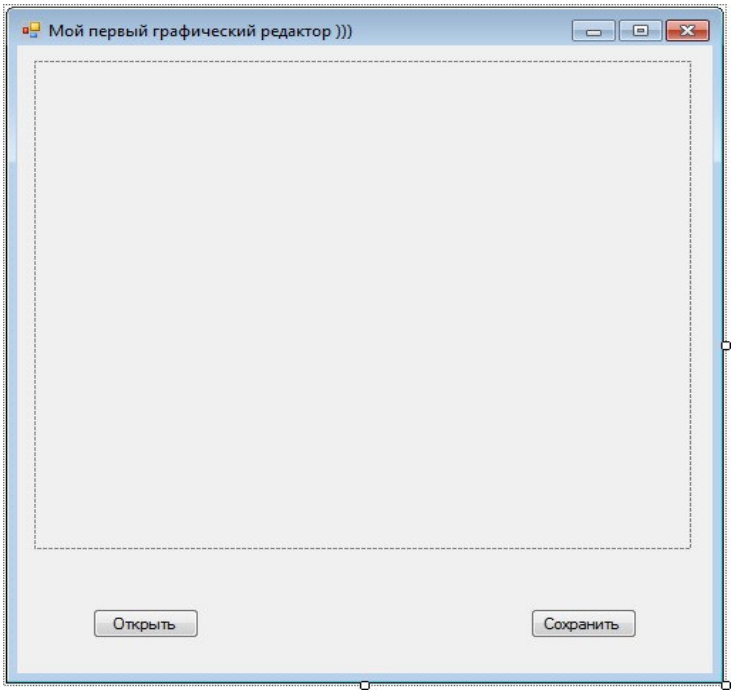


Рис. 7.1. Форма для графического редактора

В этом случае на понадобится из панели элементов размещать на форме компоненты диалоговых окон OpenFileDialog и SaveFileDialog. Эти элементы будут порождены динамически в ходе выполнения программы с помощью конструтора.

Например так:

```
OpenFileDialog dialog = new OpenFileDialog();
```

Далее они будут вызыватся с помощью метода **ShowDialog()**.

Для кнопок «Открыть» и «Сохранить» создайте свои обработчики события. Также создайте обработчик события **Load** для формы. Для элемента управления **picturebox1** создайте обработчики события **MouseDown**, **MouseMove**. Код приложения будет выглядеть следующим образом:

```
using System;
using System.Collections.Generic; using System.ComponentModel; using System.Data;
using System.Drawing; using System.Linq; using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{ public partial class Form1 : Form
    { //Объявляем переменные доступные в каждом обработчике события private
        Point PreviousPoint, point; //Точка до перемещения курсора мыши и
 текущая  точка

private Bitmap bmp; private Pen blackPen; private Graphics g;

public      Form1()      {
InitializeComponent();
            }

            private void Form1_Load(object sender, EventArgs e)
            {
                        blackPen = new Pen(Color.Black, 4); //подготавливаем перо для
```

```
рисования
    }

    private void button1_Click(object sender, EventArgs e)
    {       //открытие файла
OpenFileDialog dialog = new OpenFileDialog(); //описываем и порождаем объект dialog класса OpenFileDialog
        //задаем расширения файлов
        dialog.Filter = "Image files (*.BMP, *.JPG, *.GIF, *.TIF, *.PNG, *.ICO, *.EMF,
*.WMF)|*.bmp;*.jpg;*.gif; *.tif; *.png; *.ico; *.emf; *.wmf";
if (dialog.ShowDialog() == DialogResult.OK)//вызываем диалоговое окно и проверяем выбран ли файл
        {
Image image = Image.FromFile(dialog.FileName); //Загружаем в image изображение из выбранного файла int width = image.Width; int height = image.Height; pictureBox1.Width = width; pictureBox1.Height =
            height;

bmp = new Bitmap(image, width, height); //создаем и загружаем из image изображение в форматье bmp

            pictureBox1.Image = bmp; //записываем изображение в формате bmp в pictureBox1 g = Graphics.FromImage(pictureBox1.Image);
            //подготавливаме объект Graphics для
рисования в pictureBox1

        }
    }

    private void pictureBox1_MouseDown(object sender, MouseEventArgs e)
    { // обработчик события нажатия кнопки на мыши
        // записываем в предыдущую точку (PreviousPoint) текущие координаты PreviousPoint.X = e.X; PreviousPoint.Y
        = e.Y;
    }


    private void pictureBox1_MouseMove(object sender, MouseEventArgs e)
    {//Обработчик события перемещения мыши по pictuteBox1 if (e.Button == MouseButtons.Left) //Проверяем нажата ли левая кнопка
мыши
            {       //запоминаем в point текущее положение курсора мыши
     point.X = e.X;


            point.Y = e.Y;

            //соеденяем линией предыдущую точку с текущей
```

```
                    g.DrawLine(blackPen, PreviousPoint, point);

                        //текущее положение курсора мыши сохраняем в PreviousPoint
                    PreviousPoint.X = point.X; PreviousPoint.Y
                        = point.Y;
                        pictureBox1.Invalidate();//Принудительно вызываем переррисовку
                    pictureBox1
                    }
                }

                private void button2_Click(object sender, EventArgs e)
                { //сохранение файла
                    SaveFileDialog savedialog = new SaveFileDialog();//описываем и
                порождаем объект savedialog

        //задаем свойства для savedialog savedialog.Title = "Сохранить картинку как ...";
savedialog.OverwritePrompt = true; savedialog.CheckPathExists = true; savedialog.Filter =
            "Bitmap File(*.bmp)|*.bmp|" + "GIF File(*.gif)|*.gif|" + "JPEG File(*.jpg)|*.jpg|" + "TIF
    File(*.tif)|*.tif|" + "PNG File(*.png)|*.png"; savedialog.ShowHelp
        = true;
        // If selected, save
        if (savedialog.ShowDialog() == DialogResult.OK)//вызываем диалоговое окно и
проверяем задано ли имя файла
                        {
                        // в строку fileName записываем указанный в savedialog полный
                    путь к файлу string fileName
                        = savedialog.FileName;
                        // Убираем из имени три последних символа (расширение файла)
                    string strFilExtn =
                            fileName.Remove(0, fileName.Length - 3);
                        // Сохраняем файл в нужном формате и с нужным расширением
                    switch (strFilExtn)
                        { case "bmp":
                                bmp.Save(fileName,
                          System.Drawing.Imaging.ImageFormat.Bmp); break; case
                            "jpg":
                                bmp.Save(fileName,
                          System.Drawing.Imaging.ImageFormat.Jpeg); break; case
                            "gif":
                                bmp.Save(fileName,
                          System.Drawing.Imaging.ImageFormat.Gif); break; case
                            "tif":
                                bmp.Save(fileName,
                          System.Drawing.Imaging.ImageFormat.Tiff); break; case
                            "png":
                                bmp.Save(fileName,
                          System.Drawing.Imaging.ImageFormat.Png); break; default:
                                break;
                        }
```

```
            }
        }


    }
}
```

Далее добавим в проект кнопку для перевода изображения в градации серого цвета:

```
        private void button3_Click(object sender, EventArgs e)
            { //циклы для перебора всех пикселей на изображении for (int i = 0; i
    < bmp.Width; i++)
                for (int j = 0; j < bmp.Height; j++) {
    текущей точке текущей точке текущей точке каналов
    int R = bmp.GetPixel(i, j).R; //извлекаем в R значение красного цвета в int G =
bmp.GetPixel(i, j).G; //извлекаем в G значение зеленого цвета в int B = bmp.GetPixel(i, j).B;
//извлекаем в B значение синего цвета в int Gray = (R = G + B)/3; // высчитываем среденее
арифметическое трех
    Color p = Color.FromArgb(255, Gray, Gray, Gray); //переводим int в значение цвета.
255 - показывает степень прозрачности. остальные значения одинаковы для трех каналов
R,G,B bmp.SetPixel(i, j, p); //записываме полученный цвет в текущую
                точку
                }
            Refresh(); //вызываем функцию перерисовки окна
        }
```

Данный код демонстрирует возможность обращения к отдельным пикселям. Цвет каждого пикселя хранится в модели RGB и состоит из трех состовляющих: красного, зеленого и синего цвета, называемых каналами. Значение каждого канала может варьироваться в диапазоне от 0 до 255.

7.4. Выполнение индивидуального задания

Добавьте в приведенный графический редактор свои функции в соответствии с вариантом.

1) Расширьте приложение путем добавления возможности выбора пользователем цвета и величины кисти.
2) Разработайте функцию, добавляющую на изображение 1000 точек с координатами заданными случайным образом. Цвет, также, задается случайным образом.
3) Создайте функцию, переводящую изображение в черно-белый формат.
4) Разработайте функцию, оставляющую на изображении только один из каналов (R,G,B). Канал выбирается пользователем.
5) Создайте функцию, выводящую на изображение окружность. Центр окружности совпадает с центром изображения. Все точки вне окружности закрашиваются черным цветом. Все точки внутри окружности остаются неизменными. Радиус окружности задается пользователем.
6) Создайте функцию, выводящую на изображение треугольник. Все точки вне треугольника закрашиваются синим цветом. Все точки внутри треугольника остаются неизменными.

7) Создайте функцию, выводящую на изображение ромб. Все точки вне ромба переводятся в градации серого цвета. Все точки внутри ромба закрашиваются зеленым цветом. 8) Разработайте функцию, которая выводит на изображение черные горизонтальные линии для каждой четной строки.

9) Разработайте функцию, которая выводит на изображение черные вертикальные линии для каждого нечетного столбца.

10) Создайте функцию, разбивающую изображение на четыре равные части. В каждой оставте значение только одного канала R, G и B, а в четвертой выведите градации серого цвета.

11) Разработайте функцию, заменяющую все точки синего цвета на точки красного цвета.

12) Создайте функцию, инвертирующую изображение в градациях серого цвета в негатив.

13) Создайте функцию, изменяющую яркость изображения. Путем прибавления или уменьшения заданной пользователем величины к каждому каналу.

14) Создайте функцию, переводящую изображение в черно-белый формат в соответсвии с пороговым значением, которое ввел пользователь.

15) Разработайте функцию для создания эффекта мозаики. При этом изображения разбивается на прямоугольные фрагменты, в каждом из которых выбирается цвет средней точки и этим же цветом закрашивается весь фрагмент.

16) Разработайте функцию, разбивающую изображение на фрагменты, в каждом из которых остается только один из каналов (R, G, B).

Практическое занятие №21. Простейшая анимация

Цель работы: изучить возможности Visual Studio по создание простейшей анимации. Написать и отладить программу, выводящую на экран анимационное изображение.

8.1. Работа с таймером

Класс для работы с таймером (Timer) формирует в приложении повторяющиеся события. События повторяются с переодичностью, указанной в милисекундах, в свойстве **Interval**. Установка свойства **Enabled** в значение **true** запускает таймер. Каждый тик таймера порождает событие **Tick**, обработчик которого обычно и создают в приложении. В этом обработчике могут изменятся каике либо величины, и вызватся принудительная перирисовка окна. Напоминаем, что вся отрисовка при создании анимации должна находится в обработчике события **Paint**.

8.2. Создание анимации

Для создании простой анимации достаточно использовать таймер, при тике которого будут изменятся параметры изображения (например, координаты концов отрезка) и обрабочки события **Paint** для рисования по новым параметрам. При таком подходе не надо заботиться об удалении старого изображения (как в идеологии MS DOS), ведь оно создается в окне заново.

В качестве примера рассмотрим код анимации секундной стрелки часов:

```
using System;
using System.Collections.Generic; using System.ComponentModel; using System.Data;
using System.Drawing; using System.Linq; using System.Text; using
System.Windows.Forms;

namespace WindowsFormsApplication1
```

```
{ public partial class Form1 : Form
        { //описываем переменные достыпные в любом обработчике событий класса
 Form1 private int x1, y1, x2, y2, r;

            private double a;
            private Pen pen = new Pen(Color.DarkRed, 2);

            public Form1()
            {
                InitializeComponent();
            }

            private void Form1_Paint(object sender, PaintEventArgs e)
            {
                Graphics g = e.Graphics;
                g.DrawLine(pen, x1, y1, x2, y2); //рисуем секундную стрелку

            } private void Form1_Load(object sender, EventArgs e)
                {          //определяем цент экрана x1 =
                ClientSize.Width / 2; y1 = ClientSize.Height / 2; r = 150;
                //задаем радиус a = 0; //задаем угол поворота
                //определяем конец часовой стрелки с учетом центра экрана x2 = x1 +
        (int) (r * Math.Cos(a)); y2 = y1 - (int)
            (r * Math.Sin(a)); }

            private void timer1_Tick(object sender, EventArgs e)
            { a -= 0.1;//уменьшаем угол на 0,1 радиану
                //определяем конец часовой стрелки с учетом центра экрана x2 = x1 +
        (int)(r * Math.Cos(a)); y2 = y1 - (int)(r
                * Math.Sin(a));
                Invalidate(); //вынудительный вызов перерисовки (Paint)
            }
    }
}
```

8.3. *Выполнение индивидуального задания* Изучите с помощью справки MSDN методы и свойства классов **Graphics**, **Color**, **Pen** и **SolidBrush**. Создайте собственное приложение выводящий на форму рисунок, состоящий из различных объектов (линий, многоугольников, эллипсов, прямоугольников и пр.), не закрашенных и закрашенных полностью. Используйте разные цвета и стили линий (сплошные, штриховые, штрих-пунктирные).

1) Создайте программу, показывающую пульсирующее сердце.
2) Создайте приложение, отображающее вращающийся винт самолета.
3) Разработайте программу анимации двигающегося человечка.
4) Создайте программу, показывающую движение окружности по синусоиде.
5) Создайте приложение, отображающее движение окружности по спирали.
6) Разработайте программу анимации падения снежинки.

7) Создайте программу, показывающую скачущий мячик.

8) Создайте приложение, отображающее движение окружности вдоль границы окна.

9) Разработайте программу анимации летающего бумеранга.

10) Создайте программу, показывающую падение нескольких звезд одновременно.

11) Создайте приложение, отображающее хаотичное движение звезды в окне.

12) Разработайте программу анимации взлета ракеты. Старт осуществляется по нажатию специальной «красной» кнопки.

13) Создайте программу, показывающую движение окружности вдоль многоугольника.

Число вершин вводится пользователем до анимации.

14) Создайте приложение, отображающее броуновское движение молекулы в окне.

15) Разработайте программу анимации движения планет в солнечной системе.

16) Создайте программу, показывающую движение квадратика по траектории, состоящей из 100 точек, и хранящихся в массиве.

17) Создайте приложение, имитирующие механические часы.

18) Разработайте программу анимации падения несколько листков с дерева. Движение не должно быть линейным.

19) Создайте программу, показывающую движение окружности по спирали с плавно изменяющейся скоростью.

20) Создайте приложение, отображающее движение автомобиля с вращающимимся колесами.

Практическое занятие №22. Интегрированный язык запросов LINQ

Цель работы: освоение приемов создания запросов с помощью языка LINQ при работе с массивами данных.

9.1. Запросы LINQ

Запрос **LINQ** (Language-Integrated Query) представляет собой выраже- ние, с помощью которого можно получать данные от источника данных. Запросы обычно выражаются на специальном языке запросов, например, **SQL** для реляционных баз данных и **XQuery** для **XML**.

LINQ – это модель для работы с данными в различных видах источни- ков данных и в различных форматах.

Все операции запроса **LINQ** можно разбить на три различные группы:

1. Получение источника данных.
2. Создание запроса. 3. Выполнение запроса.

9.1.1. Источник данных

Источник данных для выполнения запросов **LINQ** должен поддержи- вать интерфейс **IEnumerable(Of T)**, где **T** – тип элементов источника дан- ных.

В качестве источника данных могут выступать массивы (списки), базы данных и **XML**-документ.

Для использования массива в качестве источника данных достаточно объявление массива и присваивание значений элементам.

Например, если в качестве источника данных используется массив це- лых чисел, его объявление может иметь вид: class Array_INT_LINQ

```
{     static void Main()
  { int[] numbers = new int[7] { 0, 1, 2, 3, 4, 5, 6 ,7 ,8 ,9};
     ….
  }}
```

Массив numbers может использоваться как источник данных.

Если в качестве источника запросов выступает **XML**-документ, необ- ходимо подключение сборки **System.Xml.Linq**, а также объявление пере- менной типа **XElement**. Например: using System.Xml.Linq;

XElement contacts = XElement.Load(@"c:\myList.xml");

Использование **LINQ** для базы **SQL** требует модуль **System.Data.Linq**. Работа с базой данных построена на основе объектно-реляционного сопо- ставления классов приложения и таблиц базы данных. Заранее разработан- ные запросы заполняют списки (массивы) объектов данными из таблиц базы данных.

Для организации связи с базой данных требуется объявление перемен- ной типа DataContext. Например: using System.Data.Linq;

DataContext db = new DataContext (@"c:\northwind\northwnd.mdf");

9.1.2. Создание запроса

Запрос указывает, какую информацию нужно извлечь из источника или источников данных. При необходимости запрос также указывает спо- соб сортировки и группировки. Запрос хранится в переменной запроса и инициализируется выражением запроса. В C# используется синтаксис: *тип переменная запроса* = **from** *переменная диапазона* **in** *источник*

where *условие отбора данных*
group *группировка* **orderby**
сортировка

select *возвращаемое значение* ; Части **where, group, orderby** могут отсутствовать.

В качестве типа может быть использован тип **IEnumerable**<…> либо ключевое слово **var**. Предложение **from** указывает источник данных, пред- ложение **where** применяет фильтр, а предложение **select** указывает тип воз- вращаемых элементов. В **LINQ** переменная запроса хранит сведения, необ- ходимые для предоставления результатов при последующем выполнении запроса.

Например, для массива **numbers** можно задать запрос для выделения всех чётных чисел с помощью объявления IEnumerable<int> evenNumQuery = from num in numbers where ((num % 2) = = 0) or (num % 3) = = 0) select num;

либо

var evenNumQuery = from num in numbers where ((num % 2) = = 0) or (num % 3) = = 0) select num;

9.1.3. Выполнение запроса

Фактическое выполнение запроса откладывается до выполнения ите- рации переменной запроса в операторе **foreach**. Эту концепцию называют *отложенным выполнением*. Например, для цикла

```
foreach (int num in numQuery)
{
  Console.Write("{0,1} ", num);
```

} оператор **foreach** извлекает один элемент результата запроса.

Тип переменной **num** должен соответствовать типу данных источника данных. Например, в предыдущем запросе переменная итерации **num** со- держит каждое (по очереди) значение, которое делится на 2 или 3 без остатка.

Запросы, выполняющие статистические функции над диапазоном ис- ходных элементов, должны сначала выполнить выборку этих элементов. Примерами таких запросов являются **Count**, **Max**, **Min**, **Average** , **Last** и **First**. Они выполняются без явного оператора **foreach**. Следующий запрос возвращает количество четных чисел в исходном массиве. var evenNumQuery = from num in numbers where (num % 2) = = 0 select num;

```
int evenNumCount = evenNumQuery.Count();
```

Для вычисления значений функций **Max**, **Min**, **Average**, **Last** и **First** требуется преобразование к простой последовательности с помощью метода **ConvertAll**<*mun*>. Например, для анализа поля года может быть использована следующая последовательность команд:

```
var StudentQuery6=
                    (from st in students select st ).ToList().ConvertAll<Int32> (delegate
 (Student r) { return (r.God); }); int v1 = (Int32)
StudentQuery6.Average(); int v3 = StudentQuery6.Max(); int v4 =
StudentQuery6.Min();
```

Чтобы принудительно вызвать немедленное выполнение любого за- проса и кэшировать его результаты, можно вызвать метод **ToList(Of TSource)** или **ToArray(Of TSource)**. Например:

```
List<int> numQuery2 =
(from num in numbers where (num % 2) = = 0 select num).ToList(); или var
numQuery3 =
  (from num in numbers where (num % 3) = = 0 select num).ToArray();
```

9.1.4. Переменная запроса LINQ

Переменные запросов **LINQ** определены как **IEnumerable(Of T)** или как производный тип, например **IQueryable(Of T)**. Если переменная запро- са имеет тип **IEnumerable< Student >**, это означает, что запрос при выпол- нении выведет последовательность объектов **Student** (ноль или более эле- ментов). Например: public class Student

```
{ public string FirstName { get; set; } public string LastName { get; set; } public int
ID { get; set; } public int God { get; set; } public List<int> Ball;
} static List<Student> students = new List<Student>
        { new Student {FirstName="Иван", LastName="Дягтерёв", ID=111, Ball=
 new List<int> {97, 92, 81, 60}},
          new Student {FirstName="Михаил", LastName="Остапенко", ID=122,
  Ball= new List<int> {94, 92, 91, 91} }
};
IEnumerable< Student > StudentQuery =
from st in students where st. First = = "Иван " select st; foreach (Student cst in
StudentQuery)
{ Console.WriteLine(cst.LastName + ", " + cst.FirstName); }
```

Вместо объявления типа в запросе можно использовать ключевое сло- во **var**. Ключевое слово **var** сообщает компилятору о необходимости опре- деления типа переменной запроса с помощью просмотра источника данных, указанного в предложении **from**. Например: var customerQuery2 =

from st in students where st. First = = "Иван " select st; foreach(var cst in StudentQuery)

{ Console.WriteLine(cst.LastName + ", " + cst.FirstName); } Переменная **cst** будет иметь тип поля элемента **st** запроса.

Ключевое слово **var** удобно, когда тип переменной является очевид- ным или когда не требуется явно указывать тип.

9.1.5. Переменная диапазона запроса

В первую очередь в запросе **LINQ** нужно указать источник данных. В запросе **LINQ** первым идет предложение **from** для указания *источника данных* (**students**) и *переменная диапазона* (**st**).

var queryAllStudents = from st in students select st;

Переменная диапазона имеет сходство с переменной итерации в цикле **foreach** за исключением того, что в выражении запроса не происходит фактической итерации. При выполнении запроса переменная диапазона будет использоваться как ссылка на каждый последующий элемент в **students**. Поскольку компилятор может определить тип **st**, нет необходимости указывать его в явном виде.

9.1.6. Фильтрация

Фильтр приводит к возвращению запросом только тех элементов, для которых выражение является истинным. Фильтр задается с помощью части **where**. Фильтр содержит выражение, которое возвращает логическое значе- ние для очередного элемента последовательности. В следующем примере возвращаются записи, в которых поле FirstName равно "Иван". var customerQuery2 = from st in students where st. FirstName = = "Иван " select st;

Условия отбора элементов можно комбинировать с использованием операции **AND (и)** и **OR (или)**. В C# операции и/или записываются как

«&&» и «||» соответственно. Условия можно группировать с помощью скобок. Например: where (st.FirstName = ="Михаил" || st.LastName = = "Дягтерёв") && God=1999

Определяет условие отбора студентов 1999 года рождения с именами "Михаил" или с фамилией "Дягтерёв".

9.1.7. Сортировка

Предложение **orderby** обеспечивает выдачу элементов последовательности в зависимости от правила сравнения по умолчанию для сортируемого типа. Например, следующий запрос может быть расширен для сортировки результатов на основе свойства Name. Поскольку Name является строкой, сравнение по умолчанию выполняется в алфавитном порядке. var query3 = from st in customers where st.God = = 2001 orderby

st.LastName ascending select st;

Для упорядочения результатов в обратном порядке используется пред- ложение **orderby**…**descending**.

9.1.8. Группировка

Предложение **group** позволяет группировать результаты на основе указанного ключа. Например, можно указать, что результаты должны быть сгруппированы по **God** так, чтобы все

студенты одного года рождения были в одной группе. var queryByGod = from st in Students group cust by st.God, FirstName; foreach (var

stGroup in queryByGod)

{ Console.WriteLine(customerGroup.Key); foreach (var cst in stGroup) { Console.WriteLine(" {0}", cst.FirstName); } }

Когда запрос завершается предложением **group**, результаты представ- ляются в виде списка из списков. Каждый элемент в списке является объектом, имеющим поле **Key** и список членов группы, сгруппированных по этому ключу. При итерации запроса, создающего последовательность групп, необходимо использовать вложенный цикл **foreach**. Внешний цикл выполняет итерацию каждой группы, а внутренний цикл — итерацию чле- нов каждой группы.

Если необходимо ссылаться на результаты операции группировки, можно использовать ключевое слово **into** для создания идентификатора, который можно будет запрашивать. Следующий запрос возвращает только те группы, которые содержат более двух заказчиков. var custQuery = from cust in Students group cust by cust.Last into custGroup where

custGroup.Count() > 2 orderby custGroup.Key select custGroup;

9.1.9. Выбор (проецирование) полей объекта

Предложение **select** создает результаты запроса и задает форму или тип каждого возвращаемого элемента. Например, можно указать, будут ли результаты состоять из полных объектов **Students**, только из одного члена, подмножества членов или некоторых других типов данных, на основе вы- числений или создания новых объектов. Когда предложение **select** создает что-либо отличное от копии исходного элемента, операция называется *про- екцией*.

Для проецирования результата запроса в части **select** реализуют созда- ние нового объекта анонимного класса (класс не имеет собственного име- ни) по формату: from *запрос* select new { *список полей* }

Например, если из списка **List<Student>** требуется выбрать только по- ля фамилии и года рождения, то запрос может иметь вид: from st in students select new { st.LastName, st.God };

9.2. Пример использования запроса LINQ

Пусть имеются студенты, которые учатся в различных группах. Каж- дый студент имеет следующие характеристики: имя, фамилия, год рождения, номер группы, набор оценок, полученных в сессию. Для хранения информации о студенте может быть использован класс: public class Student

{ public string FirstName { get; set; } public string LastName { get; set; } public int ID { get; set; } public string Ngr { get; set; } public int God { get; set; } public List<int> Ball; }

Дополнительное поле **ID** добавлено для однозначной идентификации студента. Для хранения данных обо всех студентах можно использовать список:

List<Student> students;

Первоначальное заполнение списка студентов может быть осуществ- лено при создании: students = new List<Student>

{new Student {FirstName="Светлана", LastName="Богатырева", ID=111, Ngr = ="ИФ-51в", God=1990, Ball= new List<int> {4, 3, 5, 4}}, new Student {FirstName="Иван", LastName="Поддубный", ID=122, Ngr =

```
="ИФ-51в", God=1991, Ball= new List<int> {4, 3, 4, 3}}, new Student
        {FirstName="Марина", LastName="Шаталина", ID=133, Ngr =
="ИФ-51г", God=1990, Ball= new List<int> {5, 5, 5, 5}}, new
Student {FirstName="Петр", LastName="Бержной", ID=144,
        Ngr = ="ИФ-51г", God=1989, Ball= new List<int> {4, 4, 5, 5}}
};
```

Запрос на выдачу данных по заданному значению имени студента (ис- пользование фильтра):

```
IEnumerable<Student> StudentQuery = from st in students where st.FirstName = =
    "Светлана" select st; foreach (Student
cst in StudentQuery)
    {
      Console.WriteLine(cst.LastName + " " + cst.FirstName); }
```

Запрос на выдачу всех данных в отсортированном по фамилии виде:

```
var
    StudentQuery1 =
    from st in students orderby st.LastName ascending select st; foreach (Student
cst in StudentQuery1)
    {
      Console.WriteLine(cst.LastName + " " + cst.FirstName + " " + cst.God
                                + " " + cst.Ngr);
      foreach (int i in cst.Ball)
      { Console.Write(i + " "); }; Console.WriteLine();
    }
```

Использование сложного фильтра по «И»:

```
IEnumerable<Student>
StudentQuery2 = from st in students where st.Ngr = = "ИФ- 51г" && st.God>1988
    select st;
foreach (Student cst in StudentQuery2)
    {
      Console.WriteLine(cst.LastName + " " + cst.FirstName); }
```

Использование сложного фильтра по «ИЛИ»:

```
IEnumerable<Student>
StudentQuery3 =
    from st in students where st.Ngr = = "ИФ-51г" || st.Ngr = = "ИФ-51в" select st;
    foreach (Student cst in StudentQuery3)
    {
      Console.WriteLine(cst.LastName + " " + cst.FirstName); }
```

Группировка записей по году рождения может быть выполнена с помощью следующего запроса:

```
var StudentQuery4 =
    from st in students group st by st.God into f select f; foreach (var nameGroup
in StudentQuery4) {
    Console.WriteLine("Key: {0}", nameGroup.Key); foreach (var student in
nameGroup)
{Console.WriteLine("\t{0}, {1}", student.LastName, stu- dent.FirstName); }
}
```

Следует обратить внимание на тип переменной **StudentQuery4** – **var.** Выбор типа **var** позволяет иметь группы произвольного типа. Наличие **into f** обеспечивает формирование списка по группам. Каждая группа имеет дополнительное поле **Key**.

При необходимости осуществлять выборку только некоторых полей можно применить операцию **new** для формирования нового класса (ано- нимного) с указанием полей основного класса, которые вошли в новый класс.

```
var StudentQuery5 =
from st in students orderby st.LastName ascending select new { st.LastName, st.FirstName,
st.God }; foreach (var cst in
StudentQuery5)
{
Console.WriteLine(cst.LastName + "\t " + cst.FirstName + "\t " + cst.God);
}
```

9.3. Задание для выполнения лабораторной работы

Разработать консольное приложение для работы с множеством данных в виде списка (массива).

Разработать запросы к списку с использованием типов **IEnumerable**<…>
и **var**:

а) вывод всей информации из списка;

б) с использованием фильтра на отдельное поле и на совокупность полей;

в) в отсортированном виде по одному и нескольким полям; г) с группировкой данных по одному и нескольким полям; д) с формированием проекции.

Варианты заданий:

1. Разработать структуру данных для хранения информации о книгах.

Книга характеризуется: названием, фамилией автора, стоимостью, датой издания, издательством, списком инвентарных номеров (книга в нескольких экземплярах).

2. Разработать структуру данных для хранения информации о товарах на складе. Товар характеризуется: названием, стоимостью, количеством (в штуках), списком дат поступления на склад, производителем (наименование фирмы).

3. Разработать структуру данных для хранения информации о вычислительной технике на предприятии. Вычислительная техника характеризуется: названием, стоимостью, вычислительной мощностью, списком лиц, эксплуатирующих вычислительную технику.

4. Разработать структуру данных для хранения информации о проектах, выполняемых на предприятии. Для проекта хранится информация: шифр проекта, наименование проекта, стоимость работ для выполнения проекта, дата начала проекта и дата окончания проекта, список лиц, участвующих в проекте.

5. Разработать структуру данных для хранения информации о кинотеатрах города. Для кинотеатра хранится информация: наименование кинотеатра, вместительность (количество мест), год постройки, ранг кинотеатра (для просмотра видеофильмов, для просмотра широкоформатных фильмов, наличие стереоформатного оборудования и т.п.).

6. Разработать структуру данных для хранения информации о троллейбусных маршрутах города. Для каждого маршрута хранится информация: наименование начальной и конечной остановки, количество троллейбусов на маршруте, время проезда от начала маршрута до конца, список номеров троллейбусов на маршруте.

Контрольные вопросы

1. Для чего используется запрос **LINQ**?
2. Какие основные этапы применения запросов **LINQ**?
3. Что может выступать в качестве источника данных в **LINQ**?

4. Какой интерфейс должен поддерживать источник данных?

5. Какой модуль требуется подключать, если в качестве источника данных используется **XML**-документ, база данных?

6. Какие части содержит запрос **LINQ**? Для чего они используются?

7. Какой оператор обеспечивает фактическое выполнение запроса?

8. Что называется отложенным выполнением запроса?

9. Как принудительно вызвать немедленное выполнение запроса?

10. Для чего используется группировка данных?

11. Какие агрегирующие функции можно использовать для работы с группой?

12. Для чего используется переменная запроса?

13. Как задается фильтр запроса? С помощью каких операций можно создавать сложные фильтры?

14. С помощью какого предложения задается сортировка при выдаче значений от источника данных? Как указывается сортировка по возрастанию и по убыванию?

Практическое занятие №23. Создание выходной последовательности элементов в LINQ

Цель работы: получение навыков формирования выходных данных для выполнения запросов с использованием **LINQ**

10.1. Создание выходной последовательности запросов

С помощью запроса **LINQ** можно использовать исходную последова- тельность в качестве входных данных и изменять ее различными способами для создания новой выходной последовательности:

- изменить порядок элементов в последовательности при помощи сор- тировки и группировки, не изменяя элементов;
- объединить несколько входных последовательностей в одну выход- ную последовательность, которая имеет новый тип;
- создать выходные последовательности, элементы которых состоят только из одного или нескольких свойств каждого элемента в исходной последовательности (проецирование);
- создать выходные последовательности, элементы которых состоят из результатов операций, выполняемых над исходными данными;
- создать выходные последовательности в другом формате. Например, можно преобразовать данные из строк **SQL** или текстовых файлов в **XML**.

Выходные последовательности одного запроса могут использоваться как входные последовательности для нового запроса.

10.1.1. Сортировка и группировка

При необходимости получать элементы выходной последовательности в определенном порядке может быть выполнена сортировка.

Сортировка данных осуществляется в зависимости от правила сравне- ния по умолчанию для сортируемого типа.

Для сортировки последовательности по одному или нескольким клю- чам используется оператор **orderby**:

orderby *поле1 направление сортировки, поле2 направление сортировки*

После ключевого слова **orderby** через запятую может быть указано одно или несколько полей элемента последовательности с указанием направления сортировки: **ascending** – по возрастанию (по умолчанию), **descending –** по убыванию.

Например, следующий запрос может быть расширен для сортировки результатов на основе свойства **Name** (сравнение выполняется в лексико- графическом порядке).

```
var queryLondonCustomers3 = from cust in customers
where cust.City = = "London" orderby cust.Name ascending select cust;
```

Группировка данных позволяет изменить порядок в выходной после- довательности так, что в одну группу будут входить элементы с одинако- вым значением поля группировки.

group *переменная диапазона* **by** *поле группировки* **into** *имя группы.*

Имя группы может использоваться в части **where select** запроса вместо переменной диапазона.

```
var queryLastNames = from
student in students
group student by student.LastName into newGroup orderby newGroup.Key select newGroup;
```

При выполнении запроса элементы каждой группы выдаются в одном экземпляре. Для каждой группы может также создоваться новый элемент на основе групповой операции.

Когда запрос завершается предложением **group**, результаты представ- ляются в виде списка из списков. Каждый элемент в списке является объек- том, имеющим член **Key** и список членов группы, сгруппированных по этому ключу. При итерации запроса, создающего последовательность групп, необходимо использовать вложенный цикл **foreach**. Внешний цикл выполняет итерацию каждой группы, а внутренний цикл — итерацию чле- нов каждой группы.

```
var queryCustomersByCity = from cust in customers group cust by cust.City; foreach
(var customerGroup in queryCustomersByCity)
{ Console.WriteLine(customerGroup.Key); foreach (Customer customer in customerGroup) {
Console.WriteLine(" {0}", customer.Name); }
}
```

10.1.2. Слияние нескольких входных последовательностей в одну выходную

Запрос **LINQ** можно использовать для создания выходной последова- тельности, содержащей элементы из нескольких входных последовательно- стей. В следующем примере показано объединение двух находящихся в памяти структур данных, но те же принципы могут применяться для соеди- нения данных из источников **XML**, **SQL** или **DataSet**. Предположим, что существуют два следующих типа классов:

```
class Student
{ public string FirstName { get; set; } public string LastName {get; set;} public int ID { get;
set; } } class Teacher
{ public string First { get; set; } public string Last { get; set; } public int ID { get; set; } public
string Book { get; set; }}
static List<Student> students = new List<Student>
{ new Student {FirstName="Светлана", LastName="Богатырева", ID=1}, new Student
{FirstName="Иван", LastName="Поддубный", ID=1}, new Student {FirstName="Марина",
LastName="Шаталина", ID=2}, new Student {FirstName="Петр", LastName="Бережной", ID=3}
};
static List< Teacher > teachers = new List< Teacher >
```

{ new Teacher{ First= "Олег" ,Last="Козлов" ,ID=1, Book="Физика" }, new Teacher{ First= "Павел",Last="Петров" ,ID=1, Book="Химия" }, new Teacher{ First= "Юра",Last="Иванов",ID=1, Book="Математика"}, new Teacher{ First= "Ира" ,Last="Сидорова" ,ID=1, Book="История" }

};

Тогда с помощью операции **Concat** можно объединить результаты двух запросов в один. При объединении следует учитывать, что оба подза- проса должны формировать одинаковый тип данных.

var StudentQuery7_1 = from st in students orderby st.LastName ascending where st.God= =1990 select st; var StudentQuery7_2 = from st in students orderby

st.LastName as- cending

where st.God= =1991 select st;

foreach (var a in StudentQuery7_1.Concat(StudentQuery7_2))

{ Console.WriteLine("\t{0}, {1}", a.LastName, a.FirstName); } Если источники данных имеют различную структуру, можно исполь- зовать анонимный тип для формирования выходной последовательности. Например:

var StudentQuery7 = from st in students orderby st.LastName ascending select new { st.FirstName , st.LastName}; var StudentQuery8 = from th in teachers orderby th.Last ascending

select new {

FirstName=th.First, LastName=th.Last }; foreach (var a in

StudentQuery7.Concat(StudentQuery8)) {

Console.WriteLine("\t{0}, {1}", a.LastName, a.FirstName); }

10.1.3. Выбор подмножества каждого исходного элемента

Существует два основных способа выбора подмножества каждого элемента в исходной последовательности: выбор одного поля или нескольких полей.

Чтобы выбрать только один член исходного элемента, используется операция «.» (точка).

Например, если объект **Teachers** содержит несколько свойств, то выбрать в качестве результата только свойство **Book** можно, указав в части **select** требуемое свойство.

var query = from cust in teachers select cust.Book;

Для создания элементов, содержащих более одного свойства исходного элемента, требуется использовать инициализатор объектов с именованным объектом либо с анонимным типом. Например:

var query = from cust in teachers

select new { Teach = cust.Name, Lesson = cust.Book};

10.1.4. Выполнение операций над исходными элементами

Выходная последовательность может быть результатом вычислений с использованием исходных элементов в качестве входных аргументов. Например: double[] rad = { 1, 2, 3 };

var query_grad = from r in rad select String.Format("Grad = {0}", (r * 180/ * 3.14)); foreach (string s in query) Console.WriteLine(s);

10.1.5. Создание вложенных запросов

В следующем примере показано, как создавать вложенные группы в выражении запроса **LINQ**. Каждая группа, созданная в соответствии с го- дом, затем подразделяется на группы на основе имен учащихся:

var queryNestedGroups =

from student in students group student by student.God into newGroup1 from newGroup2 in

```
(from student in newGroup1 group student by student.LastName) group
    newGroup2 by newGroup1.Key; foreach (var outerGroup in
    queryNestedGroups) { Console.WriteLine("DataClass.Student God = {0}",
    outerGroup.Key);
    foreach (var innerGroup in outerGroup)
    { Console.WriteLine("\tNames that begin with: {0}",
    innerGroup.Key); foreach (var innerGroupElement in innerGroup)
    { Console.WriteLine("\t\t{0} {1}", innerGroupElement.LastName,
innerGroupElement.FirstName);
    } } }
```

10.1.6. Соединение последовательностей

Простое внутреннее соединение, которое сопоставляет элементы из двух источников данных на основании простого ключа, осуществляется с помощью оператора **join.** Например, для слияния последовательности stu- dents и teachers по полю ID, можно использовать запрос:

```
var query = from person in students join
th_person in teachers on person.ID equals th_person.ID select new{ PName=
person.FirstName, TName = th_person.First };
```

10.2. Задание для выполнения лабораторной работы

Разработать консольное приложение для создания последовательностей на основе других последовательностей.

Варианты заданий:

1. Разработать последовательности для хранения данных: кинофильм – название, год выпуска, жанр, режиссёр; спектакль – название, автор, жанр, учётный номер актёра главной роли; мер. актеры –

фамилия, имя, отчество, год рождения, амплуа, учетный но-

б) сгруппировать данные в последовательности «актёры» по амплуа и по году рождения;

в) слить последовательности «кинофильм» и «спектакль» и выбрать в новую последовательность: название, жанр, режиссер/автор;

г) соединить последовательности «спектакль» и «актёры» и выдать название спектакля, фамилию автора, фамилию актёра в главной роли.

2. Разработать последовательности для хранения данных:

товар – название, стоимость, количеством (в штуках), производитель (шифр); склад – название, общая площадь, количество автопогрузчиков, год постройки;
производитель – название фирмы, руководитель, шифр.

Построить запросы:

а) выдать данные из последовательности «товар» в отсортированном по стоимости виде;

б) сгруппировать данные в последовательности «склад» по количеству автопогрузчиков и по году постройки;

в) сформировать запрос для подсчета общего числа погрузчиков на складах;

г) соединить последовательности «товар» и «производитель» по полю «шифр» и выдать данные: наименование товара, фирма производитель.

3. Разработать последовательности для хранения данных:

река – название, длина, максимальная ширина, число притоков, шифр реки; море – название, площадь, шифр моря;

корабль – название, число пассажиров, пароходство, год постройки, тип (речной, морской), шифр реки/моря.

Построить запросы:

а) выдать данные из последовательности «река» в отсортированном по длине виде;

б) сгруппировать данные в последовательности «корабль» по пароход- ству и по году постройки;

в) сформировать запрос для подсчета числа кораблей в пароходстве;

г) соединить последовательности «корабль» и «река» по полю «шифр» и выдать данные: наименование корабля, название реки.

4. Разработать последовательности для хранения данных:

самолет – тип самолета, грузоподъемность, максимальная дальность, размах крыльев, длина разбега, шифр компании; вертолет – тип вертолета, грузоподъемность, максимальная высота, дальность полета, шифр компании; авиакомпания – название, место размещения офиса, дата образования фирмы, шифр компании.

Построить запросы:

а) выдать данные из последовательности «самолет» в отсортированном по грузоподъёмности виде;

б) сгруппировать данные в последовательности «вертолет» по максимальной высоте подъема и дальности полета;

в) сформировать запрос для подсчета числа самолетов в авиакомпании;

г) соединить последовательности «самолет» и «авиакомпания» по полю «шифр» и выдать данные: наименование самолета, название авиакомпании.

5. Разработать последовательности для хранения данных:

жилой дом – тип проекта, число этажей, число подъездов, дата постройки, шифр; район города – название, адрес районной администрации, количество жителей, площадь, шифр; список домов – шифр района, шифр дома. Построить запросы:

а) выдать данные из последовательности «жилой дом» в отсортированном по количеству этажей виде;

б) сгруппировать данные в последовательности «жилой дом» по типу проекта подъема и количеству этажей;

в) сформировать запрос для подсчета числа домов в районе;

г) соединить последовательности «жилой дом», «район города» и «список домов» по полю «шифр» и выдать данные: наименование района, тип дома, дата постройки.

6. Разработать последовательности для хранения данных:

грузовой автомобиль – марка автомобиля, грузоподъемность, дата выпуска, дата капитального ремонта, государственный номер, шифр автопар- ка; такси – марка автомобиля, количество посадочных мест, дата выпуска, государственный номер, шифр автопарка; автопарк – название, адрес размещения, площадь для размещения автомобилей, шифр. Построить запросы:

а) выдать данные из последовательности «грузовой автомобиль» в отсортированном по дате выпуска виде;

б) сгруппировать данные в последовательности «такси» по марке автомобиля и по дате выпуска;

в) сформировать запрос для подсчета количества такси в автопарке;

г) соединить последовательности «грузовой автомобиль» и «автопарк» по полю «шифр» и выдать данные: наименование автопарка, тип автомобиля, государственный номер автомобиля.

Контрольные вопросы

1. Какие изменения можно выполнять над исходными последовательностями для получения новых последовательностей?
2. Изменяет ли группировка исходную последовательность или влияет только на выходную последовательность?
3. Как задается имя группы?
4. Где может быть использовано имя группы?
5. Как осуществить перебор групп и элементов в группе для выходной последовательности?
6. Какой метод позволяет добавить в одну последовательность другую? 7. Каким требованиям должны удовлетворять последовательности для их слияния в одну последовательность?
8. С помощью кокой операции осуществляется выборка элементов объекта последовательности для формирования подмножества полей?
9. Что понимают под вложенными запросами?
10. Как формируется запрос на соединения двух последовательностей?

ты ее выполнения.