МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
**«МОСКОВСКИЙ АВТОМОБИЛЬНО-ДОРОЖНЫЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ (МАДИ)»**
ВОЛЖСКИЙ ФИЛИАЛ

Кафедра гуманитарных и естественнонаучных дисциплин

# Методические указания к лабораторным работам по дисциплине ПРОГРАММИРОВАНИЕ

## ЧАСТЬ II

Направление подготовки

***09.03.01 Информатика и вычислительная техника***

Направленность (профиль, специализация) образовательной программы

***«Автоматизированные системы обработки информации и управления»***

Квалификация

бакалавр

Чебоксары
2019

# СОДЕРЖАНИЕ

## ТРЕБОВАНИЯ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

Все лабораторные работы курса “Интернет программирование” выполняются в едином порядке, в соответствии с едиными требованиями.

**Порядок выполнения работы.**

1. Изучить "Краткие теоретические сведения".
2. Получить у преподавателя условия задач (номер варианта).
3. Решить задачи (см. п.
4. **Порядок решения задачи.**).
5. Показать работу программ преподавателю.
6. Распечатать тексты полученных программ.
7. Оформить отчет (пояснительную записку) (см. п.
8. **Требования к отчету**).
9. Сдать отчет (пояснительную записку) преподавателю.
10. Подготовиться к ответам на контрольные вопросы.
11. Защитить работу.

**Порядок решения задачи.**

1. Изучить условие задачи.
2. Разработать алгоритм решения задачи.
3. Согласовать алгоритм решения задачи с ведущим преподавателем
4. Разработать порядок работы с программой.
5. Написать и ввести программу.
6. Отладить программу.
7. Скорректировать алгоритм и порядок работы программы по результатам отладки.
8. Ответить на вопросы задания (если есть).

**Общие требования к программам.**

- Программа должна выводить на терминал реквизиты авторов (фамилию, имя и группу).
- Программа, использующая ввод с клавиатуры, должна подсказывать пользователю, что ему делать.

**Требования к отчету.**

*Требования к оформлению.*

1. Отчет выполняется на листах формата А4 с использованием любого текстового процессора и распечатывается на принтере.
2. Титульный лист отчета выполняется по стандартной форме (см. Рисунок 1).
3. Рамка на последующих листах отчета необязательна.

4. Листы отчета необходимо скрепить.
5. Отчет должен быть подписан исполнителем.
6. Тексты программ должны содержать комментарии к использованию переменных и работе программы.
7. Тексты программ распечатываются на принтере.

*Содержание отчета.*

1. Титульный лист (см. Рисунок 1).
2. Цель работы.
3. Описание решений задач (см. п. 0).
4. Тексты программ (распечатки).

*Порядок описания решения задачи.*

Решение каждой задачи описывается в следующем порядке;

1. Условие задачи.
2. Физическое и математическое решение задачи.

Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«МОСКОВСКИЙ АВТОМОБИЛЬНО-ДОРОЖНЫЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ (МАДИ)»
Волжский филиал

Факультет очный
Кафедра математики и информатики
Направление подготовки Автоматизированные системы обработки информации и управления

О Т Ч Е Т

к лабораторной работе №1

**«Доступ к базам данных. СУБД MySQL»**

Дата сдачи «__»_____20__г Выполнил: студент гр. ОП-14, Петров И.В.
Зачтено «__»_____20__г Преподаватель: Семенов Б.И.

Чебоксары 2015

Рисунок 1. Пример оформления титульного листа.

## ТЕМАТИКА ЛАБОРАТОРНЫХ РАБОТ

**Лабораторная работа №1. Выполнение программы простой структуры. Вычисление выражений с использованием стандартных функций.**

**Цель работы**

Освоить создание программ простой структуры в функциональном языке высокого уровня. Подготовить общие черты программ простой структуры на языке программирования Visual C++ в функциональной парадигме и ЯВУ java в объектно-ориентированной парадигме.

**Основные сведения**

**1. Основы работы с Visual Studio C++**

Основные способы создания программы на языке высокого уровня C++.

**Общие сведения:**

Язык C (читается как Си) в основе своей был создан в 1972 г. как язык для операционной системы UNIX [1.2]. Автором этого языка считается Денис М. Ритчи (DENNIS M. RITCHIE).

Отличительной особенностью среды Microsoft Visual Studio 2010 является то, что она поддерживает работу с несколькими языками программирования и программными платформами. Поэтому, перед тем, как начать создание программы на языке C, необходимо выполнить несколько подготовительных шагов по созданию проекта и выбора и настройки компилятора языка C для трансляции исходного кода

После запуска Microsoft Visual Studio 2010 появляется следующая стартовая страница, которая показана на рис. 1.1.

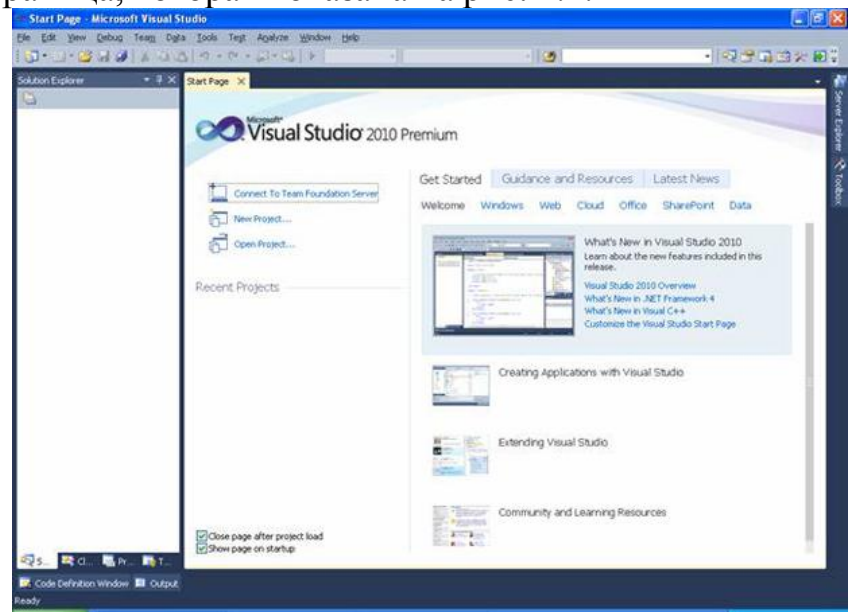


Рис. 1.1. Стартовая страница Visual Studio 2010

Следующим шагом является создание нового проекта. Для этого в меню File необходимо выбрать New – Project (или комбинацию клавишCtrl +

Shift + N ). Результат выбора пунктов меню для создания нового проекта показан на рис. 1.2.

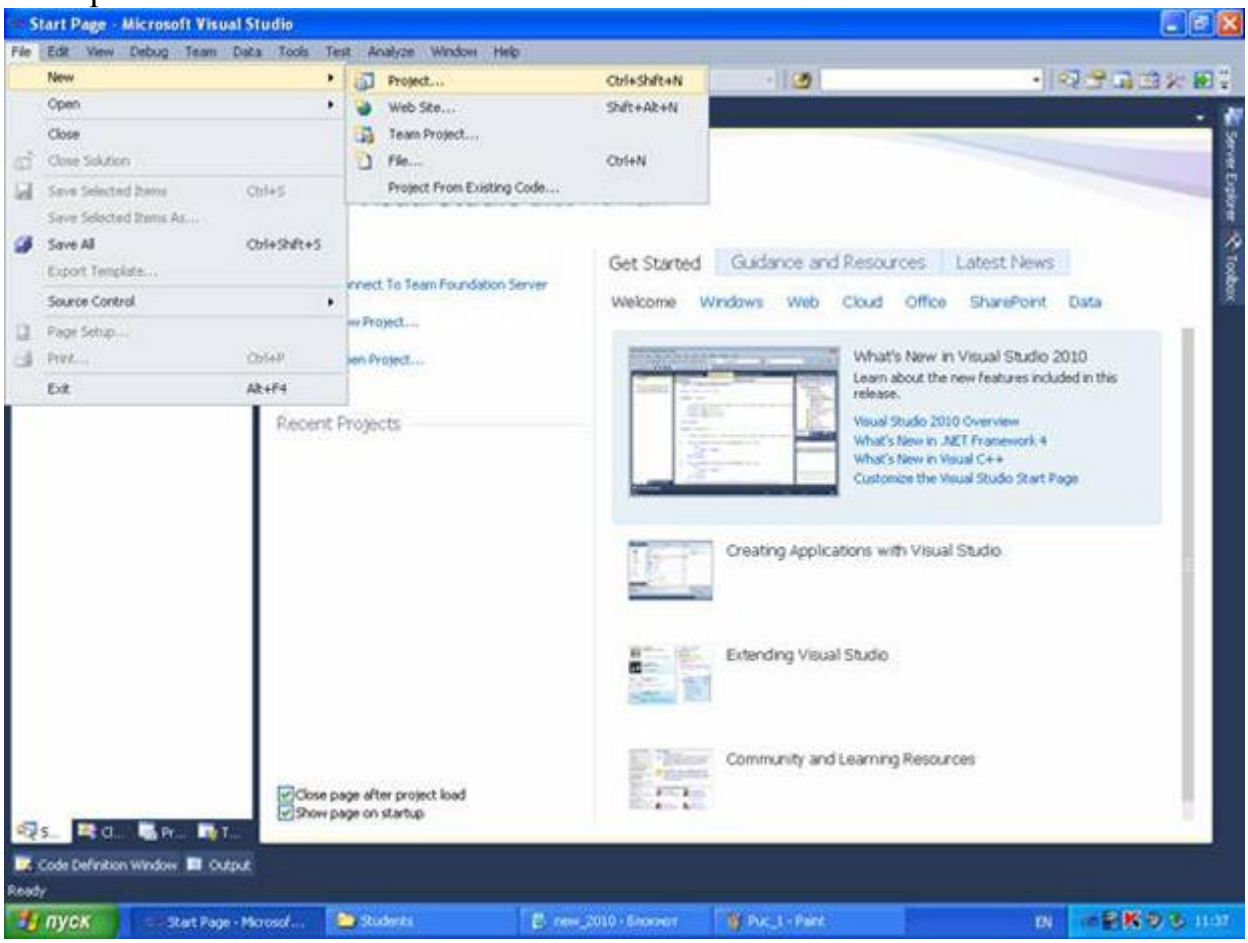


Рис. 1.2. Окно с выбором нового проекта

Среда Visual Studio отобразит окно New Project, в котором необходимо выбрать тип создаваемого проекта. Проект (project) используется вVisual Studio для логической группировки нескольких файлов, содержащих исходный код, на одном из поддерживаемых языков программирования, а также любых вспомогательных файлов. Обычно после сборки проекта (которая включает компиляцию всех входящих в проект файлов исходного кода) создается один исполняемый модуль.

В окне New Project следует развернуть узел Visual C++, обратиться к пункту Win32 и на центральной панели выбрать Win32 Console Application. Выбор этой опции показан на рис. рис. 1.3.

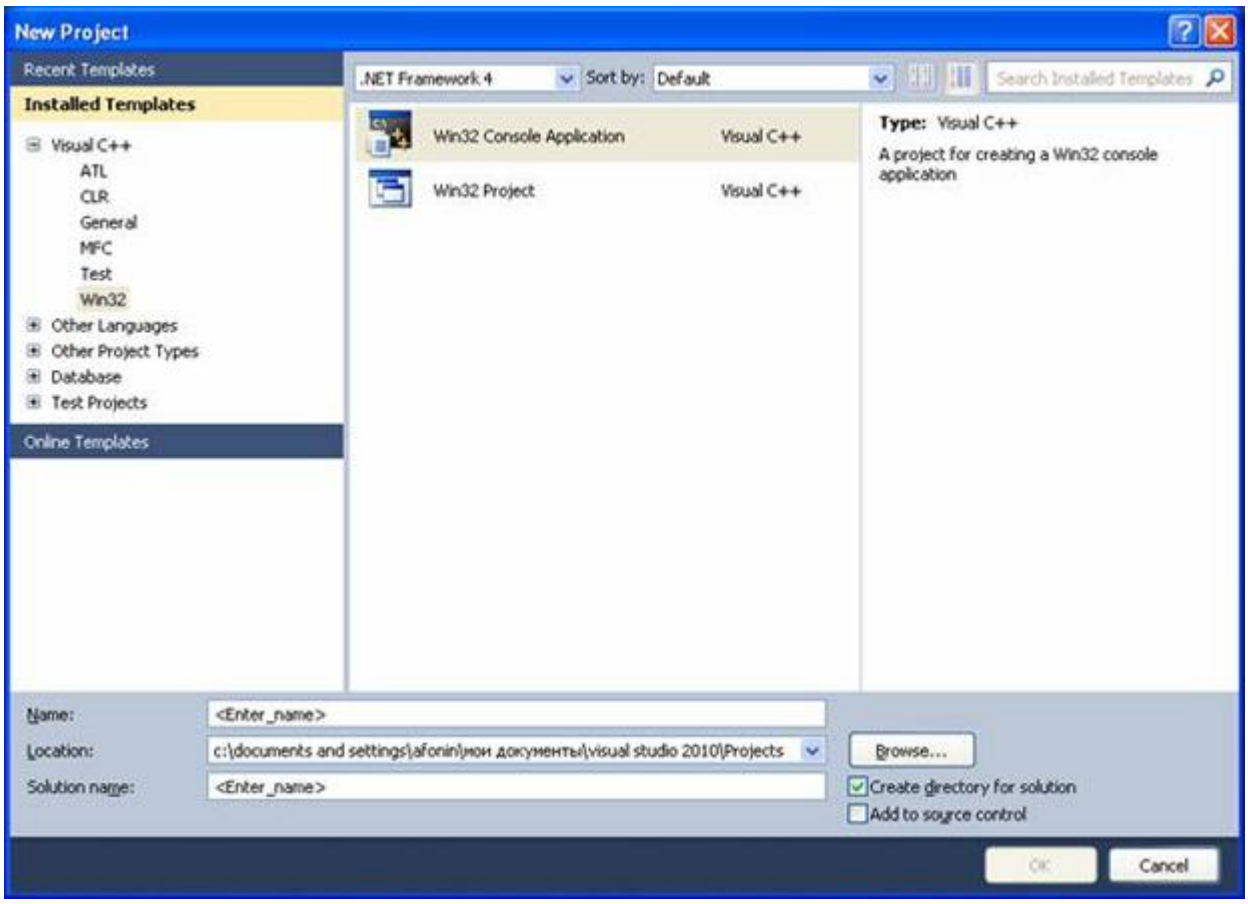


Рис. 1.3. Выбор типа проекта

Затем в поле редактора Name (где по умолчанию имеется <Enter_name>) следует ввести имя проекта, например, hello. В поле Locationможно указать путь размещения проекта, или выбрать путь размещения проекта с помощью клавиши (кнопки) Browse. По умолчанию проект сохраняется в специальной папке Projects.

Одновременно с созданием проекта Visual Studio создает решение. Решение (solution) – это способ объединения нескольких проектов для организации более удобной работы с ними.

После нажатия кнопки OK откроется окно Win32 Application Wizard (мастер создания приложений для операционных систем Windows).

Выбор имени проекта может быть достаточно произвольным: допустимо использовать числовое значение, допустимо имя задавать через буквы русского алфавита.

В дальнейшем будем использовать имя, набранное с помощью букв латинского алфавита и, может быть, с добавлением цифр.


Рис. 1.5. Мастер создания приложения

На первой странице представлена информация о создаваемом проекте, на второй можно сделать первичные настройки проекта.

В дополнительных опциях ( Additional options ) следует поставить галочку в поле Empty project (пустой проект) и снять (убрать) галочку в поле Precompiled header.

После нажатия кнопки Finish, получим экранную форму, показанную на рис. 1.8, где приведена последовательность действий добавления файла для создания исходного кода к проекту. Стандартный путь для этого: подвести курсор мыши к папке Source Files из узла hello в левой части открытого проекта приложения, выбрать Add и New Item (новый элемент).

После выбора (нажатия) New Item получим окно, показанное на рис. 1.9, где через пункт меню Code узла Visual C++ выполнено обращение к центральной части панели, в которой осуществляется выбор типа файлов. В данном случае требуется обратиться к закладкеC++ File (.cpp).

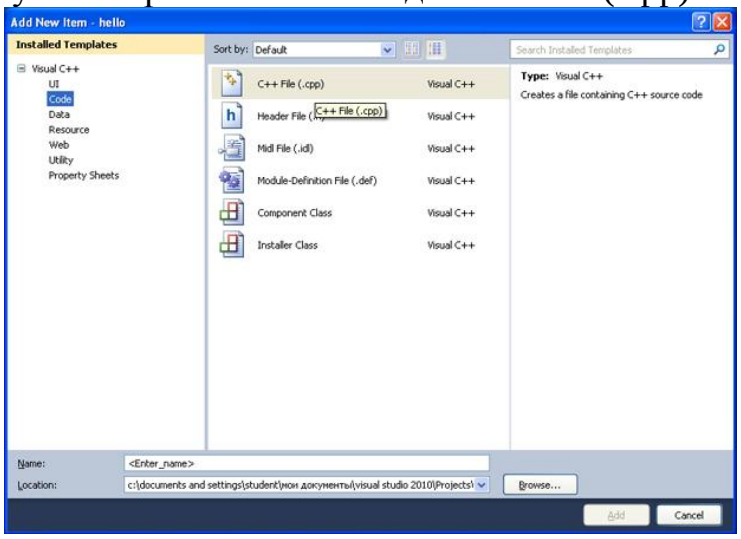


Рис. 1.9. Окно выбора типа файла для подключения к проект

Теперь в поле редактора Name (в нижней части окна) следует задать имя нового файла и указать расширение ".c". Например, main.c. Имя файла может быть достаточно произвольным, но имеется негласное соглашение, что имя файла должно отражать его назначение и логически описывать исходный код, который в нем содержится. В проекте, состоящем из нескольких файлов, имеет смысл выделить файл, содержащий главную функцию программы, с которой она начнет выполняться. В данном пособии такому файлу мы будем задавать имямain.c, где расширение .c указывает на то, что этот файл содержит исходный код на языке C, и он будет транслироваться соответствующим компилятором. Программам на языке C принято давать расширение .c. После задания имени файла в поле редакторaName, получим форму, показанную на рис. 1.10.

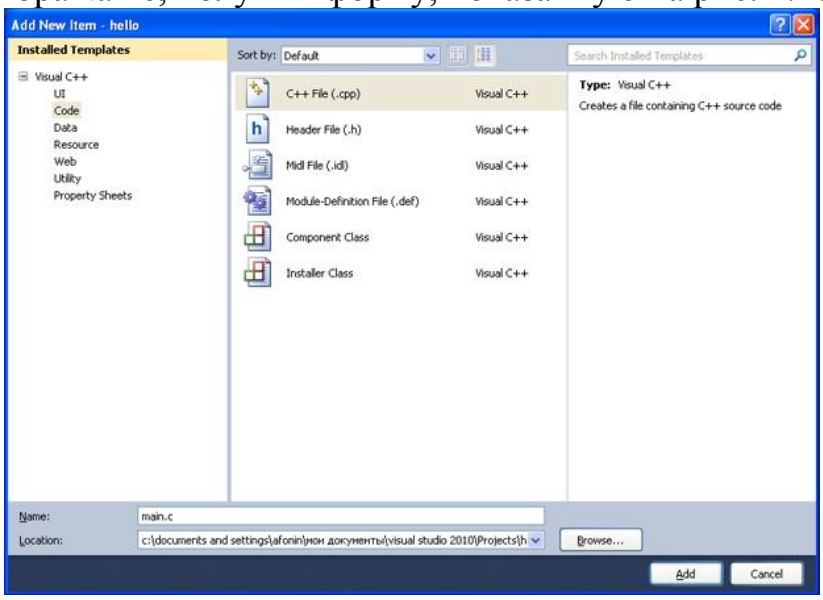


Рис. 1.10. Задание имени файла, подключаемому к проекту

Затем следует нажать кнопку Add. Вид среды Visual Studio после добавления первого файла к проекту показан на рис. 1.11. Добавленныйфайл отображается в дереве Solution Explorer под узлом Source Files (файлы с исходным кодом), и для него автоматически открывается редактор.

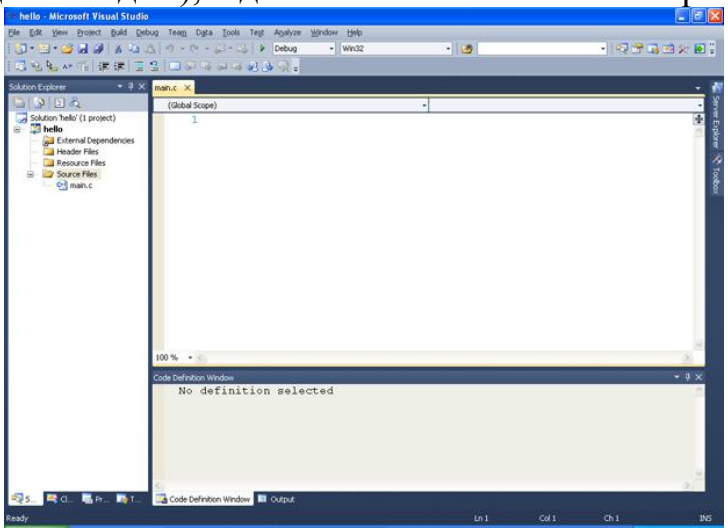


Рис. 1.11. Подключение файла проекта

На рис. 1.11 в левой панели в папке Solution Explorer отображаются файлы, включенные в проект в папках. Приведем описание.

Папка Source Files предназначена для файлов с исходным кодом. В этой папке отображаются файлы с расширением .c.

Папка Header Files содержит заголовочные файлы с расширением .h.

Папка Resource Files содержит файлы ресурсов, например изображения и т. д.

Папка External Dependencies отображает файлы, не добавленные явно в проект, но использующиеся в файлах исходного кода, например включенные при помощи директивы #include. Обычно в папке External Dependencies присутствуют заголовочные файлы стандартной библиотеки, использующиеся в проекте.

Следующий шаг состоит в настройке проекта. Для этого в меню Project главного меню следует выбрать hello Properties (или с помощью последовательного нажатия клавиш Alt+F7). Пример обращения к этому пункту меню показан на рис. 1.12.

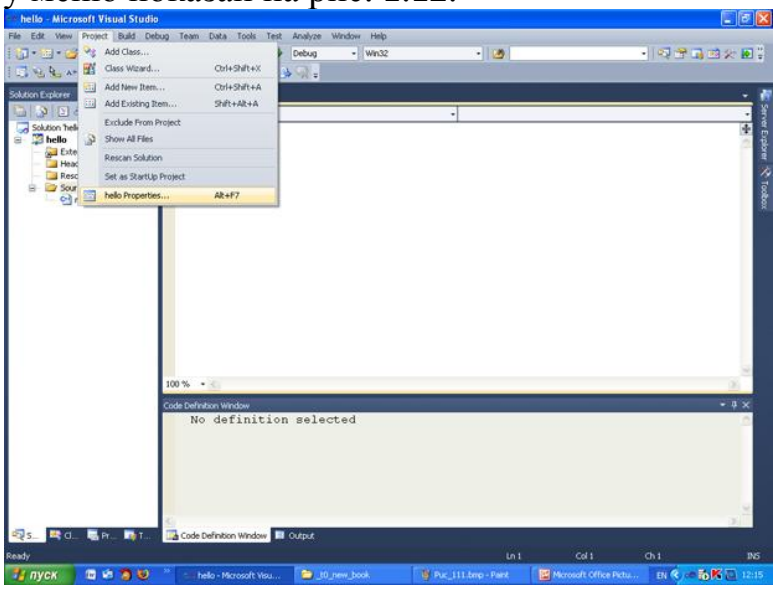


Рис. 1.12. Обращение к странице свойств проекта

После того как произойдет открытие окна свойств проекта, следует обратиться (с левой стороны) к Configuration Properties. Появится ниспадающий список, который показан на рис. 1.13. Выполнить обращение к узлу General, и через него в правой панели выбрать Character Set, где установить свойство Use Multi-byte Character Set. Настройка Character Set (набор символов) позволяет выбрать, какаякодировка символов – ANSI или UNICODE – будет использована при компиляции программы. Для совместимости со стандартом C89 мы выбираем Use Multi-Byte Character Set. Это позволяет использовать многие привычные функции, например, функции по выводу информации на консоль.

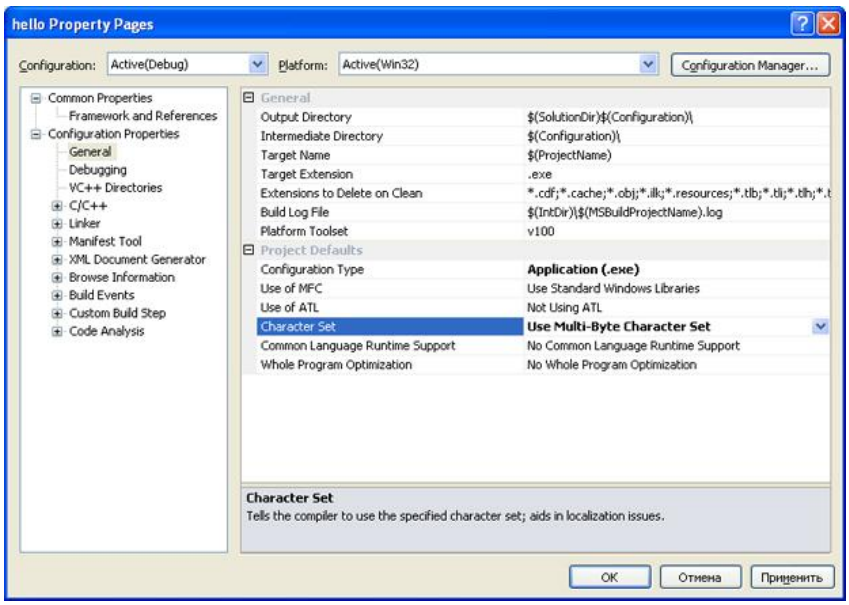


Рис. 1.13. Меню списка свойств проекта

В этом редакторе наберем программу, выводящую традиционное приветствие "Hello World". Для компиляции созданной программы можно обратиться в меню Build, или, например, набрать клавиши Ctr+F7. В случае успешной компиляции получим следующую экранную форму, показанную на рис. 1.17.

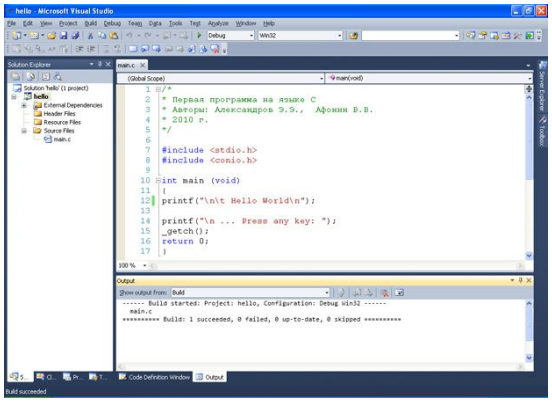


Рис. 1.17. Успешно откомпилированная первая программа на языке С

Для приведенного кода программы запуск на ее исполнение из окна редактора в Visual Studio 2010 можно нажать клавишу F5. рис. 1.18 показан результат исполнения первой программы.

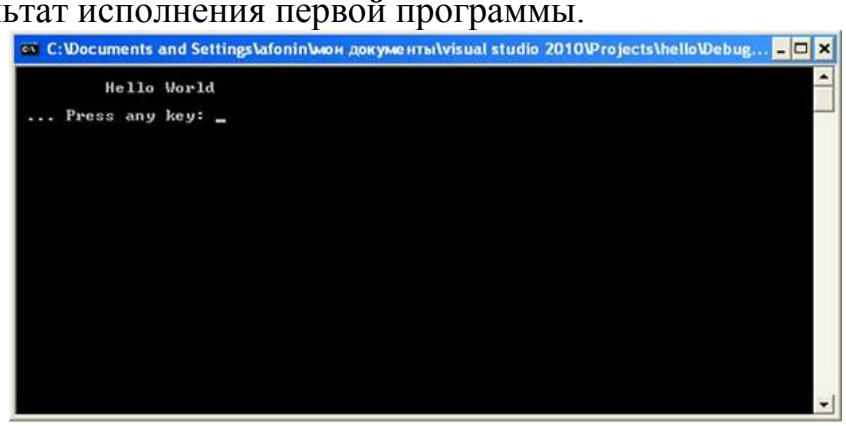


Рис. 1.18. Консольный вывод первой программы на языке

Произведем разбор первой программы. Во-первых, надо отметить, что в языке С нет стандартных инструкций (операторов) для вывода сообщений

на консоль (окно пользователя). В языке С предусматриваются специальные библиотечные файлы, в которых имеются функции для этих целей. В приведенной программе используется заголовочный файл с именем stdio.h (стандартный ввод–вывод), который должен быть включен в начало программы. Для вывода сообщения на консоль используется функция printf(). Для работы с консолью включен также заголовочный файл conio.h., который поддерживает функцию _getch(), которая извлекает символ из потока ввода, т. е. она предназначенная для приема сообщения о нажатии какой-либо (почти любой) клавиши на клавиатуре. С другими компиляторами, возможно, потребуется getch(), т.е. без префиксного нижнего подчеркивания. Строка программы

int main (void)

сообщает системе, что именем программы является main() – главная функция, и что она возвращает целое число, о чем указывает аббревиатура "int". Имя main() – это специальное имя, которое указывает, где программа должна начать выполнение [1.1]. Наличие круглых скобок после слова main() свидетельствует о том, что это имя функции. Если содержимое круглых скобок отсутствует или в них содержится служебное слово void, то это означает, что в функцию main() не передается никаких аргументов. Тело функции main()ограничено парой фигурных скобок. Все утверждения программы, заключенные в фигурные скобки, будут относиться к функции main().

В теле функции main() имеются еще три функции. Во-первых, функции printf() находятся в библиотеке компилятора языка С, и они печатают или отображают те аргументы, которые были подставлены вместо параметров. Символ "\n" составляет единый символ newline(новая строка), т.е. с помощью этого символа осуществляется перевод на новую строку. Символ "\t" осуществляет табуляцию, т.е. начало вывода результатов программы с отступом вправо.

Функция без параметров _getch() извлекает символ из потока ввода (т.е. ожидает нажатия почти любой клавиши). С другими компиляторами, возможно, потребуется getch(), т.е. без префиксного нижнего подчеркивания.

Последнее утверждение в первой программе

return 0;

указывает на то, что выполнение функции main() закончено и что в систему возвращается значение 0 (целое число). Нуль используется в соответствии с соглашением об индикации успешного завершения программы [1.3].

В завершение следует отметить, что все действия в программе завершаются символом точки с запятой.

Все файлы проекта сохраняются в той папке, которая сформировалась после указания в поле Location имени проекта (hello). На рис. 1.19 показаны папки и файлы проекта Visual Studio 2010..

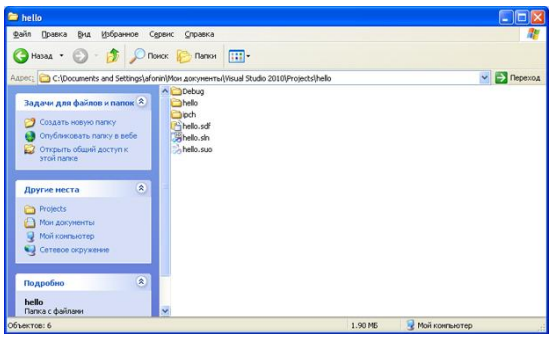


Рис. 1.19. Файлы и папки созданного проекта

На рис. 1.19 файлы с полученными расширениями означают:

hello.sln – файл решения для созданной программы. Он содержит информацию о том, какие проекты входят в данное решение. Обычно, эти проекты расположены в отдельных подкаталогах. Например, наш проект находится в подкаталоге hello;

hello.suo – файл настроек среды Visual Studio при работе с решением, включает информацию об открытых окнах, их расположении и прочих пользовательских параметрах.

hello.sdf – файл содержащий вспомогательную информацию о проекте, который используется инструментами анализа кода Visual Studio, такими как IntelliSense для отображения подсказок об именах и т.д.

Файлы папки Debug показаны на рис. 1.20.

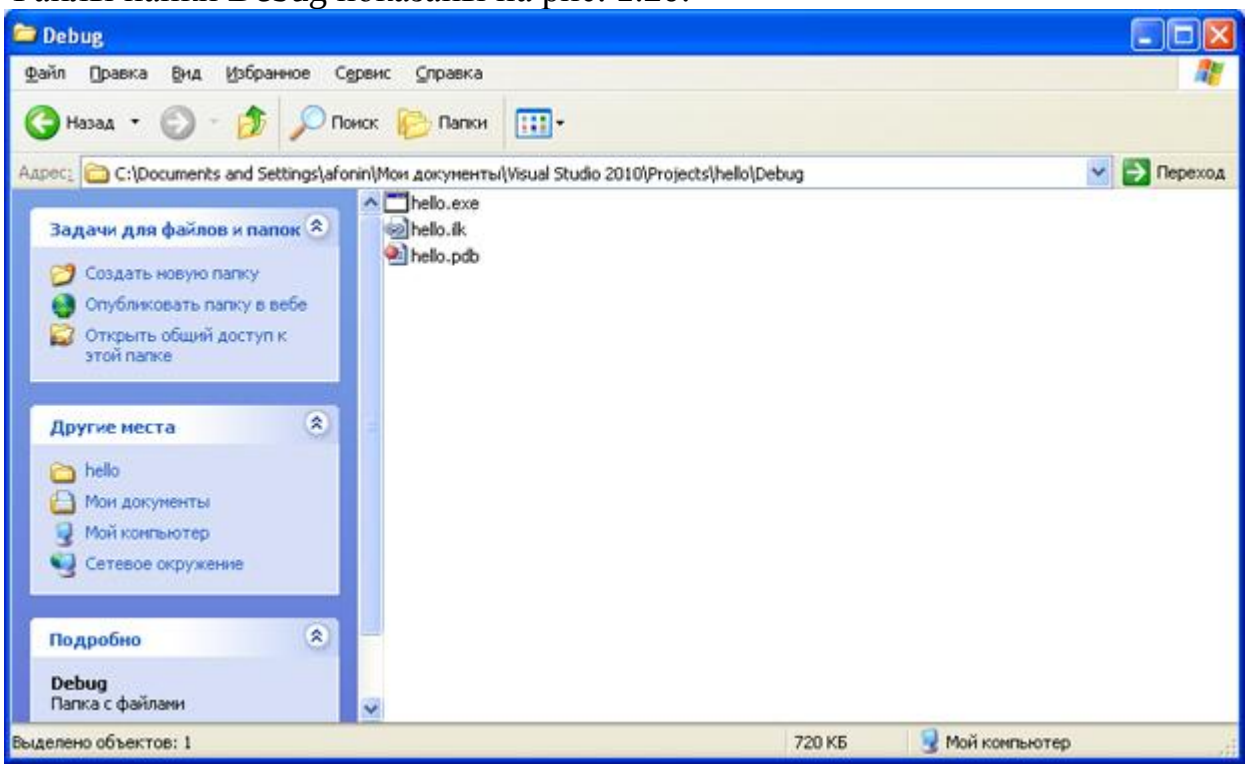


Рис. 1.20. Файлы папки Debug

Рассмотрим файлы в соответствии с рис. 1.20.

hello.exe – исполняемый файл проекта;

hello.ilk – файл "incremental linker", используемый компоновщиком для ускорения процесса компоновки;

hello.pdb – отладочная информация/информация об именах в исполняемых файлах, используемая отладчиком.

Файлы папки hello показаны на рис. 1.21.

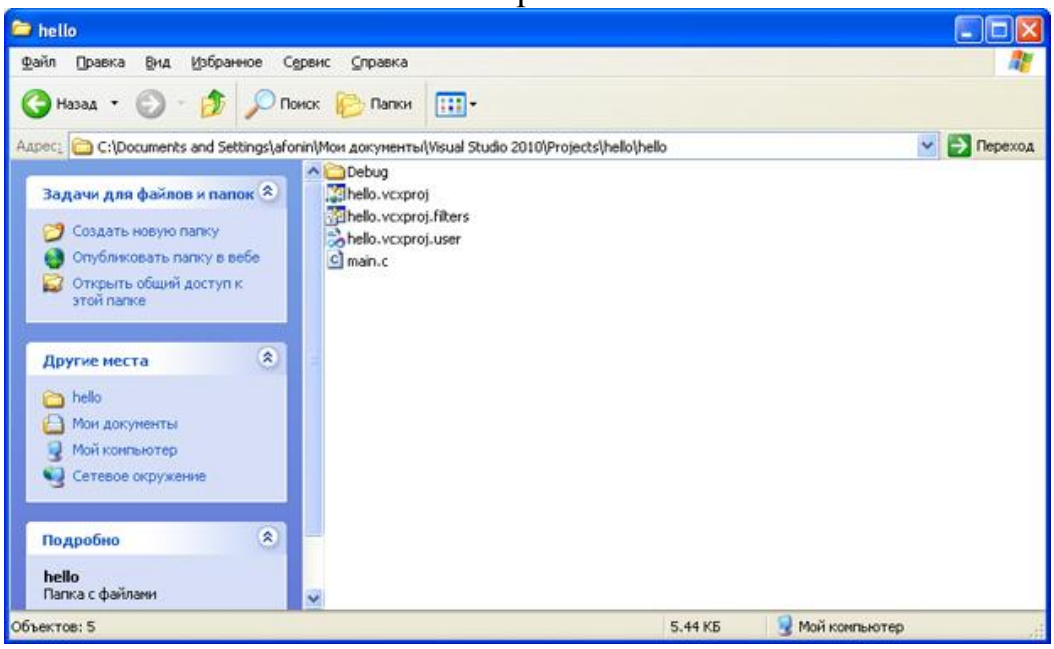


Рис. 1.21. Содержимое папки hello

Характеристика содержимого папки hello:

main.c – файл исходного программного кода,

hello.vcxproj – файл проекта,

hello.vcxproj.user – файл пользовательских настроек, связанных с проектом,

hello.vcxproj.filters – файл с описанием фильтров, используемых Visual Studio Solution Explorer для организации и отображения файлов с исходным кодом.

Практическая часть

В практической части выполните следующие задания на основе рассмотренной программы hello:

Напишите программу, которая выводила бы на консоль название факультета, где учитесь, номер группы, свою фамилию, имя и отчество в разных строках дисплея (консоли) с помощью одной функции printf().

Вывод выполните с помощью нескольких функций printf() (количество функций должно соответствовать каждой порции информации).

Для задания пункта 2 вывод информации выполните в различных строках подряд, т.е. без межстрочного пропуска.

Проверьте программу без ключевого слова void для функции main().

Примечание. Вывод требуемой информации осуществляется с помощью букв латинского алфавита. Комментарии в программе могут быть сделаны после символа "//" или внутри комбинации символов "/* */".

**Список индивидуальных заданий**

1. Напишите программу «Hello, world!» и преобразуйте ее в программу «Привет, мир!».
2. Напишите программу для перевода температуры в градусах по Фаренгейту в градусы по Цель- сию по формуле C = 5/9 (F – 32).
3. Напишите программу для вычисления площади треугольника по трем сторонам.
4. Заданы моменты начала и конца некоторого промежутка времени в часах, минутах и секундах (в пределах одних суток). Найти продолжительность этого промежутка в тех же единицах.

**Лабораторная работа №2. Использование основных операторов языка C++.**

**Цель работы:** выработать практические навыки работы с VBScript изучить встроенные и внешние объекты. научиться создавать, вводить в компьютер, выполнять и исправлять простейшие программы на языке VBScript в режиме диалога, познакомиться с диагностическими сообщениями компилятора об ошибках при выполнении программ, реализующих линейные алгоритмы

**Общие сведения:**

Оператор – выражение Любое выражение, завершающееся точкой с запятой, рассматривается как оператор, выполнение которого заключается в вычислении выражения. Частным случаем выражения является пустой оператор. i++; //инкремент x+=y+x;//аддитивное присваивание t=a>b;// присваивание результата отношения ; //пустой оператор

Составной оператор Составной оператор (блок) – это последовательность описаний и операторов, заключенная в фигурные скобки. Блок воспринимается компилятором как один оператор. { n++; summa+=n; }

Операторы выбора Операторы выбора – это условный оператор и переключатель.

1. Условный оператор имеет полную и сокращенную форму. if (выражение-условие) оператор; //сокращенная форма В качестве выражения-условия могут использоваться арифметическое выражение, отношение и логическое выражение. Если значение выражения-условия отлично от нуля (т. е. истинно), то выполняется оператор.

if (x<y&&x=0)

{

x1=(-b-sqrt(d))/(2*a);

x2=(-b+sqrt(d))/(2*a); Console.WriteLine(”x1={0},x2={1}”,x1,x2); } else cout<<“\n

Решения нет”;

2. Переключатель определяет множественный выбор.

switch (выражение) {

case константа1 : оператор1 ;

case константа2 : оператор2 ; . . . . . .

. [default: операторы;]

}

При выполнении оператора switch, вычисляется выражение, записанное после switch, оно должно быть целочисленным. Полученное значение последовательно сравнивается с константами, которые записаны следом за case. При первом же совпадении выполняются операторы, помеченные данной меткой. Если выполненные операторы не содержат оператора

перехода, то далее выполняются операторы всех следующих вариантов, пока не появится оператор перехода или не закончится переключатель.

Если значение выражения, записанного после switch, не совпало ни с одной константой, то выполняются операторы, которые следуют за меткой default. Метка default может отсутствовать.

```
int i;
string buf;
Console.Write(”Введите число:”);
buf=Console.ReadLine();
i=int.Parse(buf);
switch(i) {
case 1: Console.WriteLine("\nthe number is one");
break;
case 2: Console.WriteLine ("\n2*2="+i*i);
break;
case 3: Console.WriteLine("\n3*3="+i*i);
break;
case 4: Console.WriteLine ("\n"+i+" is very beautiful!");
 break;
 default: WriteLine ("\nThe end of work");
}
```

Результаты работы программы: При вводе 1 будет выведено: The number is one При вводе 2 будет выведено: 2*2=4 При вводе 3 будет выведено: 3*3=9 При вводе 4 будет выведено: 4 is very beautiful! При вводе всех остальных чисел будет выведено: The end of work

Операторы циклов Цикл с предусловием: while (выражение-условие) оператор; В качестве <выражения-условия> чаще всего используется отношение или логическое выражение. Если оно истинно, т. е. не равно 0, то тело цикла выполняется до тех пор, пока выражение-условие не станет ложным.

```
Console.Write("?  ");
buf = Console.ReadLine();
a = int.Parse(buf);
while (a != 0) { if (a % 2 == 0) s += a; Console.Write("? ");
 buf = Console.ReadLine(); a = int.Parse(buf);
}
```

Цикл с постусловием: do оператор while (выражение-условие); Тело цикла выполняется до тех пор, пока выражение-условие истинно.

```
 do
{
Console.Write("? ");
buf = Console.ReadLine();
 a = int.Parse(buf);
 if (a % 2 == 0 && a != 0) s += a; }
while (a != 0);
```

Цикл с параметром:

for (выражение_1;выражение-условие;выражение_3) оператор; выражение_1 и выражение_3 могут состоять из нескольких выражений, разделенных запятыми. Выражение_1 – задает начальные условия для цикла (инициализация). Выражение-условие определяет условие выполнения цикла, если оно не равно 0, цикл выполняется, а затем вычисляется значение выражения_3. Выражение_3 – задает изменение параметра цикла или других переменных (коррекция).

Цикл продолжается до тех пор, пока выражение-условие не станет равно 0. Любое выражение может отсутствовать, но разделяющие их « ; » должны быть обязательно.

```
1. for (int i = 0; i < n; i++)
{
 Console.WriteLine("?");
 buf = Console.ReadLine();
 a = int.Parse(buf);
if (a % 2 == 0) s += a;
}
2. // Изменение шага корректировки
 for ( n=2; n>60; n+=13) { оператор; }
3.//проверка условия отличного от того, которое налагается на //число итераций
 for ( num=1;num*num*num<216; num++) { оператор; }
4. //коррекция с помощью умножения
for ( d=100.0; d<150.0;d*=1.1) { оператор; }
5.//коррекция с помощью арифметического выражения
for (x=1;y<=75;y=5*(x++)+10) { оператор; }
 6. //использование нескольких корректирующих выражений, тело //цикла отсутствует
 for (x=1, y=0; x<10;x++,y+=x);
```

Часто переменная, которая управляет циклом for, необходима только для этого цикла и больше никак не используется. В этом случае можно объявить ее в разделе инициализации цикла.

Например, следующая программа вычисляет как сумму, так и факториала чисел от 1 до 5. Управляющая переменная i здесь объявляется в цикле

```
for. public static void Main()
{ int sum = 0; int fact = 1; // Вычисляем сумму и факториал чисел от 1 до 5.
for(int i = 1; i <= 5; i++)
{
 sum += i; // i известна только в пределах цикла. fact *= i;
} // Но здесь переменная i неизвестна. Console.WriteLine("Сумма равна " + sum);
Console.WriteLine("Факториал равен " + fact); }
```

Операторы перехода Операторы перехода выполняют безусловную передачу управления.

В С++ есть пять операторов, изменяющих естественный порядок выполнения вычислений:

оператор безусловного перехода goto;

оператор выхода из цикла break;

оператор перехода к следующей итерации цикла continue;

оператор возврата из функции return ;

оператор генерации исключения throw. break – оператор прерывания цикла. { оператор; if (<выражение_условие>) break; оператор; }

Т. е. оператор break целесообразно использовать, когда условие продолжения итераций надо проверять в середине цикла. // Найти сумму чисел, числа вводятся с клавиатуры до тех пор, пока не будет //введено 100 чисел или 0.

```
for(s=0, i=1; i<100;i++)
{
 Console.WriteLine("?");
buf = Console.ReadLine();
x = int.Parse(buf); // если ввели 0, то суммирование заканчивается
 if( x==0) break; s+=x;
 }  continue –
```

переход к следующей итерации цикла.

Он используется, когда тело цикла содержит ветвления. //Найти количество и сумму положительных чисел

```
for( k=0,s=0,x=1;x!=0;)
{ Console.WriteLine("?");
buf = Console.ReadLine();
x = int.Parse(buf);
 if (x<=0) continue;
k++;
s+=x; }
```

goto <метка> – передает управление оператору, который содержит метку.

В теле той же функции должна присутствовать конструкция: <метка>:оператор; Метка – это обычный идентификатор, областью видимости которого является функция.

Оператор goto передает управления оператору, стоящему после метки.

Использование оператора goto оправдано, если необходимо выполнить переход из нескольких вложенных циклов или переключателей вниз по тексту программы или перейти в одно место функции после выполнения различных действий.

Применение goto нарушает принципы структурного и модульного программирования, по которым все блоки, из которых состоит программа, должны иметь только один вход и только один выход. Нельзя передавать управление внутрь операторов if, switch и циклов. Нельзя переходить внутрь

блоков, содержащих инициализацию, на операторы, которые стоят после инициализации. return – оператор возврата из функции.

Он всегда завершает выполнение функции и передает управление в точку ее вызова.

Вид оператора: return [выражение];

**Контрольные вопросы:**

Решить указанные в варианте задачи, используя основные операторы языка C++. При решении задачи, использовать все типы циклов (for, while, do while).

2. Дана последовательность из n целых чисел. Найти среднее арифметическое этой последовательности.
3. Дана последовательность из n целых чисел. Найти сумму четных элементов этой последовательности.
4. Дана последовательность из n целых чисел. Найти сумму элементов с четными номерами из этой последовательности.
5. Дана последовательность из n целых чисел. Найти сумму нечетных элементов этой последовательности.
6. Дана последовательность из n целых чисел. Найти сумму элементов с нечетными номерами из этой последовательности.
7. Дана последовательность из n целых чисел. Найти минимальный элемент в этой последовательности.
8. Дана последовательность из n целых чисел. Найти номер максимального элемента в этой последовательности.
9. Дана последовательность из n целых чисел. Найти номер минимального элемента в этой последовательности.
10. Дана последовательность из n целых чисел. Найти максимальный элемент в этой последовательности.
11. Дана последовательность из n целых чисел. Найти сумму минимального и максимального элементов в этой последовательности.
12. Дана последовательность из n целых чисел. Найти разность минимального и максимального элементов в этой последовательности.
13. Дана последовательность из n целых чисел. Найти количество нечетных элементов этой последовательности.
14. Дана последовательность из n целых чисел. Найти количество четных элементов этой последовательности.
15. Дана последовательность из n целых чисел. Найти количество элементов этой последовательности, кратных числу K.
16. Дана последовательность из n целых чисел. Найти количество элементов этой последовательности, кратных ее первому элементу.
17. Дана последовательность из n целых чисел. Найти количество элементов этой последовательности, кратных числу K1 и не кратных числу K2.

18. Дана последовательность из n целых чисел. Определить, каких чисел в этой последовательности больше: положительных или отрицательных.

19. Дана последовательность целых чисел, за которой следует 0. Найти среднее арифметическое этой последовательности.

20. Дана последовательность целых чисел, за которой следует 0. Найти сумму четных элементов этой последовательности.

21. Дана последовательность целых чисел, за которой следует 0. Найти сумму элементов с четными номерами из этой последовательности.

22. Дана последовательность целых чисел, за которой следует 0. Найти сумму нечетных элементов этой последовательности.

23. Дана последовательность целых чисел, за которой следует 0. Найти сумму элементов с нечетными номерами из этой последовательности.

24. Дана последовательность целых чисел, за которой следует 0. Найти минимальный элемент в этой последовательности.

25. Дана последовательность целых чисел, за которой следует 0. Найти номер максимального элемента в этой последовательности.

26. Дана последовательность целых чисел, за которой следует 0. Найти номер минимального элемента в этой последовательности.

27. Дана последовательность целых чисел, за которой следует 0. Найти максимальный элемент в этой последовательности.

28. Дана последовательность целых чисел, за которой следует 0. Найти сумму минимального и максимального элементов в этой последовательности.

29. Дана последовательность целых чисел, за которой следует 0. Найти разность минимального и максимального элементов в этой последовательности.

30. Дана последовательность целых чисел, за которой следует 0. Найти количество нечетных элементов этой последовательности.

31. Дана последовательность целых чисел, за которой следует 0. Найти количество четных элементов этой последовательности.

32. Дана последовательность целых чисел, за которой следует 0. Найти количество элементов этой последовательности, кратных числу К.

33. Дана последовательность целых чисел, за которой следует 0. Найти количество элементов этой последовательности, кратных ее первому элементу.

34. Дана последовательность целых чисел, за которой следует 0. Найти количество элементов этой последовательности, кратных числу К1 и не кратных числу К2.

35. Дана последовательность целых чисел, за которой следует 0. Определить, каких чисел в этой последовательности больше: положительных или отрицательных. , всего n слагаемых; S=1+3+5+7+ …, всего n слагаемых; S=1+2-3+4+5-6+7+8-9+…, всего n слагаемых; S=15+17-19+21+23-25+…, всего n слагаемых;

**Лабораторная работа №3. Работа с одномерными массивами..**

**Цель работы:** научиться правильно использовать условный оператор if; научиться составлять программы решения задач на разветвляющиеся алгоритмы.

**Общие сведения.**

Стандартный вид объявления одномерного массива следующий:

тип имя_переменной [размер];

В С массивы должны определяться однозначно, чтобы компилятор мог выделить для них место в памяти. Здесь тип объявляет базовый тип массива и является типом каждого элемента массива. Параметр размер определяет, сколько элементов содержит массив. В одномерном массиве полный размер массива в байтах вычисляется следующим образом:

общее число байт = sizeof (базовый тип) *число элементов

У всех массивов первый элемент имеет индекс 0. Поэтому, если написать

char p [10];

то будет объявлен массив символов из 10 элементов, причем эти элементы адресуются индексом от 0 до 9. Следующая программа загружает целочисленный массив числами от 0 до 9 и выводит его:

```
#include <stdio.h>
int main(void)
{
int x[10]; /* резервирует место для 10 целочисленных элементов */
int t;
for(t=0; t<10; ++t) x[t] = t;
for(t=0; t<10; ++t) printf("%d ", x[t]);
return 0;
}
```

В С отсутствует проверка границ массивов. Можно выйти за один конец массива и записать значение в какую-либо переменную, не относящуюся к массиву, или даже в код программы. Работа по предоставлению проверки

границ возлагается на программиста. Например, следует убедиться, что массив символов, куда осуществляется ввод, имеет достаточную длину для принятия самой длинной последовательности символов.

Одномерные массивы  это на самом деле списки информации одного типа. Например, таблица. показывает, как массив а располагается в памяти, если он начинается с адреса 1000 и объявлен следующим образом:

char a [7];

| | | | | | | |
|---|---|---|---|---|---|---|
| a[0] | a[1] | a[2] | a[3] | a[4] | a[5] | a[6] |
| 1000 | 1001 | 1002 | 1003 | 1004 | 1005 | 1006 |

Таблица. Массив символов, включающий семь элементов и начинающийся по адресу 1000

**Лабораторная работа №5. Программирование с использованием процедур и функций..**

**Цель работы:**
изучение средств языка С++, используемых при наследовании классов.

**Общие сведения**

Все программные единицы в Си носят название функций и разницу в оформлении настоящих подпрограмм и настоящих функций можно обнаружить по типу возвращаемого значения. Если в качестве указания типа использовано служебное слово void, то перед нами типичная подпрограмма (в терминах Си — функция, не возвращающая значение). Перед заголовком объявляемой функции всегда присутствует либо стандартный тип, либо описатель типа с последующей звездочкой. Последнее означает, что функция возвращает указатель на данное соответствующего типа. В частности, допускается, что таковым указателем может быть "void *" В самом общем виде заголовок функции выглядит следующим образом:

void имя_функции([параметры])

ИЛИ

тип имя_функции([параметры])

ИЛИ

тип * имя_функции([параметры])

Тело функции, расположенное после заголовка, заключается в фигурные скобки. Если функция возвращает значение, то в ее теле должен присутствовать хотя бы один оператор return с указанием возвращаемого значения. Например:

```
int sign(int x)
{
/* Определение знака целого числа */
if(x<0) return -1;
if(x>0) return 1;
return 0; }
```

В отличие от Бейсика и Паскаля функции Си, не имеющие параметров, всегда сопровождаются пустыми круглыми скобками. Например — cirscr ().

Головная программа на Си представлена функцией main, которая может иметь до трех параметров, связанных с извлечением аргументов из командной строки. Чаще всего эта функция не имеет параметров и не возвращает значение:

void main(void)

К функции-подпрограмме в Си обращаются, указав ее имя со списком фактических параметров:

имя_функции(фактические параметры);

Функция, возвращающая значение, может использоваться как операнд в выражении соответствующего типа:

int qq;

qq=getch(); /*ожидание ввода кода символа с клавиатуры*/

Однако в большинстве примеров предыдущих глав вы могли заметить, что к функции getch обращаются и как к обычной подпрограмме, игнорируя возвращаемое значение. Это правило распространяется на любые функции Си. Конечно, не каждая из них в этом варианте может оказаться полезной. Например, допустимо, но нелепо встретить строку:

sin(0.5);

Системная функция в данном случае проработает, но эффекта от такого вызова никто не заметит. Разве что немного увеличится время работы программы. А в случае с функцией getch игнорирование результата позволяет зафиксировать момент, когда пользователь нажал какую-то клавишу.

**Лабораторная работа №6. Динамические массивы.**

**Цель работы:** познакомить с понятием *"множество"* в языке программирования Pascal; выработать навыки работы со структурой данных множество.

**Общие сведения**

При объявлении статического массива, его размером должна являться числовая константа, а не переменная. В большинстве случаев, целесообразно выделять определенное количество памяти для массива, значение которого изначально неизвестно.

Например, необходимо создать динамический массив из N элементов, где значение N задается пользователем. В предыдущем уроке мы учились выделять память для переменных, используя указатели. Выделение памяти для динамического массива имеет аналогичный принцип.

Создание динамического массива

```
#include <iostream>
using namespace std;
int main()
{
    int num; // размер массива
    cout << "Enter integer value: ";
    cin >> num; // получение от пользователя размера массива

    int *p_darr = new int[num]; // Выделение памяти для массива
    for (int i = 0; i < num; i++) {
        // Заполнение массива и вывод значений его элементов
        p_darr[i] = i;
        cout << "Value of " << i << " element is " << p_darr[i] << endl;
    }
    delete [] p_darr; // очистка памяти
    return 0;
```

}

Синтаксис выделения памяти для массива имеет вид указатель = new тип[размер]. В качестве размера массива может выступать любое целое положительное значение.

Обычно, объем памяти, необходимый для той или иной переменной, задается еще до процесса компиляции посредством объявления этой переменной. Если же возникает необходимость в создание переменной, размер которой неизвестен заранее, то используют динамическую память. Резервирование и освобождение памяти в программах на C++ может происходить в любой момент времени. Осуществляются операции распределения памяти двумя способами:

с помощью функции malloc, calloc, realloc и free;

посредством оператора new и delete.

Функция malloc резервирует непрерывный блок ячеек памяти для хранения указанного объекта и возвращает указатель на первую ячейку этого блока. Обращение к функции имеет вид:

void *malloc(size);

Здесь size — целое беззнаковое значение, определяющее размер выделяемого участка памяти в байтах. Если резервирование памяти прошло успешно, то функция возвращает переменную типа void *, которую можно привести к любому необходимому типу указателя.

Функция — calloc также предназначена для выделения памяти. Запись ниже означает, что будет выделено num элементов по size байт.

void *calloc (nime, size);

Эта функция возвращает указатель на выделенный участок или NULL при невозможности выделить память. Особенностью функции является обнуление всех выделенных элементов.

Функция realloc изменяет размер выделенной ранее памяти. Обращаются к ней так:

char *realloc (void *p, size);

Здесь p — указатель на область памяти, размер которой нужно изменить на size. Если в результате работы функции меняется адрес области памяти, то новый адрес вернется в качестве результата. Если фактическое значение первого параметра NULL, то функция realloc работает также, как и функция malloc, то есть выделяет участок памяти размером size байт.

Для освобождения выделенной памяти используется функция free. Обращаются к ней так:

void free (void *p size);

Здесь p — указатель на участок памяти, ранее выделенный функциями malloc, calloc или realloc.

Операторы new и delete аналогичны функциям malloc и free. New выделяет память, а его единственный аргумент — это выражение, определяющее количество байтов, которые будут зарезервированы. Возвращает оператор указатель на начало выделенного блока памяти. Оператор delete освобождает память, его аргумент — адрес первой ячейки блока, который необходимо освободить.

Динамический массив — массив переменной длины, память под который выделяется в процессе выполнения программы. Выделение памяти осуществляется функциями calloc, malloc или оператором new. Адрес первого элемента выделенного участка памяти хранится в переменной, объявленной как указатель. Например, следующий оператор означает, что описан указатель mas и ему присвоен адрес начала непрерывной области динамической памяти, выделенной с помощью оператора new:

int *mas=new int[10];

Выделено столько памяти, сколько необходимо для хранения 10 величин типа int.

Фактически, в переменной mas хранится адрес нулевого элемента динамического массива. Следовательно, адрес следующего, первого элемента, в выделенном участке памяти — mas+1, а mas+i является адресом i-го элемента. Обращение к i-му элементу динамического массива можно

выполнить, как обычно mas[i], или другим способом *(mas +i). Важно следить за тем, чтобы не выйти за границы выделенного участка памяти.

Когда динамический массив (в любой момент работы программы) перестает быть нужным, то память можно освободить с помощью функции free или оператора delete.

Предлагаю рассмотреть несколько задач, закрепляющих данный урок:

Задача 1

Найти сумму вещественных элементов динамического массива.

```
//Пример использования функции malloc и free
#include "stdafx.h"
#include <iostream>
using namespace std;
int main()
{
setlocale(LC_ALL,"Rus");
int i, n;
float *a; //указатель на float
float s;
cout<<"\n"; cin>>n; //ввод размерности массива
//выделение памяти под массив из n вещественных элементов
a=(float *)malloc(n*sizeof(float));
cout<<"Введите массив A \n";
//ввод элементов массива
for (i=0; i<n; i++)
{
cin>>*(a+i);
}
//накапливание суммы элементов массива
for (s=0, i=0; i<n; i++)
s+=*(a+i);
```

```
//вывод значения суммы
cout<<"S="<<s<<"\n";
//освобождение памяти
free(a);
system("pause");
return 0;
}
```

Задача 2

Изменить динамический массив целых чисел таким образом, чтобы его положительные элементы стали отрицательными и наоборот. Для решения задачи мы будем умножать каждый элемент на -1.

```
 //Пример использования операторов new и delete
#include "stdafx.h"
#include <iostream>
using namespace std;
int main()
{
setlocale(LC_ALL,"Rus");
int i, n;
//ввод количества элементов массива
cout<<"n="; cin>>n;
//выделение памяти
int *a=new int[n];
cout<<"Введите элементы массива:\n";
//ввод массива
for (i=0; i<n; i++)
cin>>a[i];
//вывод заданного массива
for (i=0; i<n; i++)
```

```
cout<<a[i]<<"\t"<<"\n";
//преобразование массива
for (i=0; i<n; i++)
a[i]=-a[i];
//вывод преобразованного массива
for (i=0; i<n; i++)
cout<<a[i]<<"\t"; cout<<"\n";
//освобождение памяти
delete []a;
system("pause");
return 0;
}
```

Контрольные вопросы

## Лабораторная работа №7. Массивы структур и массивы строк

**Цель работы:** познакомиться с понятиями "процедура" и "функция" в языке программирования Pascal, рассмотреть их сходства и различия, закрепить практические навыки работы с системой TURBO Pascal на примере реализации алгоритмов при помощи процедур и функций, научиться применять метод последовательной детализации в практическом программировании; применять процедуры и функции при решении задач.

**Общие сведения**

Структуры относятся к сложным типам данных. В предшествующих версиях MATLAB они именовались записями, что приводило к конфликтам в терминологии MATLAB и систем управления базами данных. Этот тип данных стал именоваться структурами после того, как широкое распространение получили средства MATLAB для работы с базами данных с использованием языка запросов Sequential Query Language (SQL).. Структуры MATLAB и их поля в отличие от полей записей баз данных не являются объектами SQL, но зато обращения к структурам могут быть откомпилированы и к ним возможен прямой доступ, минуя сложные и медленные механизмы систем управления базами данных. Структуры могут содержать разнородные данные, относящиеся к некоторому именованному объекту. Например, объект man (человек) может характеризоваться следующими данными:

| Поле | Значение | Комментарий |
|---|---|---|
| Man(i....) | | Имя записи |
| .name | Иван | Имя человека |
| .surname | Петров | Фамилия |
| .date | 1956 | Год рождения |
| .height | 170.5 | Рост |
| .weight | 70.34 | Вес |

Первые два столбца представляют схему структуры. Как нетрудно заметить, каждая i-я структура состоит из ряда полей, имеющих имена, например man(i).name, man(i) .date и т. д. Поля могут содержать данные любого типа — от пустого поля [ ] до массивов. Приведенная выше структура имеет размер 1x1. MATLAB поддерживает и массивы структур, что позволяет создавать мощные базы данных.

Поле структуры может содержать другую вложенную структуру или массив структур. Это позволяет создавать вложенные структуры и даже многомерные массивы структур. Поскольку в данной книге такие структуры не используются, отсылаем заинтересованного читателя к книге [42], где они описаны более подробно.

Создание структур и доступ к их компонентам

Для задания структур на языке MATLAB можно использовать операторы присваивания, что иллюстрирует следующий пример:

» man.name='Иван';

» man.surname-'Петров':
» man.date-1956;
» man.height=170.5;
» man.weight=70.34;

Здесь построена базовая структура без индексного указателя. Теперь можно просмотреть полученную структуру, просто указав ее имя:

```
» man
man =
name: 'Иван'
surname: 'Петров'
date: 1956
height: 170.5000
weight: 70.3400
```

Нетрудно догадаться, что компоненты структуры можно вызывать по имени и менять их значения. При этом имя компонента состоит из имени структуры и имени поля, разделенных точкой. Это поясняют следующие примеры:

```
» man.date
ans =
1956
» man.date=1964
man =
name: 'Иван'
surname: 'Петров'
date: 1964
height: 170.5000
weight: 70.3400
```

Как уже отмечалось, в MATLAB 6.0 существует проблема с записью символов кирилли-цы в командном режиме. Так, в командном режиме нельзя вводить в аргументы структур малую букву «с» русского алфавита — она ведет к переводу строки. Этого ограничения нет при задании структур в программах, хотя и в этом случае ошибки при вводе символов кириллицы не исключены.

Для создания массива структур вводится их индексация. Например, вектор структур можно создать, введя индекс в скобках после имени структуры. Так, для создания новой, второй структуры, можно поступить следующим образом:

```
»man(2).name='Петр';
»man(2).Surname='Сидоров';
» man(2).date=1959;
»man(2)
ans =
name: 'Петр'
```

surname: 'Сидоров'
date: 1959
height: [ ]
weight: [ ]
» man(2).surname
ans =
Сидоров
» length(man)
ans = .2

Обратите внимание на то, что не все поля данной структуры заполнены. Поэтому значением двух последних полей структуры 2 оказываются пустые массивы. Число структур в массиве структур позволяет найти функция length (см. последний пример). Эта же функция может использоваться и для нахождения размера любого вектора или размерности многомерного непустого массива, так как length(X)= MAX(size(X)), если X — непустой массив, и length(X)=0, если X=[ ].

Функция создания структур

Для создания структур используется следующая функция:

struct('fieldl' .VALUES1, 'field2' .VALUES2,...) — возвращает созданную данной функцией структуру, содержащую указанные в параметрах поля 'fieldn' с их значениями ' VALUESn'. Значением может быть массив ячеек;

struct(OBJ) — конвертирует объект OBJ в эквивалентную структуру или массив структур. OBJ может быть объектом или массивом Java.

Пример:

» S=struct('student'.'Иванов'.'group'.2.'estimate','good')
S =
student: 'Иванов'
group: 2
estimate: 'good'

Проверка имен полей и структур

Выполнение операций с полями и элементами полей выполняется по тем же правилам, что и при работе с обычными массивами. Однако существует ряд функций, осуществляющих специфические для структур операции [Помимо функций isstruct и isfields вы можете использовать для тестирования массивов структур функцию isa(имя объекта, 'struct') и команду или функцию whos имя объекта. — Примеч. ред.].

Приведенные ниже функции служат для тестирования имен полей и структур записей:

isfielcKS, 'field') — возвращает логическую 1, если 'field' является именем поля структуры S;

isstruct(S) — возвращает логическую 1, если S — структура, и 0 в ином случае. Их применение на примере структуры man показано ниже:

```
» isfieldCman.'name')
ans =
1
» isfield(man.'family')
ans =
0
» isstruct(man)
ans =
1
» isstruct(many)
??? Undefined function or variable 'many'.
» isstruct('many')
ans =
0
```

Функция возврата имен полей

Следующая функция позволяет вывести имена полей заданной структуры:

fieldnames (S) — возвращает имена полей структуры S в виде массива ячеек (см. урок 15). Пример:

```
» fieldnames(man)
ans =
'name'
'surname'
'date'
'height'
'weight'
```

Функция возврата содержимого полей структуры

В конечном счете работа со структурами сводится к выводу и использованию содержимого полей. Для возврата содержимого поля структуры S служит функция getfield:

getfield(S, 'field') — возвращает содержимое поля структуры S, что эквивалентно S. field;

getfield(S.{i,j}, 'field', {k}) - 'эквивалентно F=S(i J).field(k). Пример:

```
» getfield(man(2),'name')
ans =
Петр
```

Функция присваивания значений полям

Для присваивания полям заданных значений используется описанная далее функция selfield:

setfielcKS. 'field' ,V) — возвращает структуру S с присвоением полю 'field' значения V, что эквивалентно S.field=V;

setfie1d(S,{i .j}, 'field1 ,{k},V) - эквивалентно S(i ,j).field(k)=V. Пример:

» setfield(man(2).'name'.'Николай')

ans =

name: 'Николай'

surname: 'Сидоров'

date: 1959

height: [ ]

weight: [ ]

Удаление полей

Для удаления полей структуры можно использовать следующую функцию:

rmfield(S, 'field') — возвращает структуру S с удаленным полем S.'field';

rmfield(S,FIELDS) — возвращает структуру S с несколькими удаленными полями. Список удаляемых полей FIELDS задается в виде массива символов (строки) или строкового массива ячеек.

Пример:

» rmfield(man(2).'surname')

ans =

name: 'Петр'

date: 1959

height: []

weight: []

Применение массивов структур

Массивы структур находят самое широкое применение. Например, они используются для представления цветных изображений. Известно, что цветные изображения формата RGB состоят из массивов интенсивности трех цветов - красного R, зеленого G и синего B. При этом каждый массив содержит данные о координатах точки (они определяются целочисленными индексами массива) и о ее яркости (число от 0 до 1 в формате чисел с плавающей запятой). Чтобы некоторое изображение, например pic, несло данные о цвете всех точек, придется представить изображение массивом структур с полями pic.r, pic.g и pic.b.

Еще более сложные структуры (но, в принципе, вполне очевидные) нужны для разработки баз данных, например о работниках предприятия, службах города, городах страны и т. д. Во всех этих случаях особенно важна возможность доступа к отдельным полям структур и возможность присвоения таким полям уникальных имен.

Может показаться, что этот тип данных имеет малое отношение к математическим возможностям системы MATLAB. Однако надо помнить, что поиск информации в больших базах данных, сортировка этой информации и прочие операции, не говоря уже о сложной обработке массивов изображений, — все это примеры явно математических, хотя и достаточно специфических, операций. Причем операций нередко с многомерными структурами. Возможность MATLAB выполнять подобные операции быстро и эффективно (прежде всего с позиций минимальных затрат памяти) открывает перед этой системой очень большие возможности в этой области — впрочем, пока еще ждущие своей реализации.

**Лабораторная работа №8. Функции в С++.**

**Цель работы:** изучить реализацию на языке С++ отношений между классами: агрегации, наследования, зависимости

**Общие сведения**

Очень часто в программировании необходимо выполнять одни и те же действия. Например, мы хотим выводить пользователю сообщения об ошибке в разных местах программы, если он ввел неверное значение. без функций это выглядело бы так:

```
#include <iostream>
#include <string>

using namespace std;

int main()
{
   string valid_pass = "qwerty123";
   string user_pass;
   cout << "Введите пароль: ";
   getline(cin, user_pass);
   if (user_pass == valid_pass) {
      cout << "Доступ разрешен." << endl;
   } else {
      cout << "Неверный пароль!" << endl;
   }
   return 0;
}
```

А вот аналогичный пример с функцией:

```
#include <iostream>
#include <string>

using namespace std;

void check_pass (string password)
{
   string valid_pass = "qwerty123";
   if (password == valid_pass) {
      cout << "Доступ разрешен." << endl;
   } else {
      cout << "Неверный пароль!" << endl;
   }
}
```

```
int main()
{
   string user_pass;
   cout << "Введите пароль: ";
   getline (cin, user_pass);
   check_pass (user_pass);
   return 0;
}
```

По сути, после компиляции не будет никакой разницы для процессора, как для первого кода, так и для второго. Но ведь такую проверку пароля мы может делать в нашей программе довольно много раз. И тогда получается копипаста и код становится нечитаемым. Функции — один из самых важных компонентов языка C++.

Любая функция имеет тип, также, как и любая переменная.
Функция может возвращать значение, тип которого в большинстве случаев аналогично типу самой функции.
Если функция не возвращает никакого значения, то она должна иметь тип void (такие функции иногда называют процедурами)

При объявлении функции, после ее типа должно находиться имя функции и две круглые скобки — открывающая и закрывающая, внутри которых могут находиться один или несколько аргументов функции, которых также может не быть вообще.

после списка аргументов функции ставится открывающая фигурная скобка, после которой находится само тело функции.
В конце тела функции обязательно ставится закрывающая фигурная скобка.
Пример построения функции

```
#include <iostream>
using namespace std;

void function_name ()
{
   std::cout << "Hello, world" << std::endl;
}

int main()
{
   function_name(); // Вызов функции
   return 0;
}
```

Перед вами тривиальная программа, Hello, world, только реализованная с использованием функций.

Если мы хотим вывести «Hello, world» где-то еще, нам просто нужно вызвать соответствующую функцию. В данном случае это делается так: function_name();. Вызов функции имеет вид имени функции с последующими круглыми скобками. Эти скобки могут быть пустыми, если функция не имеет аргументов. Если же аргументы в самой функции есть, их необходимо указать в круглых скобках.

Также существует такое понятие, как параметры функции по умолчанию. Такие параметры можно не указывать при вызове функции, т.к. они примут значение по умолчанию, указанно после знака присваивания после данного параметра и списке всех параметров функции.

В предыдущих примерах мы использовали функции типа void, которые не возвращают никакого значения. Как многие уже догадались, оператор return используется для возвращения вычисляемого функцией значения.

Рассмотрим пример функции, возвращающей значение на примере проверки пароля.

```
#include <iostream>
#include <string>

using namespace std;

string check_pass (string password)
{
   string valid_pass = "qwerty123";
   string error_message;
   if (password == valid_pass) {
      error_message = "Доступ разрешен.";
   } else {
      error_message = "Неверный пароль!";
   }
   return error_message;
}

int main()
{
   string user_pass;
   cout << "Введите пароль: ";
   getline (cin, user_pass);
   string error_msg = check_pass (user_pass);
```

```
    cout << error_msg << endl;
    return 0;
}
```

В данном случае функция check_pass имеет тип string, следовательно она будет возвращать только значение типа string, иными словами говоря строку. Давайте рассмотрим алгоритм работы этой программы.

Самой первой выполняется функция main(), которая должна присутствовать в каждой программе. Теперь мы объявляем переменную user_pass типа string, затем выводим пользователю сообщение «Введите пароль», который после ввода попадает в строку user_pass. А вот дальше начинает работать наша собственная функция check_pass().

В качестве аргумента этой функции передается строка, введенная пользователем.

Аргумент функции — это, если сказать простым языком переменные или константы вызывающей функции, которые будет использовать вызываемая функция.

При объявлении функций создается формальный параметр, имя которого может отличаться от параметра, передаваемого при вызове этой функции. Но типы формальных параметров и передаваемых функии аргументов в большинстве случаев должны быть аналогичны.

После того, как произошел вызов функции check_pass(), начинает работать данная функция. Если функцию нигде не вызвать, то этот код будет проигнорирован программой. Итак, мы передали в качестве аргумента строку, которую ввел пользователь.

Теперь эта строка в полном распоряжении функции (хочу обратить Ваше внимание на то, что переменные и константы, объявленные в разных функциях независимы друг от друга, они даже могут иметь одинаковые имена. В следующих уроках я расскажу о том, что такое область видимости, локальные и глобальные переменные).

Теперь мы проверяем, правильный ли пароль ввел пользователь или нет. если пользователь ввел правильный пароль, присваиваем переменной error_message соответствующее значение. если нет, то сообщение об ошибке.

После этой проверки мы возвращаем переменную error_message. На этом работа нашей функции закончена. А теперь, в функции main(), то значение, которое возвратила наша функция мы присваиваем переменной error_msg и выводим это значение (строку) на экран терминала.

Также, можно организовать повторный ввод пароля с помощью рекурсии (о ней мы еще поговорим). Если объяснять вкратце, рекурсия — это когда функция вызывает сама себя. Смотрите еще один пример:

```
#include <iostream>
#include <string>

using namespace std;

bool password_is_valid (string password)
{
	string valid_pass = "qwerty123";
	if (valid_pass == password)
		return true;
	else
		return false;
}

void get_pass ()
{
	string user_pass;
	cout << "Введите пароль: ";
	getline(cin, user_pass);
	if (!password_is_valid(user_pass)) {
		cout << "Неверный пароль!" << endl;
		get_pass (); // Здесь делаем рекурсию
	} else {
		cout << "Доступ разрешен." << endl;
	}
}

int main()
{
	get_pass ();
	return 0;
}
```

Функции очень сильно облегчают работу программисту и намного повышают читаемость и понятность кода, в том числе и для самого разработчика (не удивляйтесь этому, т. к. если вы откроете код, написанный вами полгода назад,не сразу поймете соль, поверьте на слово).

Не расстраивайтесь, если не сразу поймете все аспекты функций в C++, т. к. это довольно сложная тема и мы еще будем разбирать примеры с функциями в следующих уроках.

Совет: не бойтесь экспериментировать, это очень хорошая практика, а после прочтения данной статьи порешайте элементарные задачи, но с использованием функций. Это будет очень полезно для вас.

**Контрольные вопросы**

**Лабораторная работа №10. Хранение данных на внешних носителях.**

**Цель работы:** изучить реализацию на языке C++ отношений между классами: агрегации, наследования, зависимости

**Общие сведения**

STL обеспечивает общецелевые, стандартные классы и функции, которые реализуют наиболее популярные и широко используемые алгоритмы и структуры данных.

STL строится на основе шаблонов классов, и поэтому входящие в нее алгоритмы и структуры применимы почти ко всем типам данных.

Ядро библиотеки образуют три элемента: контейнеры, алгоритмы и итераторы.

Контейнеры (containers) - это объекты, предназначенные для хранения других элементов. Например, вектор, линейный список, множество.

Ассоциативные контейнеры (associative containers) позволяют с помощью ключей получить быстрый доступ к хранящимся в них значениям.

В каждом классе-контейнере определен набор функций для работы с ними. Например, список содержит функции для вставки, удаления и слияния элементов.

Алгоритмы (algorithms) выполняют операции над содержимым контейнера. Существуют алгоритмы для инициализации, сортировки, поиска, замены содержимого контейнеров. Многие алгоритмы предназначены для работы с последовательностью (sequence), которая представляет собой линейный список элементов внутри контейнера.

Итераторы (iterators) - это объекты, которые по отношению к контейнеру играют роль указателей. Они позволяют получить доступ к содержимому контейнера примерно так же, как указатели используются для доступа к элементам массива.

С итераторами можно работать так же, как с указателями. К ним можно применить операции *, инкремента, декремента. Типом итератора объявляется тип iterator, который определен в различных контейнерах. Существует пять типов итераторов:

Итераторы ввода (input iterator) поддерживают операции равенства, разыменования и инкремента.

==, !=, *i, ++i, i++, *i++

Специальным случаем итератора ввода является istream_iterator.

Итераторы вывода (output iterator) поддерживают операции разыменования, допустимые только с левой стороны присваивания, и инкремента.

++i, i++, *i = t, *i++ = t

Специальным случаем итератора вывода является ostream_iterator.

Однонаправленные итераторы (forward iterator) поддерживают все операции итераторов ввода/вывода и, кроме того, позволяют без ограничения применять присваивание.

==, !=, =, *i, ++i, i++, *i

Двунаправленные итераторы (bidirectional iterator) обладают всеми свойствами forward-итераторов, а также имеют дополнительную операцию декремента (--i, i--, *i--), что позволяет им проходить контейнер в обоих направлениях.

Итераторы произвольного доступа (random access iterator) обладают всеми свойствами bidirectional-итераторов, а также поддерживают операции сравнения и адресной арифметики, то есть непосредственный доступ по индексу.

i += n, i + n, i -= n, i - n, i1 - i2, i[n], i1 < i2, i1 <= i2, i1 > i2, i1 >= i2

В STL также поддерживаются обратные итераторы (reverse iterators). Обратными итераторами могут быть либо двунаправленные итераторы, либо итераторы произвольного доступа, но проходящие последовательность в обратном направлении.

Вдобавок к контейнерам, алгоритмам и итераторам в STL поддерживается еще несколько стандартных компонентов. Главными среди них являются распределители памяти, предикаты и функции сравнения.

У каждого контейнера имеется определенный для него распределитель памяти (allocator), который управляет процессом выделения памяти для контейнера.

По умолчанию распределителем памяти является объект класса allocator. Можно определить собственный распределитель.

В некоторых алгоритмах и контейнерах используется функция особого типа, называемая предикатом. Предикат может быть унарным и бинарным. Возвращаемое значение: истина либо ложь. Точные условия получения того или иного значения определяются программистом. Тип унарных предикатовUnPred, бинарных - BinPred . Тип аргументов соответствует типу хранящихся в контейнере объектов.

Определен специальный тип бинарного предиката для сравнения двух элементов. Он называется функцией сравнения (comparison function). Функция возвращает истину, если первый элемент меньше второго. Типом функции является тип Comp .

Особую роль в STL играют объекты-функции.

Объекты-функции - это экземпляры класса, в котором определена операция "круглые скобки" (). В ряде случаев удобно заменить функцию на объект-функцию. Когда объект-функция используется в качестве функции, то для ее вызова используется operator().

```
class less
{

public:
  bool operator()(int x, int y)
  {
    return x < y;
   }
};
```

В STL определены два типа контейнеров: последовательности и ассоциативные.

Ключевая идея для стандартных контейнеров заключается в том, что когда это представляется разумным, они должны быть логически взаимозаменяемыми. Пользователь может выбирать между ними, основываясь на соображениях эффективности и потребности в специализированных операциях. Например, если часто требуется поиск по ключу, можно воспользоваться map (ассоциативным массивом). С другой стороны, если преобладают операции, характерные для списков, можно воспользоваться контейнером list. Если добавление и удаление элементов часто производится в концы контейнера, следует подумать об использовании очереди queue, очереди с двумя концами deque, стека stack. По умолчанию пользователь должен использовать vector ; он реализован, чтобы хорошо работать для самого широкого диапазона задач.

Идея обращения с различными видами контейнеров и, в общем случае, со всеми видами источников информации - унифицированным способом ведет к понятию обобщенного программирования. Для поддержки этой идеи STL содержит множество обобщенных алгоритмов. Такие алгоритмы избавляют программиста от необходимости знать подробности отдельных контейнеров.

В STL определены следующие классы-контейнеры (в угловых скобках указаны заголовочные файлы, где определены эти классы):

bitset - множество битов <bitset.h>

vector - динамический массив <vector.h>

list - линейный список <list.h>

deque - двусторонняя очередь <deque.h>

stack - стек <stack.h>

queue - очередь <queue.h>
priority_queue - очередь с приоритетом <queue.h>
map - ассоциативный список для хранения пар ключ/значение, где с каждым ключом связано одно значение <map.h>
multimap - с каждым ключом связано два или более значений <map.h>
set - множество <set.h>
multiset - множество, в котором каждый элемент не обязательно уникален <set.h>

Типы:
value_type - тип элемента
allocator_type - тип распределителя памяти
size_type - тип индексов, счетчика элементов и т.д.
iterator - ведет себя как value_type *
reverse_iterator просматривает контейнер в обратном порядке
reference - ведет себя как value_type &
key_type - тип ключа (только для ассоциативных контейнеров)
key_compare - тип критерия сравнения (только для ассоциативных контейнеров)
mapped_type - тип отображенного значения

Итераторы:
begin() - указывает на первый элемент
end() - указывает на элемент, следующий за последним
rbegin() - указывает на первый элемент в обратной последовательности
rend() - указывает на элемент, следующий за последним в обратной последовательности

Доступ к элементам:
front() - ссылка на первый элемент
back() - ссылка на последний элемент
operator [](i) - доступ по индексу без проверки
at(i) - доступ по индексу с проверкой

Включение элементов:
insert(p, x) - добавление x перед элементом, на который указывает p
insert(p, n, x) - добавление n копий x перед p
insert(p, first, last) - добавление элементов из [first:last] перед p
push_back(x) - добавление x в конец
push_front(x) - добавление нового первого элемента (только для списков и очередей с двумя концами)

Удаление элементов:
pop_back() - удаление последнего элемента
pop_front() - удаление первого элемента (только для списков и очередей с двумя концами)
erase(p) - удаление элемента в позиции p
erase(first, last) - удаление элементов из [first:last]
clear() - удаление всех элементов

Другие операции:
size() - число элементов
empty() - контейнер пуст
capacity() - память, выделенная под вектор (только для векторов)

reserve(n) - выделяет память для контейнера под n элементов

resize(n) - изменяет размер контейнера (только для векторов, списков и очередей с двумя концами)

swap(x) - обмен местами двух контейнеров

==, !=, < операции сравнения

Операции присваивания:

operator =(x) - контейнеру присваиваются элементы контейнера x

assign(n, x) - присваивание контейнеру n копий элементов x (не для ассоциативных контейнеров)

assign(first, last) - присваивание элементов из диапазона [first:last]

Ассоциативные операции:

operator [](k) - доступ к элементу с ключом k

find(k) - находит элемент с ключом k

lower_bound(k) - находит первый элемент с ключом k

upper_bound(k) - находит первый элемент с ключом, большим k

equal_range(k) - находит lower_bound (нижнюю границу) и upper_bound (верхнюю границу) элементов с ключом k

Вектор vector в STL определен как динамический массив с доступом к его элементам по индексу.

```
template <class T, class Allocator = allocator<T> > class std::vector
{
// ...
};
```

где T - тип предназначенных для хранения данных. Allocator задает распределитель памяти, который по умолчанию является стандартным.

В классе vector определены следующие конструкторы:

```
explicit vector(const Allocator &a = Allocator());
explicit vector(size_type число, const T &значение = T(),
                const Allocator &a = Allocator());

vector(const vector<T, Allocator> &объект);

template<class InIter> vector(InIter начало, InIter конец,

const Allocator &a = Allocator());
```

Первая форма представляет собой конструктор пустого вектора. Во второй форме конструктора вектора число элементов - это число, а каждый элемент равен значению значение. Параметр значение может быть значением по умолчанию. Третья форма конструктора вектор - это конструктор копирования. Четвертая форма - это конструктор вектора, содержащего диапазон элементов, заданный итераторами начало и конец.

```
vector<int> a;
vector<double> x(5);
vector<char> c(5, '*');
vector<int> b(a); // b = a
```

Для любого объекта, который будет храниться в векторе, должен быть определен конструктор по умолчанию. Кроме того, для объекта должны быть определены операторы < и == .

Для класса вектор определены следующие операторы сравнения:

==, <, <=, !=, >, >=.

Кроме этого, для класса vector определяется оператор индекса [].

Новые элементы могут включаться с помощью функций insert(), push_back(), resize(), assign().

Существующие элементы могут удаляться с помощью функций erase(), pop_back(), resize(), clear()

Доступ к отдельным элементам осуществляется с помощью итераторов begin(), end(), rbegin(), rend().

Манипулирование контейнером, сортировка, поиск в нем и тому подобное возможно с помощью глобальных функций файла - заголовка<algorithm.h>.

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

using namespace std;

int main(void)
{
  vector<int> v;
  for(int i = 0; i < 10; i++)
  {
    v.push_back(i);
  }
  cout << "size = " << v.size() << "\n";
  for (int i = 0; i < 10; i++)
  {
    cout << v[i] << " ";
  }
  cout << endl;
  for (int i = 0; i < 10; i++)
  {
    v[i] = v[i] + v[i];
  }
  for (int i = 0; i < v.size(); i++)
  {
    cout << v[i] << " ";
  }
  cout << endl;
  return 0;
}
```

Доступ к вектору через итератор

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

using namespace std;

int main(void)
{
  vector<int> v;
  for (int i = 0; i < 10; i++)
  {
    v.push_back(i);
```

```
  }
  cout << "size = " << v.size() << "\n";
  vector<int>::iterator p = v.begin();
  while (p != v.end())
  {
    cout << *p << " ";
    p++;
  }
  return 0;
}
```

Вставка и удаление элементов

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

using namespace std;

int main(void)
{
  vector<int> v(5, 1);
  // вывод
  for (int i = 0; i < 5; i++)
  {
    cout << v[i] << " ";
  }
  cout << endl;
  vector<int>::iterator p = v.begin();
  p += 2;
  // вставить 10 элементов со значением 9
  v.insert(p, 10, 9);
  //вывод
  p = v.begin();
  while (p != v.end())
  {
    cout << *p << " ";
    p++;
  }
  // удалить вставленные элементы
  p = v.begin();
  p += 2;
  v.erase (p, p + 10);
  // вывод
  p = v.begin();
  while (p != v.end())
  {
    cout << *p << " ";
    p++;
  }
  return 0;
}
```

Вектор содержит объекты пользовательского класса

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

```
#include "student.h"

using namespace std;
int main(void)
{
  vector<STUDENT> v(3);
  v[0] = STUDENT("Иванов", 45.9);
  v[1] = STUDENT("Петров", 30.4);
  v[0] = STUDENT("Сидоров", 55.6);
  // вывод
  for (int i = 0; i < 3; i++)
  {
    cout << v[i] << " ";
  }
  cout << endl;
  return 0;
}
```

Ассоциативный массив содержит пары значений. Зная одно значение, называемое ключом (key), мы можем получить доступ к другому, называемому отображенным значением (mapped value).

Ассоциативный массив можно представить как массив, для которого индекс не обязательно должен иметь целочисленный тип:

V &operator [](const K &) возвращает ссылку на V, соответствующий K .

Ассоциативные контейнеры - это обобщение понятия ассоциативного массива.

Ассоциативный контейнер map - это последовательность пар (ключ, значение), которая обеспечивает быстрое получение значения по ключу. Контейнер map предоставляет двунаправленные итераторы.

Ассоциативный контейнер map требует, чтобы для типов ключа существовала операция < . Он хранит свои элементы отсортированными по ключу так, что перебор происходит по порядку.

Спецификация шаблона для класса map:

```
template<class Key, class T, class Comp = less<Key>,
        class Allocator = allocator<pair> >
class std::map;
```

В классе map определены следующие конструкторы:

```
explicit map(const Comp &c = Comp(), const Allocator &a = Allocator());
map(const map<Key, T, Comp, Allocator> &ob);
template<class InIter> map(InIter first, InIter last, const Comp &c = Comp(),

const Allocator &a = Allocator());
```

Первая форма представляет собой конструктор пустого ассоциативного контейнера, вторая - конструктор копии, третья - конструктор ассоциативного контейнера, содержащего диапазон элементов.

Определена операция присваивания:

map &operator =(const map &);

Определены следующие операции: ==, <, <=, !=, >, >= .

В map хранятся пары ключ/значение в виде объектов типа pair .

Создавать пары ключ/значение можно не только с помощью конструкторов класса pair, но и с помощью функции make_pair, которая создает объекты типа pair , используя типы данных в качестве параметров.

Типичная операция для ассоциативного контейнера - это ассоциативный поиск при помощи операции индексации ([]).

mapped_type &operator [](const key_type &K);

Множества set можно рассматривать как ассоциативные массивы, в которых значения не играют роли, так что мы отслеживаем только ключи.

```
template<classT, class Cmp = less<T>, class Allocator = allocator<T> >

class std::set
{
  //...
};
```

Множество, как и ассоциативный массив, требует, чтобы для типа T существовала операция "меньше" (< ). Оно хранит свои элементы отсортированными, так что перебор происходит по порядку.

Каждый алгоритм выражается шаблоном функции или набором шаблонов функций. Таким образом, алгоритм может работать с очень разными контейнерами, содержащими значения разнообразных типов. Алгоритмы, которые возвращают итератор, как правило, для сообщения о неудаче используют конец входной последовательности. Алгоритмы не выполняют проверки диапазона на их входе и выходе. Когда алгоритм возвращает итератор, это будет итератор того же типа, что и был на входе. Алгоритмы в STL реализуют большинство распространенных универсальных операций с контейнерами, такие как просмотр, сортировка, поиск, вставка и удаление элементов.

Алгоритмы определены в заголовочном файле <algorithm.h> .

Ниже приведены имена некоторых наиболее часто используемых функций-алгоритмов STL.

Немодифицирующие операции.

for_earch() - выполняет операции для каждого элемента последовательности

find() - находит первое вхождение значения в последовательность

find_if() - находит первое соответствие предикату в последовательности

count() - подсчитывает количество вхождений значения в последовательность

count_if() - подсчитывает количество выполнений предиката в последовательности

search() - находит первое вхождение последовательности как подпоследовательности

search_n() - находит n-е вхождение значения в последовательность

Модифицирующие операции.

copy() - копирует последовательность, начиная с первого элемента

swap() - меняет местами два элемента

replace() - заменяет элементы с указанным значением

replace_if() - заменяет элементы при выполнении предиката

replace_copy() - копирует последовательность, заменяя элементы с указанным значением

replace_copy_if() - копирует последовательность, заменяя элементы при выполнении предиката

fill() - заменяет все элементы данным значением

remove() - удаляет элементы с данным значением

remove_if() - удаляет элементы при выполнении предиката

remove_copy() - копирует последовательность, удаляя элементы с указанным значением

remove_copy_if() - копирует последовательность, удаляя элементы при выполнении предиката

reverse() - меняет порядок следования элементов на обратный

random_shuffle() - перемещает элементы согласно случайному равномерному распределению ("тасует" последовательность)

transform() - выполняет заданную операцию над каждым элементом последовательности

unique() - удаляет равные соседние элементы

unique_copy() - копирует последовательность, удаляя равные соседние элементы

Сортировка.

sort() - сортирует последовательность с хорошей средней эффективностью

partial_sort() - сортирует часть последовательности

stable_sort() - сортирует последовательность, сохраняя порядок следования равных элементов

lower_bound() - находит первое вхождение значения в отсортированной последовательности

upper_bound() - находит первый элемент, больший чем заданное значение

binary_search() - определяет, есть ли данный элемент в отсортированной последовательности

merge() - сливает две отсортированные последовательности

Работа с множествами.

includes() - проверка на вхождение

set_union() - объединение множеств

set_intersection() - пересечение множеств

set_difference() - разность множеств

Минимумы и максимумы.

min() - меньшее из двух

max() - большее из двух

min_element() - наименьшее значение в последовательности

max_element() - наибольшее значение в последовательности

Перестановки.

next_permutation() - следующая перестановка в лексикографическом порядке

prev_permutation() - предыдущая перестановка в лексикографическом порядке