Лекция 1. Структура программы

**1. Простейшая программа на языке С++**

Программа на языке С++, как и на уже знакомом вам Паскале, содержит две основные части:

объявление данных, с которыми будут производиться некоторые действия (вычисления, вывод на экран и т.п.);

операторную часть, которая задает последовательность действий, выполняемых с этими данными.

Объявление переменной содержит описание ее типа и имя переменной. Приведем примеры объявления.

Cтрока объявления

**int R;**

состоит из имени переменной R и служебного слова int, которое указывает, что в переменной R можно хранить целые числа.

В строке

**float S;**

тип данных задается служебным словом float, который указывает, что в переменной с именем S могут храниться вещественные числа – то есть числа, которые могут содержать как целую, так и дробную часть. При записи вещественных чисел на Си целую и дробную часть можно разделять точкой (например, 1.25 – это одна целая двадцать пять сотых), а можно, как и в Паскале, использовать экспоненциальную форму записи.

Операторная часть программы обязательно содержит так называемую «главную функцию»:

**void main(void)**

**{**

**}**

Функция начинается с заголовкаvoid main(void), вид которого пока можно просто запомнить. Между фигурными скобками находятся операторы программы, каждый из которых **обязательно заканчивается точкой с запятой**. Оператор задает действия, которые надо выполнить с одной или несколькими переменными.

 Объявления переменных могут размещаться как до главной функции, так и внутри нее, после фигурной скобки.

Рассмотрим пример первой программы на языке С++:

**int R1;**

**int R2;**

**int Sum;**

**void main(void) {**

**R1=5;**

**R2=230;**

**Sum=3\*R1+R2;**

**}**

Можно считать, что при запуске программы ее работа начинается с выполнения первого из этих операторов – оператора присваиванияR1=5**;** При его выполнении в переменную R1 записывается число 5. Это число будет храниться в R1 до тех пор, пока какой-нибудь другой оператор не запишет туда другое число. Операторы выполняются один за другим в порядке их записи:

второй оператор занесет число 230 в R2;

при выполнении оператора Sum=3\*R1+R2; вычислится арифметическое выражение 3\*5+230 и полученное значение 245 запишется в переменную Sum. При записи арифметических выражений используются общепринятые обозначения операций: + сложение,- вычитание, \* умножение и / деление как целых, так и вещественных чисел.

Рассмотрение первого примера закончим следующими замечаниями.

1. Язык С++ создан в результате развития языка С. Эти языки настолько близки, что рассмотренная выше программа выглядит одинаково на обоих языках. Для перехода от языка С++ к С достаточно изменить в исходном файле расширение СPP на C. Приведенный пример скомпилируется без ошибок тем же компилятором, но как программа на C. В дальнейшем мы будем считать, что изучаем язык C++, но будем указывать, какие из изучаемых элементов и синтаксических правил различны для С и С++.

2. Имя переменной – это последовательность букв и цифр, начинающаяся с буквы. Таким образом, Ab32 – это допустимое имя языка, а 1А – недопустимое, т.к. начинается не с буквы. В С и С++ прописная и строчная буквы считаются различными, поэтому Ab32 и ab32 это разные имена двух разных переменных.

3. Несколько переменных одинакового типа можно объявить одной строкой, перечислив их через запятую, например,

**int R1,R2;**.

При объявлении переменных им можно сразу присваивать начальные значения:

**int R1=10,R=21;**.

Если значения не присвоены, и переменные, как в данном примере, объявлены вне главной функции, в них в начале работы программы хранятся нули. Если переменная объявляется без инициализации внутри функции, в ней может оказаться любое значение.

4. Как и в языке Паскаль, вещественные и целочисленные переменные имеют различное представление в памяти машины, но данные вещественных и целых типов совместимы по присваиванию. Это означает, что применение в программе оператора R1=2.51; не будет считаться ошибкой. Если в программе объявлена вещественная переменная

**float V**,

то можно выполнить присваивание R1=V1.

Замечание. Компилятор Visual С++, разработанный фирмой Microsoft, выводит в таких ситуациях предупреждение

warning C4244: '=': conversion from 'float' to 'int', possible loss of data

Компилятор BC.exe фирмы Borland не выводит даже предупредительных сообщений.

Если оператор присваивания

**R1=V1**

записывает в целочисленную переменную вещественное число, то компилятор автоматически организует вызов функции преобразования представления числа, которая получает целое число отбрасыванием дробной части вещественного. Таким образом, значение V1 равное 3.1, преобразуется в 3. Из 3.9 также будет получено число 3.

**2. Вывод данных на экран**

Рассмотренная выше первая программа производит вычисления, но не предусматривает задания человеком исходных данных с клавиатуры и вывода результатов на экран дисплея.

Для ввода исходных данных в программе можно использовать оператор вызова функции форматного ввода scanf(), а для вывода - оператор вызова функции форматного вывода printf().

В следующем примере при запуске программы на экран выводится приглашение *«Ведите два числа»,* в ответ на которое человек должен набрать на клавиатуре два целых числа, разделив их пробелом или нажатием клавиши Enter. Программа вычисляет сумму и произведение этих чисел и выводит их на экран:

**#include <stdio.h>**

**int Sum, Mul, R1,R2;**

**void main(void)**

**{ printf("Введите два числа ");**

**scanf(“ %d %d”, &R1,&R2);**

**Sum=R1+R2;**

**Mul=R1\*R2;**

**printf(“Сумма равна %d Произведение равно %d”, Sum,Mul);**

**}**

Рассмотрим пример более подробно.

Оператор **printf("Введите два числа ");** выводит на экран заключенную в кавычки строку текста. Эта строка задает *формат вывода данных*. Кроме обычного текста в строке формата могут присутствовать записанные после знака % латинские буквы d, f, i, s и другие.

В последнем операторе программы

**printf(“Сумма равна %d Произведение равно %d”, Sum,Mul);**

знаками % в строке отмечаются те места, где программа вставит в строку значения переменных, причем записанная после процента буква d указывает, что это будут целые числа. В этом же операторе после строки формата перечислены через запятую выводимые переменные. На месте первого процента будет выведено значение из переменной Sum, а на месте второго – из переменной Mul.

Кстати, то же самое средствами языка С++ можно записать компактнее с использованием операции вывода в поток:

**cout<<“Сумма равна=”<< Sum <<”Произведение равно=”<<Mul;**

Если в программе объявлена вещественная переменная, пусть

float fVar=3.5;,

то при выводе значения fVar на экран в строке формата после процента указывают букву f: printf(“ %f”,fVar);.

При вводе данных, как и при выводе, указывают строку формата и имена переменных, в которые занесутся введенные с клавиатуры значения. В операторе

Scanf (“ %d %d”, &R1,&R2);

из строки формата “ %d %d” видно, что далее будут указаны имена двух переменных целого типа. Следует только заметить, что перед именами переменных нужно записывать значки & - операции получения адреса.

***Замечание****.*

Если в программе используются вызовы функций ввода-вывода (таких, как scanf() или printf()) то для их правильной обработки компилятором в программу надо включить строку **#include <stdio.h>.**

Дляиспользованием операции ввода и вывода в поток необходимо вставить в программу следующую строку:

**#include <iostream.h>**

В интегрированной среде Си есть подкаталог include, в котором находятся текстовые файлы с расширением h. Они называются заголовочными файлами, среди них находится и файл stdio.h. Перед компиляцией исходный текст программы на языке Си просматривается специальной программой, которая называется препроцессором.

Встретив строку #include <stdio.h> препроцессор находит на диске файл stdio.h и включает его целиком в нашу программу вместо этой строки. А уже после этого файл обрабатывается компилятором и переводится в машинные коды.

Если просмотреть содержимое файла stdio.h, в нем можно найти строки

int \_Cdecl printf(const char \_FAR \*\_\_format,...);

int \_Cdecl scanf (const char \_FAR \*\_\_format,...);,

ради которых stdio.h был включен в исходный текст программы.

Это так называемые прототипы функций, которые нужны компилятору, чтобы правильно сформировать вызовы функций printf() и scanf(). Кроме них stdio.h содержит константы, описания типов данных, прототипы других функций, применяемых при стандартном вводе или выводе данных. (Потому он так и называется – сокращение от standard input output).

**3. Стандартные функции**

При составлении выражений *в* языке Си используется много разнообразных стандартных функций. Как мы помним, функция – это записанная отдельно от основного текста часть программы, которая вызывается оператором вызова функции – записывается имя функции, после которого в скобках записываются фактические параметры (значения аргументов функции). Так, если в программе объявлены переменные

**float Res,V1=9**;

то в арифметических выражениях можно использовать вызов функций вычисления синуса sin(V1), косинуса cos(V1) квадратного корня sqrt(V1) и пр. При выполнении оператора

**Res = 2\*sqrt(V1);**

вычисляется квадратный корень изV1 - вещественное число 3, умножится на два и результат запишется в переменную Res. То есть, при вызове функции вычисляется ее значение (принято говорить, что функция возвращает вычисленное значение в ответ на вызов) и это значение используется в выражении как обычная переменная.

При составлении программы можно использовать вызовы:

log(V1) – вычисления натурального логарифма,

log10(V1) – вычисления десятичного логарифма,

exp(V1) – вычисления показательной функции с основанием e,

asin(V1), atan(V1) – вычисления арксинуса и арктангенса.

При использовании прямых и обратных тригонометрических функций угол задается и получается в радианах.

Функция может иметь несколько параметров. Например, функция atan2(y,x), которой передается два вещественных аргумента, вычисляет арктангенс угла, лежащего в прямоугольном треугольнике с катетами x,y против стороны y, причем второй аргумент x может быть равным нулю (при этом арктангенс будет равен π/2).

Прототипы математических функций находятся в заголовочном файле math.h. Если программа использует вызовы стандартных математических функций, то для их правильной обработки компилятором в программу надо включить строку

#include <math.h>.

Вспомним, что по умолчанию компилятор преобразует вещественное число в целое отбрасыванием дробной части Если программисту необходим другой способ округления, он должен включить в свою программу заголовочный файл с прототипами математических функций и вызвать функцию округления вещественного числа с недостатком

R1=floor(V1); или с избытком r=ceil(S);.

Для округления до ближайшего целого нет стандартной функции, но можно выполнить оператор

R1=floor(V1+0.5);

Как и в Паскале, можно узнать, сколько байтов требуется для хранения переменной, применив к ней операцию sizeof(). Объявим переменные

int A,B;

float C;

Операторы

A=sizeof(int);

A=sizeof(B);

дадут одинаковый результат – размер переменной типа int.

Оператор A=sizeof(C); запишет в A число 4 – размер переменно типа float.

**4. Реализация функций в тексте программы**

Последовательность команд, реализующая вычисление стандартной функции не формируется компилятором. Эти функции хранятся в отдельном файле – системной библиотеке функций языка Си. Если в библиотеке нет необходимой функции, программист может реализовать ее самостоятельно.

Любая функция, как и главная функция main() состоит из заголовка и заключенного в фигурные скобки тела функции. Пусть, например, нам необходима функция area (l,w) вычисления площади прямоугольного участка по его длине l и ширине w. Эта функция может быть реализована следующим образом

**float area(int x, int y)**

**{float s;**

**s= x\*y;**

**return s;**

**}**

Заголовок состоит из имени функции, после которого следует заключенный в скобки список *формальных* параметров (в данном случае это целочисленные параметры int x и int y). Перед именем функции указывается тип возвращаемого функцией значения. В данном примере слово float указывает, что функция вычислит площадь в виде вещественного, а не целого числа.

В фигурных скобках, как и в главной функции, находятся объявления переменных и операторы, реализующие вычисление функции. Вычисления должны обязательно заканчиваться оператором return, возвращающим вычисленное значение. В данном примере оператор возвращает значение, помещенное в переменную s. Но вместо переменной можно также записывать выражение, например, приведенную выше функцию можно было реализовать так:

**float area(int x, int y)**

**{return x\*y;**

**}**

В функции main мы использовали обозначение void и вместо возвращаемого значения и вместо списка параметров – это тоже тип данных, не знаю, как его лучше называть, то ли пустой, то ли неопределенный это пустые, не имеющие значения данные. Неизвестно даже, сколько байтов нужно для их хранения – поэтому и объявить переменную типа void нельзя. Этот тип указывают как раз там, где обычно ожидают какого-нибудь значения, а его нет. То есть, заголовок void main(void), указывает, что функция main не получает параметров и не возвращает значения. (На Паскале такая конструкция называлась бы процедурой без параметров). Поместим вызов нашей функции в главную программу:

**void main(void)**

**{float A,B;**

**int b,R;**

**b = 4;**

**A = area(5,b);**

**B = area(20,30);**

**}**

При выполнении программы сначала оператором b=4**;** в переменную b занесется число четыре. Следующий оператор вызовет вычисление функции area (5,b); для того, чтобы записать ее значение 25 в переменную A.

Выполнение операции вызова функции area(5,b) заключается в следующем:

– программа перестает выполнять операторы главной функции main()и переходит к выполнению функции area(), при этом ей передаются указанные при вызове значения параметров – в параметр x запишется число 5, а в параметр y значение переменной b – число четыре;

– выполняется первый оператор, записанный в теле функцииarea(), то есть строка s= x\*y;. При этом функция вычислит произведение 5\*4 и запишет его в переменную s.

– у нас очень короткая функция, но в общем случае операторы функции выполняются до тех пор, пока не встретится оператор выхода из функции return;

– в этом операторе после слова return помещается выражение, определяющее значение функции. Выполнение оператора заключается в вычислении выражения и возврате в то место, откуда функция вызывалась, но уже с результатом – значением функции.

Как видим, из описания самой функции нельзя увидеть, чему равны параметры x и y. Они потому и называются формальными, что получают фактические значения только во время выполнения программы – в момент вызова функции. Так, при следующем вызове B=area(20,30); формальные параметры x и y получат значения 20 и 10. Как мы помним из предыдущих глав, процедуры и функции тем и хороши, что позволяют проделывать одни и те же действия, но с разными исходными данными.

Замечание. Когда компилятор переводит строки языка С++ в машинные коды, он, как и мы, просматривает программу сверху вниз. Поэтому, к моменту вызова функции area() он уже обработал описание функции и знает, что функция должна получать в качестве параметров два целых числа и возвращать вещественный результат. Поэтому если вставить в программу операторы

R=area(20,30,70);

B=area(2.5,10);

компилятор распознает в них ошибки программиста. В первом операторе то, что указан лишний параметр, а также вещественный результат пытаются записать в целочисленную переменную, а во втором то, что функции передается вещественный параметр.

Если мы запишем функцию area() после главной, то во время обработки оператора A=area(5,b); у компилятора еще не будет информации для проверки правильности вызова.

Так что же, мы не можем поместить функцию вычисления площади после главной функции? Можем, но заголовок функции area()придется написать еще раз, как показано ниже:

**float area(int x, int y);**

**void main(void) {**

**float A,B;**

**int b,R**

**b=4; A=area(5,b); B=area(20,30);**

**}**

**float area(int x, int y){**

**float s;**

**s= x\*y;**

**return s;**

**}**

В первой строке программы записан заголовок float area(int x, int y);. Точка с запятой в конце заголовка говорит о том, что ниже не будет тела функции. Это только предописание, или *прототип* функции, позволяющий проверить правильность ее вызова. Сама функция описана после главной программы.

Прототипы написанных в программе нестандартных функций тоже можно поместить в заголовочный файл, например, myproto.h. Он обычно имеет расширение h и называется (от слова header) заголовочным файлом. В исходный текст программы помещают строку #include “myproto.h”.

Напомним, что перед компиляцией программы ее исходный текст подвергается предварительной обработке – прекомпиляции. Прекомпилятор (он же препроцессор) просматривает текст файла, находит строчки, которые начинаются со слова #include (чтобы их легче находить, в строке перед этим словом нельзя писать ничего, кроме пробелов) и заменяет строчку текстом из указанного в ней заголовочного файла. Файл stdio.h препроцессор должен искать в подкаталоге include среды разработки, а файл myproto.h обычно находится в текущем каталоге – там же, где исходный текст программы. Чтобы указать, где следует искать заголовочный файл, имя stdio.h пишется в угловых скобках, а имя myproto.h в кавычках.

Кроме вставки в текст содержимого включаемых файлов, препроцессор совершает и другие действия. Пусть в тексте программы несколько раз встречается число 32.5, но мы хотим писать его символическое обозначение, например, ABC. Для этого достаточно вставить в текст строку

**#define ABC 32.5**

Препроцессор просмотрит весь следующий за ней текст и везде, где встретится имя ABC заменит его на 32.5.

Рассмотрим в качестве примера следующую задачу:

*Ввести с клавиатуры радиус круга и вывести на экран его площадь. Вычисление площади оформить в виде функции*.

Для решения задачи создадим заголовочный файл round.h следующего содержания:

**#define Pi 3.141592653**

**float Round(int);**

Тогда текст программы будет иметь следующий вид:

**#include <stdio.h>**

**#include “round.h”**

**int r;**

**float S;**

**void main(void)**

**{**

**printf("Введите радиус ");**

**scanf("%d",&r);**

**S=Round(r);**

**printf("Радиус %d \n площадь %g \n", r, S);**

**}**

**float Round(int p)**

**{ return Pi\*p\*p;**

**}**

Строка #include “round.h” включит в текст этой программы прототип функции и константу Pi.

*Замечания*

1. В настоящее время различные фирмы предлагают разные версии компиляторов для языка C++. Для проверки приводимых примеров в среде MS DOS мы будем использовать компилятор фирмы Borland а для среды Windows – Visual С++ фирмы Microsoft. Создатели языков высокого уровня стремятся к тому, чтобы написанная на данном языке программа одинаково выполнялась под управлением разных операционных систем и на компьютерах с различной архитектурой. Но идеально переносимого (мобильного) текста программ обычно не получается. Например, при использовании компилятора Borland C++ объявлять константу Pi нет необходимости. В файле math.h значения ,/2,1/ и прочие определены следующим образом:

#define M\_PI 3.14159265358979323846

#define M\_PI\_2 1.57079632679489661923

#define M\_PI\_4 0.785398163397448309616

#define M\_1\_PI 0.318309886183790671538

#define M\_2\_PI 0.636619772367581343076

#define M\_1\_SQRTPI 0.564189583547756286948

#define M\_2\_SQRTPI 1.12837916709551257390.

Правда, имена выбраны таким образом, что не каждый найдет эти константы. Но в файле math.h, который использует Visual C++, таких констант нет. В разных версиях могут отличаться не только содержимое, но и состав заголовочных файлов. Например, файл mem.h есть в Borland C++, и отсутствует в Visual C++. Но прототипы функций, описанные в mem.h, можно найти в файле memory.h среды Visual C++.

2. В языке Паскаль можно было вставить описание одной функции внутрь другой. В Си все функции равноправны, вложенные функции недопустимы.

3. В Паскале и многих других языках предусмотрена возможность создания функций и процедур. В языке Си есть только функции. Аналогом процедуры языка Паскаль в Си является функция, не возвращающая никакого значения. Допустим, нам очень часто приходится выводить на экран предложение: ВВЕДИТЕ ЧИСЛО и мы хотим оформить его в виде функции, вызов которой должен иметь вид In();. Следует отметить, что язык Си рассматривает круглые скобки как операцию вызова функции. Поэтому даже при отсутствии параметров оператор вызова функции записывается с круглыми скобками. Реализация такой функции будет иметь вид:

**void In(void)**

**{printf(“ВВЕДИТЕ ЧИСЛО”);**

**return;**

**}**

Словом void, означающее пустой, не имеющий никакого значения тип данных, здесь указано, что функция In не получает параметров и не возвращает значения. Строчку return в этой функции можно было не писать – функция завершится, когда вычислительный процесс дойдет до закрывающей фигурной скобки.

Лекция 2. Технология разработки программ

**1. Создание программ для выполнения в среде MS DOS**

Рассмотрим минимальные сведения о среде разработки фирмы Borland, необходимые для построения программы и ее отладки. Пусть на компьютере уже установлено необходимое программное обеспечение в каталоге BC. Обычно активизация среды разработки выполняется запуском находящейся в подкаталоге Bin этого каталога программы BC.exe. При этом на экран выводится главное окно среды, показанное на рис. 1.

Рис. 1 - Главное окно интегрированной среды Borland C

Верхняя строка окна занята основным меню, выбор в нем каждого из пунктов File, Edit, Search и др. приводит к появлению подменю, пункты которого позволяют вызывать выполнение группы близких по назначению операций. На рисунке отображена ситуация, когда вызвано подменю ведения программных проектов. (Далее, для указания выбранной операции будем указывать пункт основного меню и пункт подменю, разделяя их знаком /. Например, Project/Open project – создать новый проект или открыть существующий.

Большие программные системы обычно состоят из нескольких файлов – исходных модулей. Каждый исходный модуль компилируется отдельно, а потом полученные объектные модули объединяются компоновщиком в один исполняемый модуль.

Для того, чтобы объединить несколько файлов в единую программу, в среде Borland С++ создается проект - отдельный файл с расширением PRJ. Чтобы открыть существующий проект или создать новый следует выбрать пункт меню *Project*, а в нем подпункт *Open Project*.

Проект содержит список файлов, которые надо объединить в одну программу, и другую информацию, например, список окон, открываемых при вызове проекта. Если открыть проект Max\_Arr.prj, откроются сразу два окна, показанные на рис. 1.

В меню есть пункты *Project/Add Item и Project/Delete* Item при помощи которых можно добавить или удалить файл из проекта.

(Замечание – я давно не работал в DOS, открыл подменю проект, а пункты в нем серые, неактивные, я не могу добавить в проект еще один файл. Чтобы пользоваться этими пунктами, должно быть открыто показанное в нижней части рисунка окно проекта со списком входящих в него файлов. Если на экране этого окна нет, оно открывается выбором в меню пункта *Windows/Project*).

Для создания проекта разделим рассмотренную выше программу на три файла:

в заголовочный файл iRound.h поместим определение числа π и предописание функции вычисления площади круга:

**#define Pi 3.141592653**

**double Round(int);**

В файл fRound.cpp выделим исходный текст функции вычисления площади:

**#include "iRound.h"**

**double S;**

**double Round(int p)**

**{**

**return Pi\*p\*p;**

**}**

Главна функция будет находиться в файле mRound.cpp:

**#include <stdio.h>**

**#include "Round.h"**

**int r;**

**double S;**

**void main(void)**

**{printf("Введите радиус ");**

**scanf("%d",&r);**

**S=Round(r);**

**printf("Радиус %d \n площадь %g \n", r, S);**

**getchar();**

**getchar();**

**}**

Для создания проекта, включающего файлы mRound.cpp и fRound.cpp необходимо выполнить следующую последовательность действий:

подготовить показанные выше файлы с исходными модулями проекта. Для этого можно использовать любой текстовый редактор, но удобно пользоваться редактором среды разработки (Пунктом File/New создать пустой файл и ввести текст файла, пользуясь при необходимости копирования блоков текста других операций редактирования возможностями подменю Edit);

выбором пункта *Project*/*Open Project* создать новый проект (для этого в открывшемся диалоговом окне следует ввести имя нового проекта);

при помощи пункта *Project/Add Item* включить в проектмодули fRound.cpp и

mRound.cpp (заголовочный файл включать необязательно, препроцессор будет искать его в текущем каталоге даже при отсутствии в списке модулейпроекта);

используя пункт Compile/Make получить исполняемый модуль программы. Вместо обращения к пунктам меню можно использовать быстрый вызов функций среды разработки при помощи “горячих клавиш”. Если проект открыт, то по нажатию клавиши F9 модули проекта файлы скомпилируются (на диск запишутся результаты компиляции - объектные файлы mRound.obj и fRound.obj). Далее эти файлы скомпонуются в один exe-файл (исполняемый модуль).

выбором пункта Run/Run программа запускается на выполнение. Если исполняемый модуль надо скомпоновать и сразу же выполнить, нажимают сочетание клавиш Ctrl/F9.

Использование проекта значительно облегчает разработку больших многомодульных программных систем. В частности, если после выполнения полного цикла разработки программы от подготовки исходных модулей до запуска готовой программы, один из исходных модулей. корректируется (выявлены ошибки или изменились требования к реализованным в нем функциям), будет обеспечена минимизация затрат ресурсов компьютера на обновление программы. Интегрированная среда разработки сравнивает даты создания исходных и объектных модулей. При создании исполняемого модуля заново компилируются только те файлы проекта, у которых дата создания исходного модуля позже даты создания объектного объектного.

Кроме того, в проект можно включать подготовленные отдельно объектные модулю (без включения исходных). Отдельные модули проекта могут быть написаны на языке ассемблера (при этом в составе среды должен быть tasm.exe и необходимо правильно настроить пункты меню Transfert).

**2. Создание программ для выполнения в среде Windows**

Начинающему программисту сложно сочетать освоение нового языка и технологии разработки приложений для операционной системы Windows. В связи с тем, что Windows – система многозадачная и многооконная, она использует свои процедуры ввода-вывода вместо стандартных, определенных синтаксисом языка программирования, будь это Паскаль, С++, Бейсик и пр. Применение используемых в ней средств построения меню, диалоговых окон и других элементов человеко-машинного интерфейса требует, чтобы разработчик уже имел определенную квалификацию.

Чтобы изучать основы программирования лучше создавать консольные приложения Windows. Консольное приложение, это программа, осуществляющая вывод в текстовое окно, аналогичное экрану дисплея при работе с MS DOS, работа которой возможна без применения специфических средств взаимодействия с операционной системой.

Покажем последовательность действий для создания консольного приложения в среде Visual С++ версии 6.

При запуске среды разработки на экран выводится главное окно среды (рис 2) с пунктами меню, позволяющими, выполнять основные действия по разработке приложения:

читать, редактировать и записывать на диск текстовые файлы;

создавать проекты из нескольких исходных модулей;

компилировать файлы проекта и создавать исполняемый модуль;

выполнять программы в автоматическом и отладочном режимах.

Рис. 2 - Окно среды Visual C++

Пункты меню, которые выполняются наиболее часто, дублируются пиктограммами. Обозначения пиктограмм верхнего ряда стандартны для офисных и других приложений фирмы Microsoft.

Во втором ряду показаны пиктограммы, щелчком на которых можно выполнить следующие действия (перечисляем слева направо):

компиляцию исходного текста и получение объектного модуля;

компоновку исполняемого модуля из объектных;

прерывание и отмену выполняемой операции (компиляции, компоновки или отладки программы);

выполнение исполняемого модуля в автоматическом режиме – восклицательный знак;

выполнение исполняемого модуля в отладочном режиме (с остановкой на контрольных точках);

установку контрольных точек – значок в виде ладони.

Программист может перемещать панели инструментов с пиктограммами по экрану, изменять по своему вкусу состав пиктограмм, вынесенных на панель инструментов. Эти операции выполняются выбором пункта Customize подменю Tools.

При выборе пункта меню File появляется показанное на этом же рисунке подменю, содержащее пункты New, Open, Close, Open WorkSpaсe и др. Для создание нового проекта и новой рабочей области (подкаталога с файлами проекта) необходимо выбрать пункт New, при этом появится диалоговое окно New.

Как видно из рис 2, в этом окне можно выбрать одну из нескольких вкладок. В частности, для создания и включения в проект новых файлов, надо выбрать вкладку Files. На рисунке показана ситуация, когда пользователь создает новую рабочую область и проект. Имя Console0 нового проекта и подкаталога для его размещения записано в поле Project Name, путь к создаваемому подкаталогу задан в поле Location. В основном поле вкладки цветом выделен тип создаваемого проекта. Выделена строкаWin32 Console Application, при выборе которой создается проект приложения, работающего в текстовом окне, аналогичном экрану MS DOS, и использующему для вывода на экран и ввода с клавиатуры стандартные операторы языка С++. (Для создания оконного приложения следовало выбрать пункт Win32 Application, а один из двух последних пунктов выбирается при создании библиотек). После нажатия не поместившейся на рисунке кнопки OK и ответа на два дополнительных вопроса будет создан пустой проект, в который надо включить файлы с исходными текстами программ на языке С++.

После этого можно заново выбрать пункты меню File/New но в появившемся окне New уже выбрать вкладку File. Из множества предложенных вариантов, часть из которых показана на рис. 3 нам потребуется указывать заголовочные файлы *C/C++ Header File* или файлы с исходным текстом *С++ Source File*. В данном случае выбран исходный модуль и задано его имя main. По нажатию кнопки OK модуль будет не только создан, но и (как показывает флажок Add Project) включен в состав проекта.

Рис. 3 - Типы файлов

После создания пустого файла выбранного типа, в левом поле окна среды разработки (см. рис. 4), отображающем состав проекта, появится имя включенного в проект файла main.cpp, а в правом поле можно набирать и редактировать исходный текст модуля.

Рис. 4 - Отображение состава проекта

На рис 1.4 показано то состояние окна, когда пользователь уже набрал программу вывода на экран 25 строк текста и даже выполнил компиляцию модуля. Поэтому в нижнем поле мы видим сообщение о том, что в результате компиляции ошибок не обнаружено.

Далее можно выполнить компоновку исполняемого модуля или, щелчком по пиктограмме с восклицательным знаком, одновременно компоновку и пуск программы.

На рисунке 4 показана ситуация, когда пользователь навел на восклицательный знак курсор мыши, пиктограмма выделилась в виде рельефной кнопки и под ней появилась подсказка *Execute Program*, объясняющая назначение кнопки. Если щелчком по этой кнопке запустить выполнение программы в автоматическом режиме, мы получим результат ее работы в консольном окне, показанный на рис 5.

Это текстовое окно, имитирующее работу программы с символьным дисплеем в среде MS DOS.

Рис. 5 - Вид консольного окна

Следует заметить, что при выводе на экран строки текста, например, операторами

**char s=”Пример текста”;**

**printf (s);**

мы увидим непонятные символы. Причина в том, что для полноты имитации текстового режима оператор printf(s) использует принятую в MS DOS кодовую страницу CP866. Это можно проверить, вызвав стандартную функцию Res = GetConsoleOutputCP() (о кодовых страницах см. лекции по Паскалю). Но поскольку компилятор при кодировании строки применяет страницу 1251 русифицированной ОС Windows, на экран невозможно вывести русский текст. Чтобы вывести русские буквы абвгд, строку придется закодировать так:

char s[]={0xa0,0xa1,0xa2,0xa3,0xa4,10,0};

Обратите внимание, на подсказка на рис. 1.4. Она сообщает, что действия, выполняемые щелчком по кнопке с восклицательным знаком, продублированы горячими клавишами Ctrl/F5. Аналогичные подсказки закреплены за каждой световой кнопкой, со средой разработки поставляется также обширная справочная документация, в результате процесс разработки и отладки программ осваивается значительно быстрее при практической работе за компьютером, чем при чтении учебника. Поэтому здесь отметим только, что пошаговое выполнение программы без захода в процедуры выполняется по нажатию функциональной клавиши F10, с заходом в процедуры F11, а автоматическое выполнение до строки, в которой установлен курсор, запускается сочетанием клавиш Ctrl/F10.

Лекция 3. Знакомство с операторами языка

**1. Условный оператор**

Мы помним, что операторы программы задают последовательность действий, которую надо выполнить с объявленными в ней переменными. Часто она заранее точно неизвестна и определяется уже во время работы по результатам предыдущих вычислений.

Пусть, например, *необходимо вычислить подоходный налог с заработка, размер которого хранится в переменной Z. Сотрудник имеет право на льготы по налогу, суммарный размер которых хранится в переменной L.*

*Если заработок больше суммы льгот (то есть Z>L), налог берется в размере N=13\*(Z-L)/100. Когда сумма разрешенных льгот больше, чем заработок, применение этого выражения даст отрицательный* Поэтому они могут участвожения *результат. Это означает, что налог не взимается, то есть, при Z<L, значение N=0.*

Заметим, что в С++ знаки <, меньше и >,больше - это операции, дающие целочисленный результат.. Если условие, заданное операцией сравнения, выполняется, результат операции равен единице, если не выполняется – нулю. То есть, при вычислении выражения Z>L получим единицу, если Z больше, чем L. В С++ нет логического типа данных, поэтому вполне допустимо написать выражение (Z>5)+(Z>L). В зависимости от значений переменных вычисление этого выражения даст значения 0, 1 или 2.

Рассмотрим программу, которая вводит значения Z и L с клавиатуры и вычисляет сумму налога условным оператором:

**#include <stdio.h>**

**int Z,L,N;**

**void main(void)**

**{**

**printf(“\n Введите сумму заработка и размер льгот”);**

**scanf("%d%d ",&Z,&L);**

**if(L<Z) N=13\*(Z-L)/100;**

**else N=0;**

**printf(“ Налог равен %d руб.”, N);**

**getchar();**

**getchar();**

**}**

Для сравнения посмотрите, как тот же ввод с клавиатуры выполнить с использованием потока:

**cin>>Z>>L;**

Выполнение оператора if(L<Z) N=13\*(Z-L)/100; else N=0;

заключается в следующем:

– вычисляется выражение, записанное в скобках;

– если результат не равен нулю, выполняется оператор после скобок, если равен нулю – оператор, записанный после else.

*Замечания*

1. Выше мы использовали запись условного оператора в виде

**if(<**выражение>**) <**оператор1> **else** <оператор2>.

Можно использовать также краткую форму условного оператора **if(<**выражение>**) <**оператор>. Здесь отсутствует ветвь **else**, поэтому при равенстве выражения нулю не выполняется никаких действий.

2. В языке Паскаль точка с запятой является разделителем между операторами, а в С++ она является неотъемлемой частью оператора. Поэтому в Паскале знак ‘**;’** перед else не ставится (слово else само по себе является хорошим разделителем), а в данном примере точка с запятой перед else указана, так как она часть оператора N=13\*(Z-L)/100;.

3.В языке С++ строчная и соответствующая ей прописная буквы – это разные символы. Поэтому в рассмотренном примере нельзя объявить переменные int Z, L, N;, а потом записать оператор n=13\*(z-l)/100;.

4. Мы помним, что происходит при завершении программы вычисления площади, приведенной в параграфе 1. Завершаясь, главная функция main() возвращает управление тому, кто ее запускал. Если программу запускали из оболочки типа Norton Commander, на экране очень быстро промелькнут выводимые программой числа и мы опять увидим окна Norton Commander, если программа запускается из среды разработки, появится окно редактора с исходным текстом. Чтобы пользователь видел на экране результат, в конце программы вставляется дополнительный оператор ввода символа. При выполнении этого оператора программа не завершается, пока пользователь не нажмет кнопку клавиатуры и на экране видны результаты вычислений.

Среди функций стандартного ввода-вывода есть функция **getchar()**;, которая вводит один символ. В предыдущем примере, чтобы приостановить программу она записана дважды. Один оператор не приостановит программу по следующей причине:

В начале программы функция **scanf("%d%d ",&Z,&L);** вводит с клавиатуры два числа Z и N. Ее особенностью является то, что при вводе числа функция отбрасывает все предшествующие первой цифре коды пробела, табуляции, перевода строки и возврата каретки, потом вводит цифры числа, пока не будет нажата клавиша Enter. Код этой клавиши функция scanf() использует как признак конца ввода числа, но оставляет во входном потоке. Это значит, что код Enter функция как бы и не вводила – он поступит следующему оператору ввода данных.

Следующим оператором ввода является вызов getchar(). Он получит и введет код Enter, оставшийся не обработанным функцией scanf(), и только второй оператор getchar() будет ожидать ввода человеком следующего символа.

Два оператора getchar() не украшают нашу программу, кроме того, как любая функция стандартного ввода, getchar() отображает символ на экране, и после нажатия клавиши, например, буквы А, ждет еще и нажатия клавиши Enter, как признака завершения ввода. Поэтому в дальнейшем мы будем пользоваться функцией консольного ввода символа getch().

Покажем применение консольного ввода на примере применения краткой формы условного оператора для определения наибольшего из трех чисел. Ниже представлена программа, которая вводит с клавиатуры три числа a, b, c, большее из них записывает в переменную max, после чего ожидает нажатия любой клавиши:

**#include <stdio.h>**

**#include <conio.h>**

**int a,b,c;**

**void main(void)**

**{**

**clrscr();**

**scanf("%d%d%d",&a,&b,&c);**

**int max=a;**

**if(a<b) max=b;**

**if(c>max) max =c;**

**printf("%d",max);**

**getch ();**

**}**

В среде Borland C оператор #include <conio.h> вставляет в исходный текст программы прототипы всех функций консольного ввода-вывода. Мы с ними знакомы – это те функции, которые в Паскале реализованы модулем Crt. Надо только записывать их имена строчными буквами: clrscr() – очистить экран, gotoxy(x,y) –переместить курсор и т.д.

Только паскалевским функциям проверки нажатия клавиши KeyPressed и ввода символа ReadKey в С++ даны другие имена – здесь они называются kbhit() и getch(). Функция getch() вводит символ без эхо-отображения и не ожидает нажатия Enter.

Не совсем понятно, почему ее достаточно вызвать один раз. Кратко (не требуя полного понимания) на это можно пока ответить так. Входной поток стандартного ввода-вывода, это массив байтов, в котором операционная система накапливает введенные символы, не передавая их операторам ввода из входного потока до нажатия Enter. Консольные же функции читают символы не из входного потока. Они опрашивают непосредственно клавиатуру. Поэтому давно нажатая клавиша Enter не оказывает никакого действия на работу функции getch(). Функция getch() ожидает нажатия клавиши, а код Enter так и остается непрочитанным из буферной области оперативной памяти, которую мы называем входным потоком.

**2. Оператор цикла в форме for**

Объявим целочисленную переменную I и рассмотрим оператор цикла на примере вывода на экран десяти чисел, кратных трем:

**int I;**

**for(I=1;I<=10;I=I+1)**

**printf(“%d”,3\*I);**

Как и в Паскале, оператор состоит из:

– заголовка цикла for(I=1;I<=10;I=I+1)

– многократно повторяемого тела цикла – оператора printf(“%d”,3\*I).

Заголовок состоит из служебного слова for и трех взятых в скобки выражений, которые определяют, сколько раз будет повторяться тело цикла, и как будет изменяться параметр цикла I.

Первое выражение I=1 выполнится один раз (до первого выполнения тела цикла) и определит начальное значение параметра цикла.

Перед каждым выполнением тела цикла вычисляется второе выражение (обычно там помещается проверка какого-то условия) и если условие не выполняется (значение выражения равно нулю), цикл завершается. Если условие выполнено (результат вычисления выражения не равен нулю), выполняется тело цикла.

После каждого выполнения тела цикла вычисляется третье выражение, которое обычно изменяет значение параметра цикла.

Как видим, в данном случае параметр I будет изменяться от одного до 10, при этом выводимое на экран значение 3\*I, будет кратно трем. Результаты экзаменов показывают, что иногда у студентов вызывают затруднения даже такие простые задачи. Поэтому рассмотрим несколько похожих примеров.

1. Те же числа, но в обратном порядке можно вывести оператором

**for(I=10;I;I=I-1)**

**printf(“%d”,3\*I);**

Оператор интересен тем, что вторым выражением является не проверка условия, а просто переменная I. Когда она станет равной нулю, цикл завершится.

2. Язык С++ предоставляет большую по сравнению с Паскалем свободу при реализации заголовка цикла. В частности, параметр цикла может быть вещественным числом, разрешено пропускать в заголовке цикла любое из выражений (а может, это называется записывать пустое выражение, потому что символы; пишутся обязательно). Следующий фрагмент показывает решение задачи с пропущенным первым выражением:

**float I=1;**

**for(;I<=10;I=I+1)**

**printf(“%f”,3\*I);.**

3. Можно пропустить также третье выражение и изменять переменную I в теле цикла:

**float I=1;**

**for(;I<=10;)**

**{I=I+1;**

**printf(“%d”,3\*I);**

**}.**

Как и в Паскале, циклически повторяется только один записанный после заголовка оператор. Если надо повторять несколько операторов, они объединяются фигурными скобками в блок. В данном случае

{

I=I+1;

Printf (“%d”,3\*I);

} - это блок операторов. Блок языка С отличается от составного оператора Паскаля тем, что в блоке после открывающей фигурной скобки можно размещать объявления переменных. Такие переменные остаются видимыми только в пределах того блока, где они объявлены (с понятиями области видимости и времени жизни переменных мы знакомились при изучении Паскаля).

В отличие от С, язык С++ разрешает объявлять переменные в любом месте, кроме управляющих конструкций операторов. Например, нельзя объявлять переменную в условном операторе (в круглых скобках после if).

Но параметр цикла можно объявлять непосредственно в заголовке (хотя заголовок – это управляющая конструкция цикла):

for(int i=0; i<10;i=i+1) printf(“%d”,I);

При этом переменная i будет доступна не только в этом цикле, но и во всех последующих операторах до конца блока.

4. Если опустить в заголовке все три выражения, результат вычисления второго пустого выражения считается ненулевым и мы получаем бесконечный цикл. В этом случае для решения задачи можно применить оператор принудительного завершения оператора цикла - break:

**int I=1;**

**for(;;) //**Такой заголовок приводит к бесконечному повторению тела цикла.

**{ if (I>10) break; //**Выполнение оператора break вызывает завершение цикла

// и переход к следующему после цикла оператору.

**I=I+1;**

**printf(“%d”,3\*I);**

**}.**

Знакомясь с оператором break языка Паскаль, мы отмечали, что его не было в авторской версии языка, его заимствовали из языка Си.

5. При решении данной задачи студенты часто пытаются в теле цикла дополнительно изменять значения I, чтобы они были кратны трем. В теле цикла при этом выводится значение I а не 3\*I, например, так:

**int I;**

**for(I=1;I<=10;I=I+1)**

**{ printf(“%d”, I);**

**I=I+3;}**

Это неправильное решение, так как не учитывается что в каждом проходе к I, будет прибавлять единицу заголовок цикла, и еще три прибавится в теле цикла. Проверьте самостоятельно сколько и каких чисел выведет на экран приведенный выше оператор.

Если не хочется применять длинную операцию умножения, цикл следует записать так: **for(I=3;I<=30;I=I+3) printf(“%d”, I);.**

6. Выведем на экран первые десять элементов последовательности

½,1/4, 1/6, 1/8...

Типичная ошибка при решении этой задачи приводит к выводу на экран значения 2-n. Элементы данной последовательности вычисляются оператором

**float I**

**for(I=1;I<10;I=I+1) printf(“%f”, 1/(2\*I));**

или, без использования умножения:

**float I, s=0;**

**for(I=1;I<10;I=I+1)**

**{ s=s+2;**

**printf(“%f”, 1/s);**

**}.**

Чтобы выводить на экран значения 2-n, следует делить на 2 предыдущее вычисленное значение степени:

**float I, s=1;**

**for(I=1;I<10;I=I+1)**

**{ s=s/2;**

**printf(“%f”, s);**

**}**

**Несколько замечаний по форматированию вывода**

Предыдущий фрагмент выводит вычисленные значения степени одной строкой.

Запишем программу вывода значений 2-n полностью и будем выводить показатель и три варианта значения степени, использующие различные спецификации вывода

**#include <stdio.h>**

**#include <conio.h>**

**int i;**

**float s=1;**

**void main(void)**

**{ clrscr();**

**for(i=1;i<20;i++)**

**{**

**printf("\n -%d cтепень \**

**равна %06.2f %6.2e %-6.3g ",i-1,s,s,s);**

**s=s/2;**

**}**

**getch();**

**}**

При пуске программа выведет на экран следующий текст:

-0 cтепень равна 001.00 1.00e+00 1

-1 cтепень равна 000.50 5.00e-01 0.5

-2 cтепень равна 000.25 2.50e-01 0.25

-3 cтепень равна 000.12 1.25e-01 0.125

-4 cтепень равна 000.06 6.25e-02 0.0625

-5 cтепень равна 000.03 3.12e-02 0.0312

-6 cтепень равна 000.02 1.56e-02 0.0156

-7 cтепень равна 000.01 7.81e-03 0.00781

-8 cтепень равна 000.00 3.91e-03 0.00391

-9 cтепень равна 000.00 1.95e-03 0.00195

-10 cтепень равна 000.00 9.77e-04 0.000977

-11 cтепень равна 000.00 4.88e-04 0.000488

12 cтепень равна 000.00 2.44e-04 0.000244

-13 cтепень равна 000.00 1.22e-04 0.000122

-14 cтепень равна 000.00 6.10e-05 6.1e-05

-15 cтепень равна 000.00 3.05e-05 3.05e-05

Здесь каждое вычисленное значение выводится с новой строки. Мы знаем, что для перевода курсора на следующую строку на экран следует вывести символ с кодом 10, а для перемещения в начало строки – символ с кодом 13. Но у этих символов нет изображения, которое можно вставить текстовым редактором в выводимую строку. Если мы напишем printf(“10строка”), то в начале строки будут два символа ‘1’,’0’с кодами 49 и 48, а не символ перевода строки с кодом 10. Для включения в строку специальных символов в С++ используют букву, перед которой записан знак \. Например, \n понимается компилятором, как символ с кодом 10, \r – символ с кодом 13, \а – символ с кодом 7 (звонок) и т.д. Чтобы включить в строку знак \, его приходится записать дважды \\. Из-за этого путь к файлу filename.dat, который находится в каталоге ABC, на языках С, С++ приходилось задавать так:

C:\\ABC\\filename.dat.

Таким образом, знак \n в программе вывода значений степени – это символ перевода строки.

Оператор вывода printf("\n -%d cтепень равна %06.2f %6.2e %-6.3g ",i-1,s,s,s); содержит очень много букв, поэтому он записан в две строки.

printf("\n -%d cтепень

равна %06.2f %6.2e %-6.3g ",i-1,s,s,s);

Для перехода в редакторе текста на новую строку мы нажимаем клавишу Enter, у нее тоже есть код, который вставляется в строку после слова «степень». Но этого кода не должно быть в строке, выводимой во время работы программы. Чтобы компилятор выбросил из строки код клавиши Enter, мы вводим \, а потом нажимаем Enter. (То есть, сочетание \<Enter > включает код перевода строки в исходный текст, но подавляет его в компилированной строке).

Вычисленная степень s выводится на экран три раза. Все три спецификации %f %g предназначены для вывода вещественных чисел:

**%f** предписывает выводить число без множителя 10n с шестью знаками после десятичной точки, последняя цифра округляется;

**%e** предписывает всегда выводить число в экспоненциальной форме (c множителем 10n);

**%g** сохраняет не больше шести значащих цифр результата (в %f шесть цифр только в дробной части, а здесь всего шесть, включая целую и дробную часть, но значащих цифр). Для очень маленького числа запишутся нули, а потом 6 цифр. Кроме того, в данном формате автоматически из двух вариантов представления числа – экспоненциального и с десятичной точкой, автоматически выбирается тот, который запишется короче.

В нашем примере после знака % перед буквой записаны цифры:

**%6.2f** означает, что на число (включая точку) отведено шесть позиций, из них две после точки. Если перед первым числом написать 0, %06.2f, в незаполненные старшие разряды запишутся нули (как в нашем примере);

**%6.3g** означает, что числу отводится не менее шести знакомест, но от числа сохраняется не более трех значащих цифр. На примере распечатки мы не видим, что числу отведено шесть позиций, потому что столбец последний и число выравнивается не по правому, а по левому краю области печати. Указание выравнивать число по левому краю дано знаком минус в спецификации %-6.3g.

Лекция 4. Работа с массивами

**1. Одномерные массивы**

Массив, это совокупность однотипных элементов, объединенных общим именем.

Например, десять целых чисел, объединенных именем Mas, объявляются следующим образом:

**int Mas[10];**То же, что на Паскале Mas: array [0..9] of integer;.

В объявлении записывают тип элементов, имя массива и, в квадратных скобках, количество элементов.

Элементами массивов могут быть не только числа. Можно например, объявить массив, состоящий из массивов. Нельзя объявлять массивы, элементами которых являются функции или данные типа void.

Массив можно передать функции как параметр, но функция не может возвращать массив в качестве результата своей работы.

Объявление массива можно совместить с его инициализацией, перечислив значения элементов в фигурных скобках:

**int A[5] ={2,-3, 0,0,7};**

В операторной части программы при обращении к отдельным элементам указывают имя массива и номер элемента в квадратных скобках. Элементы массива нумеруются начиная с нуля, т.е. оператор

**printf(“%d”,A[0]);**

выведет на экран первый элемент, число 2, а оператор

**printf(“%d”,A[4]);**

выведет последний элемент, число 7. Элемента A[5] в массиве A нет.

При инициализации массивов язык Си позволяет большую свободу, чем Паскаль. В частности, не обязательно перечислять значения всех элементов. В объявлении

**float M1[10]={4, 2.5, 0.3};** семь последних элементов автоматически заполнятся нулями. Самостоятельно проверьте, можно ли пропускать элементы, если они не последние, например, разделяя отсутствующие элементы запятыми:

float M2[10]={4, 2.5,0.3,,0,,,,20.3};.

Если при инициализации перечислены все элементы - можно не указывать размер массива: **ar[]={2,7,9,3,1};**. Заметим, что размер массива можно не указывать также если массив объявлен, как параметр функции или если объявляется ссылка на массив, определенный в другом файле. Пусть, например, наш проект состоит из двух файлов (A.cpp, B.cpp). Если A.cpp мы объявили массив float M1[10], то в B.cpp для работы с этим массивом можно записать строку

extern float M1[];,

которая указывает, что массив внешний – объявление массива сделано в другом файле. При необходимости можно указать и размер массива extern float M1[10], но инициализацию при ссылке на внешний массив (как и на любую внешнюю переменную) повторять нельзя.

*Рассмотрим в качестве примера решение следующей задачи:*

*– объявить массив из пяти целых чисел;*

*– заполнить элементы массива данными, вводимыми с клавиатуры;*

*–вывести элементы массива на экран;*

*найти сумму положительных элементов и вывести ее на экран.*

**#include <stdio.h>**

**#include <conio.h>**

**int ar[5];**

**void main(void)**

**{ clrscr();**

**printf("Enter five numbers");**//Это я по словарю привыкаю к английским словам

**for(int i=0;i<5;i++) scanf("%d",&ar[i]); //**Объявленная здесь переменная i видна и дальше.

**float Sum=ar[0];**

**for(i=1;i<5;i++) if(ar[i]>0) Sum=Sum+ ar[i];**

**for(i=0;i<5;i=i+1)printf(“%5d”, ar[I]);**

**printf("The sum is %f",Sum);**

**getch ();**

**}**

Решим задачу поиска элементов с максимальным значением в массивах, состоящих из целых чисел. Для этого реализуем функцию поиска максимального элемента в виде отдельного файла

**int MaxArr(int Mas[],int R=10)** //Файл Max\_Arr.cpp

**{ int Max=Mas[0];**

**for(int i=1;i<R;i=i+1) if(Mas[i]>Max)Max=Mas[i];**

**return Max;**

**}**

Первый параметр функции MaxArr(),– это массив из целых чисел, а второй – количество элементов массива. Размерность массива не указана, поэтому функция может находить максимальный элемент в любом массиве, состоящем из целых чисел. Язык C++ позволяет при описании функции указывать после параметра его значение по умолчанию (параметру R по умолчанию присвоено значение 10). В этом случае при вызове функции можно не указывать один или несколько последних параметров, имеющих заданные по умолчанию значения.

Ниже приведена программа, использующая данную функцию для вывода на экран максимальных элементов двух массивов разного размера.

**#include <stdio.h>**

**#include <conio.h>**

**int MaxArr(int Mas[],int R=10);** Это шаблон заголовка. В нем можно опускать имена формальных параметров и писать только их типы, например,

int MaxArr(int[],int=10);

**void main(void)**

**{ clrscr();**

**int M1[5]={-3,5,0,15,6};**

**int M2[10]={13,25,0,15,-36};**

**int Max2=MaxArr(M2);** МассивM2 состоит из 10 элементов, поэтому можно передавать только первый параметр.

**printf("\n В первом массиве %d \**

**Во втором массиве %d", MaxArr(M1,5), Max2); getch();**

**}**

Сравните с языком Паскаль – там при описании формального параметра типа массив, требовалось указывать имя параметра и имя предварительно описанного типа передаваемого массива. Из-за этого на Паскале для работы с массивами разных размеров требовалось иметь отдельные функции. Но зато на Си программист может ошибиться и задать больше или меньше элементов, чем есть в массиве. Заметим, что можно указать в заголовке функции размерность массива:

int MaxArr(int Mas[10],int R=10);

Но даже и в этом случае компилятор не производит контроль соответствия размера указанного в заголовке размеру реально передаваемого массива – предоставляя программисту большие возможности, язык С++ возлагает на него большую ответственность за работу программы.

Замечание. Если производится обращение к переменной, объявленной в другом модуле проекта, надо указать, что она внешняя, например, extern int M. Имена функций видны везде, нужно указывать лишь прототип функции.

**2. Многомерные массивы**

Как и в языке Паскаль, в С++и многомерные массивы конструируют, объявляя массив, элементы которого тоже массивы. Так:

– одномерный массив int A[10]; - это набор из 10 целых чисел;

– двумерный массив int A2[10][3]; - это массив из 10 элементов, а каждый элемент A2[i] массив из трех целых чисел;

– int A3[10][3][5]; это массив из 10 элементов, а каждый элемент A3[i] - двумерный массив размером 35;.

Двумерные массивы используются для работы с матрицами и другими прямоугольными таблицами. Для того, чтобы в программе на языке С++ объявить прямоугольную матрицу

t00 t01 t02 t03

T =t10 t11 t13 t12

t20 t21 t22 t23,

надо указать, из элементов какого типа (целых или вещественных) она состоит, дать ей имя, указать сколько в ней строк и столбцов. Если показанная выше таблица содержит вещественные числа, ее объявление будем иметь вид:

double T[3][4];

где 3 –количество строк, а 4 – столбцов. В языке Паскаль для каждого индекса указывался интервальный тип данных, задающий начальное и конечное значения, в С++ строки и столбцы всегда нумеруются с нуля, поэтому пишется только их количество.

Чтобы задать конкретный элемент массива, надо указать номер строки, в которой находится этот элемент, и номер столбца. Так, элемент, который находится во второй строке и третьем столбце надо обозначать T[1][2]. Можно рассуждать и иначе:

– выбираем элемент массива, указав его индекс - T[1];

– T[1] это массив из четырех чисел, выбираем элемент массива T[1], указав его индекс - T[1][2].

В Паскале, чтобы увеличить сходство операторов программы с математической записью элементов матриц, разрешалось перечислять индексы массива через запятую. В С++ это недопустимо. Особенно неприятно, что компилятор, встретив обозначение M [1,2] будет считать, что это M[2] и при синтаксическом контроле может не выдать ошибку.

Объявление двумерного массива также можно совмещать с его инициализацией:

**int Mas[3][4]= {{2, 7, 9,4},**

**{1,3},**

**{3,3,3,3}}**

Правила инициализации вытекают из соответствующих правил для одномерных массивов.

Двумерный массив – это одномерный массив, элементами которого являются строки матрицы. А при инициализации одномерного массива в фигурных скобках (в примере это внешние скобки) перечисляются через запятую значения элементов массива. Но каждый элемент – это тоже массив. Поэтому его значение – последовательность чисел, взятая в фигурные скобки. Как показано во второй строке, можно перечислять не все элементы строки матрицы – недостающие автоматически заполнятся нулями.

В сделанном выше объявлении массива можно не указывать число строк:

int Mas[][4]= {{2,7… и т д.

В памяти элементы массива располагаются по строкам, сначала элементы первой строки, потом второй и т. д. Для многомерных массивов (у которых больше двух индексов) это правило формулируется так:

– при размещении первым записывается в память элемент, у которого все индексы равны нулю;

– далее пробегаем последний (правый) индекс от нуля дот максимального значения;

– потом увеличиваем на единицу предпоследний индекс и заново изменяем последний от нуля до единицы;

– когда предпоследний индекс достигнет максимального значения, увеличиваем третий справа индекс и так далее.

Таким образом, в двумерном массиве целых чисел размером MN элемент с индексами i,j смещен на N\*sizeof(int) i+ sizeof(int)\*j байтов от начала массива.

Учитывая построчное расположение элементов в памяти, в языке разрешено перечислять при инициализации элементы одной строкой. Тот же массив, что показан выше, можно было объявить так:

int Mas[][4]= {2, 7, 9,4, 1,3,0,0, 3,3,3,3};, но оставлять незаполненными элементы второй строки уже нельзя. В этом случае тоже можно не указывать первую размерность. Последняя строка может быть не полной - при объявлении

int Mas[][4]={1,2,3,4, 1,3} компилятор будет отсчитывать по четыре числа в строке и создаст матрицу

1 2 3 4

1 3 0 0.

Вывести элементы двумерного массива на экран можно различными способами.

Удобно организовать цикл по строкам и вложить в него цикл по столбцам

**for(int i=0; i<3;i=i+1)**

**for (j=0;j<4;j=j+1) printf (“%5d”,Mas[i][j]);.**

Можно также учесть, что в массиве 12 чисел и сделать цикл от нуля до 12, но при этом придется использовать новую арифметическую операцию % (получение остатка от деления целых чисел)

**for(int i=0; i<12;i=i+1) printf (“%5d”,Mas[i/4][i%4]);.**

Оба приведенные выше цикла будут печатать элементы массива в одну строку.

Если мы хотим выводить массив в виде прямоугольной таблицы, перед каждой следующей строкой (или после каждой строки) массива надо вывести на экран символ перевода курсора \n., как показано ниже:

**for(int i=0; i<3;i=i+1)**

**{**

**for (j=0;j<4;j=j+1) printf (“%5d”,Mas[i][j]);.**

**printf(“\n”);**

**}**

Объясните, как изменится работа программы, если в этом фрагменте удалить фигурные скобки.

Помню, в школе нас учили решать алгебраические уравнения методом подстановки и методом алгебраического сложения. В ВУЗе в курсе алгебры мы узнаем, что решение систем линейных уравнений школьным методом алгебраического сложения называется методом Гаусса, а также знакомимся с алгоритмами вычисления определителя и обратной матрицы. Они основаны на последовательности шагов, заключающихся в умножении элементов строки на константу и прибавлении к соответствующим элементам другой строки то же матрицы.

Составим, для примера, программу решения систем уравнений методом Гаусса и решим систему:

**1.7x1 + 10.0x2 - 1.3x3 + 2.1x4 = 3.1**

**3.1x1 + 1.7x2 - 2.1x3 + 5.4x4 = 2.1**

**3.3x1 - 7.7x2 + 4.4x3 - 5.1x4 = 1.9**

**10.0x1 - 20.1x2 + 20.4x3 + 1.7x4 = 1.8**

При решении системы все ее числовые коэффициенты можно хранить в массиве, объявленном как **double a[4][5]**. Столбец свободных членов системы – это элементы a[I][4], где I=0,..., 3.

Методом Гаусса основано на последовательности преобразований системы, не изменяющих ее корней.

Если прибавить к левой и правой части уравнения одно и то же число, его корни не изменяются. (Когда x1 x2 … xn – это корни уравнения, его левая и правая части равны. Поэтому к одном уравнению можно, не изменяя корней, прибавить другое, умноженное на числовой коэффициент).

В методе Гаусса решение системы из n уравнений получается за n шагов.

На шаге номер k:

уравнение номер k делится на коэффициент a[k][k] – диагональный элемент матрицы становится равным единице;

потом (для всех i ≠ k) к уравнению номер i прибавляется уравнение номер k умноженное на минус a[i][k]. В результате в столбце k все коэффициенты, кроме расположенного на диагонали, станут равными нулю (для ручного счета указанные на данном шаге действия выполняются только для значений i>k, но нам удобнее заменить нулями все коэффициенты столбца, кроме одного).

После того, как циклом, изменяющим k от нуля до n-1 выполнится n описанных выше шагов, в каждой строке коэффициенты при всех неизвестных, кроме одного, будут равны нулю – это и есть решение.

Проделаем описанные преобразования для заданного уравнения.

Разделим все коэффициенты первой строки на a[0][0]=1.7. Первое уравнение приобретет вид:

x1 + 5.882x2 - 0.7647x3 + 1.235x4 = 1.824

Теперь, чтобы в уравнении 2 получить нулевой коэффициент при x1, все элементы первого уравнения умножаем на a[1][0]=3.1 и отнимаем от второго, получим

0x1 - 16.54 x2 + 0.2706 x3 + 1.571 x4 = 3.553

и так далее.

Программа, реализующая данный алгоритм, оказывается заметно короче его описания:

**#include<stdio.h>**

**#include<conio.h>**

**double a[4][5]={{1.7, 10.0, -1.3, 2.1, 3.1},** //Записали систему уравнений

**{1, 1.7, -2.1, 5.4, 2.1},**

**{3.3, -7.7, 4.4, -5.1, 1.9},**

**{10.0,-20.1, 20.4, 1.7, 1.8}};**

**void print (void)** //Вывод таблицы коэффициентов оформили в виде функции.

**{ for(int i=0;i<4;i++){for(int j=0;j<5;j++)**

**printf("%8.4lg ",a[i][j]);printf("\n");}**

**printf("\n");**

**}**

**void main (void)**

**{**

**clrscr();**

**print();** //Вывели исходную таблицу

**for(int k=0;k<4;k++) //**Цикл по числу уравнений

**{ double Kf=a[k][k];**

**for(int j=0;j<5;j++) a[k][j]=a[k][j]/Kf; //**Получаем единицу приxk

**for(int i=0;i<4;i++)**

**{ //**Во всех уравнениях, кроме k-го делаем коэффициент при xk равным нулю.

**Kf=a[i][k];**

**if(i!=k) for(j=0;j<5;j++) a[i][j]=a[i][j]-a[k][j]\*Kf;**

**}**

**print();//**Вывели таблицусо столбцом из нулей.

**}**

**getch();**

**}**

В приведенной программе делитель a[k][k]; предварительно записывается в отдельную переменную: Kf=a[k][k];.

Объясните, почему нельзя отказаться от использования промежуточной переменной Kf, записав вместо оператора a[k][j]=a[k][j]/Kf; оператор a[k][j]=a[k][j]/ a[k][k];.

Далее показаны результаты вывода на экран.

Исходная таблица

1.7 10 -1.3 2.1 3.1

3.1 1.7 -2.1 5.4 2.1

3.3 -7.7 4.4 -5.1 1.9

10 -20.1 20.4 1.7 1.8

Результат обработки при k=0 (сравните с ручным счетом)

1 5.882 -0.7647 1.235 1.824

0 -16.54 0.2706 1.571 -3.553

0 -27.11 6.924 -9.176 -4.118

0 -78.92 28.05 -10.65 -16.44

Результат обработки при k=1

1 0 -0.6684 1.794 0.5596

-0 1 -0.01636 -0.09498 0.2149

0 0 6.48 -11.75 1.708

0 0 26.76 -18.15 0.523

Результат обработки при k=2

1 0 0 0.5818 0.7358

0 1 0 -0.1247 0.2192

0 0 1 -1.814 0.2636

0 0 0 30.37 -6.529

Результат обработки при k=3

1 0 0 0 0.8608

0 1 0 0 0.1924

0 0 1 0 -0.1263

0 0 0 1 -0.215

Проверьте, что подстановка в уравнения значений x0=0.8608, x1=0.1924,

x2=-0.1263, x3=-0. 215 дает тождества.

Самостоятельно измените функцию печати так, чтобы вместо таблиц на экран выводились уравнения (со знаками действий и обозначениями неизвестных).

Заметим, что программа завершится аварийно, если в исходной системе на главной диагонали будет нулевой элемент.

Чтобы этого не случилось можно переставить уравнения местами.

Пример программы, реализующей модификацию данного алгоритма нечувствительную к нулевым элементам (метод Жордана-Гаусса находится на диске в папке JrdGauss, в тексте учебника мы его разберем при знакомстве с оконным интерфейсом ОС Windows).