Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Магнитогорский государственный технический университет им. Г.И. Носова»

И.Г. Самарина

ПРОГРАММИРОВАНИЕ И ОСНОВЫ АЛГОРИТМИЗАЦИИ
ЧАСТЬ 1. КУРС ЛЕКЦИЙ

*Утверждено Редакционно-издательским советом университета
в качестве учебного пособия*

Магнитогорск
2013

УДК 004.4

Рецензенты

Заместитель директора ООО «ЭПОЛ»,
О.Е. Савостьянов

Кандидат технических наук,
доцент кафедры «Электроники и микроэлектроники»,
Р.С. Пишнограев

Самарина И.Г.
Программирование и основы алгоритмизации. Часть 1. Курс лекций [Электронный ресурс]: учебное пособие / Ирина Геннадьевна Самарина; ФГБОУ ВПО «МГТУ». – Электрон. текстовые дан. (0,31 Мб). Магнитогорск: ФГБОУ ВПО МГТУ, 2013. – 1 электрон. опт. диск (CD-R). – Систем. требования: IBM PC любой, более 1 GHz; 512 МБ RAM; 10 Мб HDD; MS Windows XP и выше; Adobe Reader 7.0 и выше; CD/DVD-ROM дисковод; мышь. – Загл. с контейнера.

Электронный образовательный ресурс написан в соответствии с учебной программой дисциплины «Программирование и основы алгоритмизации» и требованиями к обязательному минимуму содержания основной образовательной программы по направлению подготовки дипломированного специалиста. Рекомендуется при изучении по дисциплинам: «Теория автоматического управления», «Моделирование систем» «Самонастраивающиеся системы», «Операционные системы реального времени», «Системы автоматизации и управления», «Автоматизированное управление в технических системах»; «Автоматизация технологических процессов и производств», «Оптимизация управления технологическими процессами металлургического производства», «Диагностика и надежность автоматизированных систем», «Базы данных в АСУ ТП» и выполнении курсовых проектов по дисциплинам непосредственно для студентов, обучающихся по направлению 220400 «Управление в технических системах» и других специальностей.

Изложены основы программирования на языке C/C++. Включает краткие теоретические материалы, примеры решения задач. Задачи ориентированные на изучение программирования линейных, ветвящихся, циклических алгоритмов с использованием основных синтаксических конструкций языка C и C++.

УДК 004.4

Содержание

ПРЕДИСЛОВИЕ

Содержание предлагаемого учебного пособия основано на программе дисциплины «Программирование и основы алгоритмизации» изучаемой студентами по направлению 220400 «Управление в технических системах».

В пособие изложены основы программирования на языке C/C++. Включают краткие теоретические материалы, примеры решения задач. Задачи ориентированны на изучение программирования линейных, ветвящихся, циклических алгоритмов с использованием основных синтаксических конструкций языка C и C++

Методика изложения учебного материала пособия в основном связана с разборкой примеров. Примеры, приведенные в учебном пособии, в основном являются законченными программами, а не отдельными фрагментами. Все примеры были проверены непосредственно с текста пособия, где они напечатаны в виде, пригодном для ввода в машину. При работе над учебным пособием использовался компилятор, входящий в состав интегрированной среды разработки Microsoft Visual C++ 2005.

ВВЕДЕНИЕ

Язык программирования ***C*** был разработан Д. Ритчи (1972 г. Bell Laboratories, США) как универсальный язык системного программирования в связи с созданием популярной операционной системы UNIX. Эта операционная система (ОС) была, в основном, написана на языке ***C***, что обеспечило ее переносимость на любые ЭВМ.

Язык ***C*** – современный язык, включающий управляющие конструкции, рекомендуемые теоретическим и практическим программированием. Такими конструкциями являются следование, ветвление, циклы, модули, называемые функциями. ***C***++, являющийся дальнейшим развитием языка ***C***, содержит такой важный инструмент, как средства объектно-ориентированного программирования (ООП).

Бьерн Страуструп является разработчиком языка ***C***++ (1979 г. AT&T Bell Laboratories, США).

Основные этапы модернизации языка были произведены 1983, 1985 и 1990-х годах. В 1994 г. начался процесс стандартизации языка совместно с ANSI (Американский национальный институт стандартов) и ISO (Международная организация по стандартизации). Поэтому существует две версии языка: традиционная и версия Standart ***C***++ (отличаются компиляторами).

Язык ***C/C***++ – это язык программирования общего назначения, хорошо известный своей эффективностью, экономичностью, и переносимостью. Преимущества ***C/C***++ обеспечивают хорошее качество разработки почти любого вида программного продукта. Использование ***C/C***++ в качестве инструментального языка позволяет получать быстрые и компактные программы.

1. СТРУКТУРА ПРОГРАММЫ НА ЯЗЫКЕ C/C++

Структуру программы рассмотрим на примере программы печатающей строку текста. Программа написана в стиле ***C*** и ***C++.***

```
/* Программа в стиле C++ */
#include <iostream>
main ()
  {
  cout« "Моя первая программа на C++";
  return 0;
  }
```

```
/* Программа в стиле C*/
#include < stdio.h>
main ()
{
printf("Моя первая программа на C++");
return 0;
}
```

Первая строка данной программы начинается с символов ***/*…*/*** – это многострочные комментарии (текст является комментарием), данный текст игнорируется компилятором. Комментарии вставляются для документирования программы и облегчения ее чтения. Они помогают другим людям читать и понимать вашу программу. Комментарий, начинающийся с символа *//*, называется однострочным, потому что он должен заканчиваться в конце текущей строки.

Строка *#include<stdio.h>* или *<iostream>* является директивой препроцессора. **Препроцессор** – это специальная программа, которая обрабатывает строки программы, начинающиеся со знака **#**. Данная строка дает указание препроцессору перед компиляцией программы включить в нее информацию, содержащуюся в файле *stdio.h*. В новом стиле оформления заголовков *.h* не используется, так как новые заголовки не являются именами файлов.

Например: старый стиль – *math*.h, новый – *cmath*.

Далее обязательная функция ***main*** **()**, а круглые скобки указывают на то, что *main* – имя функции. *main()* главная функция, в *C/C++* выполнение программы начинается с выполнения этой функции. По окончанию работы функции *main()*, выполнение программы завершается, и управление возвращается операционной системе. Функция *main()* одна в программе.

Открывающая фигурная ***{*** скобка отмечает начало последовательности операторов, образующих тело функции.

Строка *cout « "…"*; или *printf("…")*; – оператор и функция вывода, с помощью которых выводится на экран дисплея фраза, заключенная в кавычки.

Функция может возвращать значение в программу с помощью оператора возврата (***return***). Этот оператор также означает выход из функции. Если же указанный оператор отсутствует, то функция автоматически возвращает значение типа ***void*** (пустой).

Закрывающая фигурная ***}*** скобка отмечает конец последовательности операторов, образующих тело функции. На этой скобке выполнение функции и программы завершается.

Программа на ***C++*** состоит из одной или более функций, причем одна из них обязательно должна быть *main()*.

2. ВВОД – ВЫВОД В C/C++

Ввод-вывод в языке ***C++*** осуществляется потоками байтов. *Поток* – это просто последовательность байтов. В операциях ввода байты пересылаются от устройства ввода (например, клавиатуры, дисковода или соединений сети) в оперативную память. При выводе байты пересылаются из оперативной памяти на устройства (например, экран дисплея, принтер или дисковод). Язык ***C++*** предоставляет возможности для ввода-вывода как на низком, так и на высоком уровнях. Ввод-вывод на низком уровне обычно сводится к тому, что некоторое число байтов данных следует переслать от устройства в память или из памяти в устройство. При такой пересылке каждый байт является самостоятельным элементом данных. Передача на

низком уровне позволяет осуществлять пересылку больших по объему потоков ввода-вывода с высокой скоростью, но такая передача обычно оказывается неудобной для программиста и пользователя. Операции ввода-вывода на высоком уровне осуществляются путем преобразования байтов в такие значащие элементы данных, как целые числа, числа с плавающей запятой, символы, строки и т.д. Стандартные библиотеки ***C++*** имеют расширенный набор средств ввода-вывода, при этом большая часть программ включает заголовок *<iostream>*, который содержит основные сведения, необходимые для всех операций с потоками ввода-вывода.

Например: ***cin***, ***cout***, ***cerr***, ***clog***

Объект стандартного потока ввода *cin* связан со стандартным устройством ввода (с клавиатурой). Операция взять из потока (*cin – the standard input stream* – стандартный поток ввода), показанная в приведенном ниже операторе, означает, что величина переменной *x* должна быть введена из объекта *cin* в память.

Например: *cin » x* ;

Объект стандартного потока вывода *cout* связан со стандартным устройством вывода (с экраном дисплея). Операция поместить в поток (*cout – standard output stream* – стандартный поток вывода), показанная в приведенном ниже операторе, означает, что величина переменной *x* должна быть выведена из памяти на стандартное устройство вывода.

Например: *cout « x*;

Объекты *cerr* и *clog* связаны со стандартным устройством вывода сообщений об ошибках. Их различие состоит в том, что при использовании *cerr* сообщение об ошибках выводится мгновенно, а объекта *clog* сообщения об ошибках помещаются в буфер, где они хранятся до тех пор, пока буфер полностью не заполнится или пока содержимое буфера не будет выведено принудительно.

В ***C/C++*** можно так же использовать форматированный ввод-вывод с помощью функций ***printf – scanf***.

scanf("%d",&x);

где *%d* – формат вводимого числа (*%d* – целое десятичное число типа *int*; *%c* – символ типа *char*; *%lf* – число типа *double* и т.д.);

& – операция взятия адреса;

x – имя вводимой переменной.

printf("Число равно %d",x);

где *%d* – формат выводимого числа;

x – имя выводимой переменной;

"Число равно" – произвольный текст.

3. ЯЗЫК ПРОГРАММИРОВАНИЯ И ЕГО ОПИСАНИЕ

3.1. Алфавит

Языки ***C/C++***, как и любые другие языки программирования, полностью определяются заданием их алфавита (словаря исходных символов), точным описанием их синтаксиса (грамматики) и семантики (смысла).

Алфавит языка – набор основных символов (*литер*), используемых для записи алгоритма. Следует отметить, что некоторые литеры алфавита являются составными, изображаются двумя или тремя символами, но рассматриваются как неделимые (например, "+=" или "»=").

Семантика определяет смысл предложений (*операторов*), записанных на языке, как каждого в отдельности, так и их совокупности.

При написании программ на языке ***C++*** используются символы, составляющие его *алфавит*. Набор символов зависит от среды выполнения. На ЭВМ широко используется символьный набор ISO 6461983, называемый кодом ASCII (American Standard Code for Information Interchange – американский стандартный код обмена информаций). Он содержит *латинские буквы*, *арабские цифры*, *специальные* и *управляющие* символы, которые в своем большинстве входят в состав *алфавита языка* ***C++.*** Каждый символ кодируется семибитным значением. Для представления *кириллических символов и*спользуется восьмибитный *расширенный* ASCII-код, в котором единичное значение старшего бита, говорит об использовании дополнительного символьного набора.

Алфавит ***C++*** включает *латинские* прописные и строчные *буквы:* A,..., Z, a,..., z, *арабские цифры:* 0,1,..., 9, *специальные символы:* + - * / <> = | & ! \ ~ ' @ # $ % ^ ? _ : ; , . () [] {} "

В качестве *символа-разделителя* элементов (слов) предложений языка используется *пробел,* который на экране не отображается, а для наглядности при записи на бумаге часто обозначается символом _. Предложения (операторы) языка обычно заканчиваются точкой с запятой. Исключение составляют директивы препроцессора, начинающиеся с символа #, составные операторы и блоки определения функций, которые обрамлены фигурными скобками – { }.

Кроме того, имеются *управляющие символы,* которые непосредственно на экране не отображаются. Для их записи используются специальные приемы. Например, запись управляющего символа *«горизонтальная табуляция»* – ' \t'.

Использование кириллических символов в некоторых случаях возможно и целесообразно (в комментариях, символьных строках, названиях файлов, если это допускает среда выполнения). Но пока единого стандарта на кодировку кириллических символов нет. Поэтому могут быть сложности при переносе таких программ с одного компьютера на другой, при переходе из одной среды выполнения в другую.

Специальные символы используются для обозначения (именования) *операций* и записи *выражений.* Например, запись *((a + b) * z)* является *выражением,* задающим вычисление суммы значений переменных *a* и *b* с последующим умножением на значение переменной *z.*

Например, ++ и -- являются знаками *унарных* операций *инкремента* (увеличения значения *операнда* на 1) и *декремента* (уменьшения значения *операнда* на 1) соответственно. Так, оператор инкремента переменной *time* имеет вид: time++.

Из символов алфавита формируются ***лексемы*** языка:

- идентификаторы;
- знаки операций;
- константы;
- разделители.

3.2. Идентификаторы

Имена имеют многие элементы программы: константы, переменные, типы данных, функции и ряд других. Такие имена являются *идентификаторами.* Имена вводятся для того, чтобы отличать (идентифицировать) различные элементы одного вида (типа) от других и оперировать (производить действия) с ними. ***Идентификатором*** называется последовательность символов из латинских букв, символа подчеркивания и арабских цифр, которая начинается с буквы и служит для именования различных элементов программы. Примеры идентификаторов: *var, A_ptr, NO.*

Идентификаторы могут включать любое число символов, из которых значимыми являются первые 32, то есть длинные идентификаторы считаются различными, если у них отличаются последовательности из первых 32 символов.

Строчные и заглавные буквы это разные символы. Поэтому идентификатор *Radius* отличается от идентификатора *radius.*

Некоторые идентификаторы языка зарезервированы в служебных *целях* и их нельзя использовать для именования переменных, констант и функций. Такие идентификаторы называют ***служебными*** или ***ключевыми*** *(keyword)* словами и входят в алфавит языка. Используемые в стандарте C++ ключевые слова приведены ниже:

asm	*auto*	*bool*	*break*
case	*catch*	*char*	*class*
const	*const cast*	*continue*	*default*
delete	*do*	*double*	*explicit*
dynamic cast	*else*	*enum*	*float*
export	*extern*	*false*	*if*
for	*friend*	*goto*	*mutable*
Inline	*int*	*long*	*private*
namespace	*new*	*operator*	*reinterpret_cast*
protected	*public*	*register*	*sizeof*
return	*short*	*signed*	*switch*
static	*static_cast*	*struct*	*true*
template	*this*	*throw*	*typename*
try	*typedef*	*typeid*	*virtual*
union	*unsigned*	*using*	*while*
void	*volatile*	*wchar_t*	

При подключении *стандартных библиотек* добавляется ряд специальных идентификаторов, таких как *cin, cout, list, size_t, string.* Их также не рекомендуется использовать в качестве идентификаторов.

Рекомендации по ***именованию****:*

- использовать имена из постановки задачи
- давать короткие осмысленные имена, отражающие назначение переменной, функции, объекта или типа;
- не начинать с символа подчеркивания, поскольку такой прием широко используется в библиотеках системы программирования;
- следовать единой системе именования; здесь существуют различные варианты, например:
  - начинать с прописной буквы, если требуется подчеркнуть уникальность идентификатора;
  - использовать символ подчеркивания или прописные буквы внутри идентификатора для построения хорошо читаемых сложных идентификаторов.

Обычно редко удается в именах элементов программы прокомментировать ее содержание. Поэтому для пояснения отдельных частей или всей программы используют *комментарии.* Для введения однострочного комментария используют пару символов //, после которых следует поясняющий текст до конца строки. Многострочные комментарии начинаются с пары символов /* и заканчиваются парой символов */.

Рекомендации по ***комментированию****:*

- начинать программу с кратких комментариев, описывающих основные этапы алгоритма, переменные для хранения исходных данных, промежуточных и выводимых результатов;

- писать комментарии в терминах постановки задачи и выбранного метода решения;
- не вставлять комментарии в середину строки программы;
- не писать очевидных комментариев;
- начинать комментарий с той же позиции в строке, что и комментируемый текст.

3.3 Классификация типов данных

В программе задаются действия с переменными и константами, которые предварительно должны быть объявлены для того, чтобы сообщить компилятору:

- имя переменной или константы;
- размер памяти, необходимый для хранения значений;
- какие действия можно выполнять с переменной или константой;
- вид и способ выделения памяти под переменную и константу;
- начальное значение переменной или значение константы.

Важную роль в обработке данных играют типы данных языка ***C++.*** Под ***типом данных*** понимают множество допустимых значений этих данных и множество разрешенных операций над ними. Одновременно тип данных определяет и размер памяти, который занимают переменные и константы данного типа. Каждый тип данных в языке имеет *имя* (*идентификатор*), простое или составное.

Память не выделяется для типа данных, а выделяется для размещения переменной или константы.

В языке ***C++*** выделяют следующие *категории типов:*

- *базовые типы* данных;
- *производные* (определяемые) типы.

Дерево классификации типов языка C++ приведено на рис. 1.

Базовые типы имеют имена, которые являются ключевыми словами языка.

К ***базовым*** типам относятся: *скалярные типы* и *пустой* тип – *void.*

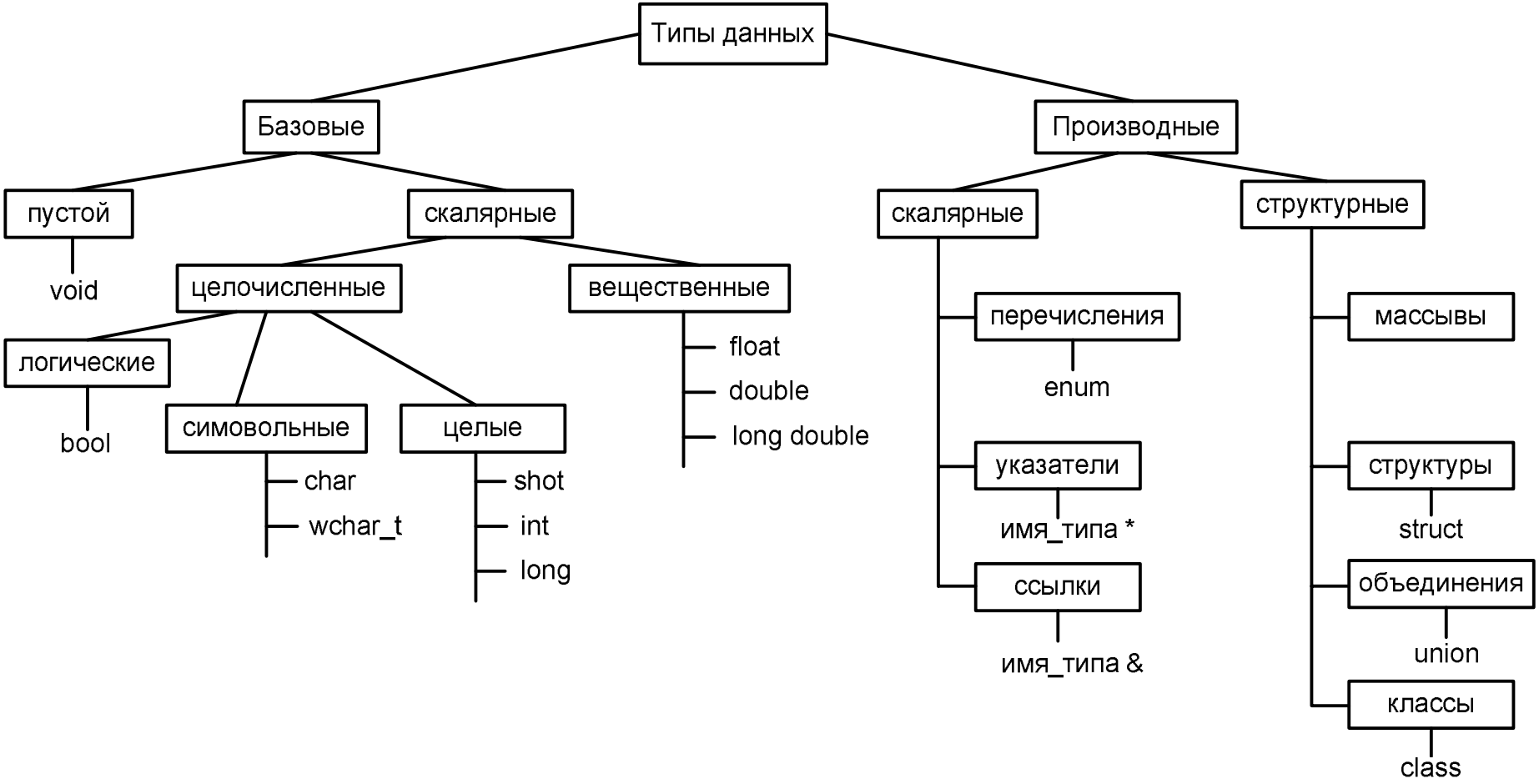

Рис. 1. Дерево классификации типов языка ***C***++

Тип ***void*** не имеет значения и введен в основном для описания функций, не возвращающих значении, и для некоторых других целей.

Скалярные типы делятся на *целочисленные* и *вещественные* типы.

Логический тип, *символьные* и *целые* типы данных являются *целочисленным типом, для* которого определены все операции с целыми числами.

Производные типы определяются (образуются) на основе базовых типов. Производные типы делятся на *скалярные* и *структурные* (агрегатные).

К скалярным производным типам относятся:

- *перечисления (**enum*** – enumeration) – множество поименованных целых значений;
- *указатели (имя_типа* *);
- *ссылки (имя_типа &).*

Структурными типами являются:

- *массивы (тип элемента имя_массива [число элементов*]);
- *структуры **(struct);***
- *объединения **(union);***
- *классы **(class).***

3.4. Переменные и константы

Обрабатываемые в программе данные можно разделить на *переменные* и *константы*. Перед использованием переменные и константы должны быть объявлены с помощью оператора объявления вида:

*[<спецификатор класса памяти>] [**const**] <спецификатор типа>*
<идентификатор> [= <начальное значение>]
[<идентификатор> [= <начальное значение>]]...;

Например: int a=5, y;
const float g = 9.81, C = 0.577216;

Квадратные скобки означают необязательность операнда.

Ключевое слово *const* указывает, что записанные справа идентификаторы являются ***символическими константами*** (константными переменными). При этом значение константы задается обязательно и в программе изменяться не может. Кроме константных переменных, константы могут задаваться в виде ***литеральных*** (самоопределенных) констант.

Тип *bool* введен в стандарте ***C++***, раньше он определялся на основе целых типов. *Логические переменные* типа *bool* могут принимать одно из двух значений: *false* (ложь) или *true* (истина). По определению значение *false* равно 0, а *true* равно 1. Логические переменные широко используются в операциях сравнения, логических операциях и логических выражениях. Размер переменной зависит от реализации, но обычно составляет 2 байта.

Например: bool reload = false;

В переменной типа *char* может храниться один из ASCII-символов.

Например: char ch = ' f ';

Одиночные кавычки используются для задания символьного значения переменной. Запись вида 's' называют *символьной константой* или *символьным литералом.*

Для явного задания диапазона можно использовать ***модификаторы*** *signed, unsigned.* В объявлениях символьных типов обычно эти модификаторы используются, когда значение символа задается с помощью числового значения или требуется «обратить внимание» на зависимость от реализации:

Например: unsigned char code = 87; // эквивалентно ‘W’

Переменные и константы целых типов также могут объявляться с помощью модификаторов *signed* и *unsigned*. При указании модификаторов *short* и *long* допускается опускать имя *int*, которое подразумевается по умолчанию. Модификатор *signed* также подразумевается по умолчанию.

Размеры целых типов зависят от реализации, но для всех версий языка ***C++*** принято, что выполняется следующая система неравенств:

sizeof(short) <= sizeof(int) <= sizeof(long).

Кроме того, во всех реализациях гарантируется

1 = sizeof (char) <= sizeof(short); 1<= sizeof(bool)<=sizeof(long);

Типы с плавающей запятой (точкой) или вещественные типы представлены тремя размерами, характеризующими точность представления вещественных чисел:

- *float* – одинарной точности;
- *double* – двойной точности;
- *long double* – расширенной точности.

Пример 1 – Объявление данных.

```
int x,y; short a, b;
long e; char st;
unsigned i,j;          /* или unsigned int i, j; */
float v, z; double u;
```

Используя, спецификатор типа *typedef*, можно в своей программе вводить удобные имена для сложных описаний типов.

Например: typedef unsigned char cod; cod p;

Здесь определена переменная *p* типа *cod*, который описан спецификатором *typedef*. Целью такого объявления обычно является введение короткого синонима, определяющего назначение типа данных в программе.

Константы, в отличие от переменных, являются фиксированными значениями, которые можно вводить и использовать в языках ***C/C++***. По способу задания константы можно разделить на *символические* и *литеральные* (самоопределенные).

Обычно константы задаются в выражениях в качестве конкретных значений операндов и значений параметров функций. Сама константа является простейшим выражением.

Литеральная константа представляет собой константу, тип и значение которой определяются ее внешним видом. В необходимых случаях память под константу выделяется автоматически, но может быть не выделена совсем, например, для литералов составе константного выражения. Содержимое выделенной памяти не изменяется и определяется типом и значением константы.

Различают следующие *виды* ***литеральных констант****:*

- числовые;
- символьные;
- строковые.

Рассмотрим на примерах эти виды констант.

Числовые константы:

- *вещественные* типа *double:*
  3,14 эквивалентна 314e-2, также эквивалентна 0.314E1;
  -*2.5* эквивалентна -0.25e1;
- *целые:*
  - *десятичные* типа *int*: 0,278, -579 – используются десятичные цифры;

- *восьмеричные:* 00, 01, …, 077777, …; – используются восьмеричные цифры, запись числа начинается с нуля
- *шестнадцатеричные:* 0x0000,0x0001,..., 0x7FFF,..., 0x FFFF,...

По форме записи числовой константы компилятор определяет ее тип: по умолчанию целые десятичные константы имеют тип *int,* вещественные *double,* если они принадлежат соответствующим множествам значений указанных типов. Если константа выходит за пределы множества значений типа *double* или *int,* то она относится к типу, следующему за *double* или *int* по мощности множества допустимых значений.

Символьные (литеральные) константы:

- *клавиатурные:* '1', ' t ', ' Y ' – клавиатурный символ задается в апострофах;
- *кодовые* – для задания некоторых управляющих и разделительных символов: ' /t ', ' /n '
- *кодовые числовые* – для задания любых ASCII-кодов символов

Для кодирования одного символа используется байт (восемь битов). Благодаря этому набор символов содержит 256 символов, образующих две группы: *печатные* символы и *непечатные* символы.

Непечатным символам соответствуют специальные управляющие коды, которые служат для управления внешними устройствами или для других видов управления. В качестве примера непечатного символа назовем символ перехода к новой странице, управляющий, например, работой принтера.

Символьная константа в языках ***C/C++*** состоит либо из одного печатного символа, заключенного в апострофы, либо управляющего кода, заключенного в апострофы. Управляющие коды представляют непечатные символы. Символьная константа рассматривается как символьный беззнаковый тип данных с диапазоном значений от 0 до 255. Константа ‘\0’ называется нулевым символом или нулевым байтом.

Например: ‘a’, ‘1’, ‘\n’

Строковые константы – последовательности **символов, ограниченные** двойными кавычками:

Например: "This string constant", "ERROR:"

Для хранения строковой константы требуется $n+1$ байтов, где n число символов между ограничителями строковой константы. Дополнительный байт последний и требуется для хранения кода 000, который записывается автоматически. Поскольку машинное представление строки имеет последний нулевой байт, то говорят, что строки С оканчиваются нулем.

Символические константы базовых типов объявляются так же, как и переменные соответствующих типов, но с указанием модификатора *const* (см. подраздел 1.4).

Например: const int x = 5;

```
const double z = 1E-5;
const float PI = 3.141593;
const char Sub = 't';
```

Другой способ задания *символических целочисленных констант* – использование типа **enum**.

Константы в языке ***C++*** можно задавать либо в явном виде (то есть указывать непосредственно значение константы), либо использовать идентификатор, которому присваивается значение константы. Определение константы с помощью идентификатора осуществляется в заголовке программы по следующей форме:

#define имя строка.

где имя - идентификатор; строка - любая последовательность символов, отделяемая от имени хотя бы одним пробелом и заканчиваемая в текущей строке.

Директива ***#define*** выполняет простую текстовую подстановку, то есть когда препроцессор встречает имя, он заменяет его на строку.

Например: #define YES 1 // ставит в соответствие имени I число 5

#define M 4

#define PI 3,14

Необходимо обратить внимание на то, что при использовании директивы *define* тип константы не имеет значения (константы YES, M, PI не имеют никакого конкретного типа). Определение констант с помощью директивы *define* наиболее предпочтительно, так как в случае изменения их значений в программе понадобится внести изменения только в одном месте.

3.5 Время существования и область видимости переменных

Каждая переменная, объявленная в программе, имеет две важнейших характеристики:

- время существования;
- область видимости.

Эти характеристики взаимосвязаны и существенно влияют на возможности использования переменной в программе. Взаимосвязь характеристик определяется способом выделения памяти для переменной.

Время существования, или время жизни переменной, измеряется в следующих двух относительных единицах.

1. *Локальное* время жизни – это время существования переменной при выполнении блока, в котором она объявлена.

2. *Глобальное* время жизни – это время существования переменной при выполнении всей программы.

Область видимости, или область действия (доступности) переменной, также измеряется в двух относительных единицах.

1. До *конца блока*, в котором объявлена переменная.
2. До *конца файла*, в котором объявлена переменная.

Управлять этими характеристиками переменных программист может двумя путями:

1. изменением места объявления переменной в программе.
2. использованием модификаторов *auto, register, static, extern.*

Автоматическая *(auto)* переменная или константа имеет *локальную* область действия и известна только внутри блока, в котором она определена. Для автоматической переменной выделяется временная память. Память выделяется при входе в блок, а при выходе из блока память, выделенная для переменной, считается свободной, т. е. Переменная уничтожается. Если спецификатор класса памяти не указан, то переменная в блоке по умолчанию считается автоматической.

Регистровая *(register)* переменная отличается от автоматической только памятью, которая выделяется для ее хранения. Регистровая переменная хранится в регистре процессора, и, соответственно, доступ к этой переменной гораздо быстрее, чем к той, которая хранится в оперативной памяти *(auto).* В случае отсутствия свободных регистров регистровая переменная становится автоматической.

Внешняя *(extern)* переменная является глобальной переменной. Спецификатор *extern* информирует компилятор, что переменная будет объявлена (без *extern)* в другом файле, где ей и будет выделена память.

Статической *(static)* переменной (константе) выделяется память после ее объявления и сохраняется до конца выполнения программы. Статические переменные при объявлении по умолчанию инициализируются нулевыми (логические, целые и вещественные) или пустыми значениями.

Пример 2 – Использование локальных и глобальных переменных

```
#include <iostream>
int globvar1 = 0;                  // глобальная переменная файла
static int globvar2 =1;// глобальная переменная файла
int main (void)
{.....................}
```

4. ОПЕРАЦИИ, ВЫРАЖЕНИЯ И ОПЕРАТОРЫ

Операции **и** ***выражения*** задают определенную последовательность действий, но не являются законченными предложениями языка.

Операнд представляет собой элемент-участник операции. Операндами могут быть *константы*, *переменные*, *вызовы функций* и *выражения*.

По числу операндов, участвующих в операции, различают:

- унарные операции (один операнд a++);
- бинарные операции (два операнда a + b);
- тернарные операции (три операнда ?:).

По типу выполняемых операций различают:

- арифметические операции – сложения (+), вычитания (-), умножения (*), деления (-), определение остатка (%); инкремента (++) и декремента (--);
- логические операции – логическое И (&&), логическое ИЛИ (||), логическое НЕ (!);
- операции отношения – больше (>), меньше (<), равно (==), не равно (!=) и т.д.);
- операцию условия (?:);
- операцию присваивания (=);
- операцию sizeof;
- операцию присваивания типов.

4.1 Арифметические операции

Операции ***инкремента*** обозначается символом 2-х плюсов “++”. Ее смысл сводится к увеличению значения операнда на 1. a = a+1 или a++;

Операция декремента обозначается символом 2-х минусов “--”. Служит для уменьшения значения операнда на 1. a = a - 1 или a--;

Существует две формы использования операции инкремента и декремента: префиксная, когда символ “++” или “--” находятся слева от операнда, и постфиксная, когда символы находятся справа от операнда.

Например: a-- постфиксная; ++a префиксная.

Эти две формы отличаются тем, в какой момент произойдет изменение (увеличение или уменьшение на 1) операнда.

В префиксной форме сначала выполняется изменение операнда на 1, а затем измененное значение используется в выражении. В постфиксной форме сначала берется значение операнда и используется в выражении, а затем операнд изменяется на 1.

Пример 3 – Операция инкремента (формы префиксная и постфиксная)

```
void main()
```

```
{ float a, b, c;
cout<<"Введите a, b, c";
cin>>a>>b>>c;
a=b+c++/5;              // a=b+++c/5;
cout<<"a="<<a<<" b="<<b<<"\n";
}
```

4.2 Операции отношения

Эти операции используются для выполнения различных сравнений и являются основой для построения логических операций.

Логическое выражение – это выражение, имеющее одно из двух значений: истинно (значение 1) или ложно (значение 0).

Сравнивать можно операнды, принимающие численное значение (целые, плавающие, указатели), но плавающие не рекомендуется.

Пример 4 – Операции отношения больше (>), меньше (<)

```
void main()
{ int i=1,j=2,k,m; // k=m=0
k=1+(i<j);
m=1+(j<i);
printf("k = %d m = %d \n",k,m);
}
```

4.3 Поразрядные операции и операции сдвига

В ***C/C++*** возможно использование следующих поразрядных логических операций:

& - поразрядное логическое И;

| - поразрядное логическое ИЛИ;

^ - поразрядное логическое сложение по модулю (исключающее ИЛИ);

~ - поразрядное отрицание.

Данные операции выполняются над каждым разрядом в отдельности.

Таблица

Поразрядные логические операции

A	B	A&B	A|B	A^B
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Поразрядные логические операции допустимо использовать для целочисленного и символьного типа, и необходимо, чтобы они имели одну длину.

Пример 5 – Поразрядные логические операции

```
void main()
{unsigned short k,l,m;
unsigned short i=0xAB00;
unsigned short j=0xABCD;
k=i&j;
l=i|j;
m=i^j;
printf("k=%x l=%x m=%x\n",k,l,m);
}
```

Операции поразрядного сдвига также применяются только к данным целочисленного и символьного типа. Эти операции сдвигают разряды вправо или влево.

Операции сдвига влево: ***1 << 2***

Эта операция сдвигает разряды операнда 1 влево на количество позиций, определяемое операндом 2. Разряды, выдвигаемые за пределы операнда 1, теряются, а освободившиеся заполняются 0 справа. Причем в арифметическом логическом устройстве запоминаются начальные значения операндов 1 и 2.

Операции сдвига вправо: ***1 >> 2***

Данная операция сдвигает разряды операнда 1 вправо на количество позиций, определяемое операндом 2. Разряды, выдвигаемые за пределы операнда 1, теряются, а освободившиеся заполняются нулями справа.

Пример 6 – Операции сдвига

```
void main()
        {unsigned short i,j,k;
        i=0xAA;
        j=0x5500;
        k=(i<<8)+(j>>8);
        printf("i=%x j=%x k=%x\n",i,j,k);
        }
```

Вызов функции представляет собой указание имени вызываемой функции, за которым в круглых скобках указывается список аргументов (возможно пустой): pow (2, n+1).

Выражение – это последовательность операндов, операций и символов-разделителей. Разделителями в C++ являются символы [ ] ( ) { } , ; : … * = #, каждый из которых выполняет свою функцию. Выражение может состоять из одного или более операций. *Простейшим выражением* является *константа*, *переменная* или *вызов функции*, *то есть* содержат знак операции и операнды. Выражение может состоять из одного или более операций:

Например: -2-(cos(x)+5*y) заменится значением *8.0*, если $x = 0$ и $y = 1$.

Порядок вычисления выражения определяется расположением знаков операций, круглых скобок и *приоритетами* выполнения операций.

Операторы так же, как операции и выражения, задают определенную последовательность действий компилятора или вычислительной машины, но, в отличие от выражений, являются законченными предложениями языка. Они могут содержать несколько выражений, разделенных запятой.

Например: *оператор вывода-ввода*, содержащий два выражения, вид:

```
cout << "Введите значение x:";
cin >> x;
```

Обычно операторы заканчиваются *точкой с запятой. Пустой оператор* содержит только точку с запятой. Простые операторы можно объединить в *составной оператор* или *блок* с помощью фигурных скобок, за которым точку с запятой можно не ставить. Составные операторы и операторы, полученные в результате вложения (суперпозиции) операторов называются *сложными.*

Операторы имеют названия, отражающие их назначение или осуществляемые действия. Так, оператор, в котором используется операция присваивания, обычно называют ***оператором присваивания***.

Например: y = (a * x + b) * x + c;

и *оператор - выражения* count–;

Операторы, в которых объявляются константы, переменные, заголовки функций и типы данных называются *операторами объявления.* Имеются операторы с другими названиями.

Операторы, задающие действия вычислительной машины, могут быть помечены идентификатором, который называется *меткой* и заканчивается двоеточием. Метки используются для перехода к ним с помощью оператора ***goto*** *(безусловного перехода).*

5. ОПЕРАТОРЫ УПРАВЛЕНИЯ

5.1 Условный оператор *if*

Условный оператор обеспечивает выполнение или невыполнение некоторого оператора или группы операторов в зависимости от заданного условия. Оператор ***if*** *явля*ется одним из самых популярных средств, изменяющих естественный порядок выполнения операторов программы. Он может использоваться в одной из следующих фор:

if (логическое выражение) оператор 1
или
if (логическое выражение) оператор 1; [else оператор 2;]

Если значение *логического выражения* истинно (отлично от нуля), то выполняется *оператор1;* если ложно (равно нулю), то для первой формы *оператор1* пропускаете, а для второй формы после пропуска *оператор1* выполняется *оператор2,* стоящий после слова *else.* Иногда после проверки условия необходимо выполнить более чем один оператор, тогда требуемая для выполнения после *if* часть программы заключается в блок с помощью фигурных скобок { }.

*Пример 7 – Проверка правильности ввода переменной, в диапазоне от 1 до 31 (**1 форма**)*

```
…..
cin>>den;
if(den<1||den>31) cout<<”Ошибка!”;
…..
```

*Пример 8 – Найти максимум из трех чисел **(2 форма)***

```
……
if(a>b&&a>c) max = a;
      else if(b>c) max = b;
                 else max = c;
cout<<”max=”<<max;
……
```

5.2 Операция условия *?:*

В языке ***C++*** имеется короткий способ записи оператора *if... else.* Для этого используют *операцию условия*. Она имеет следующую форму записи:

(логическое выражение) ? выражение1 : выражение2

Если условное выражение истинно, то выполняется *выражение1,* если ложно *выражение2.*

Пример 9 - Найти максимум из двух чисел x и y

```
.........
max = (x > y) ? x : y;
cout << "max=" << max;
```

Операцию условия удобно использовать в случаях выбора значения из двух возможных. Применение этой операции не является обязательным, так как тех же результатов можно достичь при помощи оператора *if... else.* Однако получаемые при использовании операции условия выражения более компактны и их применение приводит к получению более компактного машинного кода.

5.3 Множественный выбор: операторы *switch* и *break*

Оператор *switch,* который имеет следующую форму записи:

```
switch ( выражение )
{
case константа1: оператор1 break;
.....
case константаN: операторN break;
default: оператор break;
}
```

Оператор выбора работает следующим образом. Сначала вычисляется выражение, стоящее в скобках после слова *switch.* Затем осуществляется переход на одну из меток обозначенную словом *case*, значение константы, после которой совпало со значением выражения в скобках после *switch.* Константа, стоящая после *case,* должна быть целого типа. Если проверяемое выражение не совпало ни с одной из проверяемых констант, то осуществляется переход на метку *default* (ее использование не является обязательным).

Опытные программисты для поиска ошибок часто включают *default,* даже когда учтены все возможные случаи.

Обычно действие каждой ветви заканчивается оператором *break.* Выполнение этого оператора приводит к выходу из оператора *switch.* Если *break* отсутствует, то управление передается следующему оператору, помеченному *case* или *default.* Подобным образом выполняются все последующие операторы внутри *switch,* пока не встретится оператор *break.* Ключевые слова *case* и *default* не могут находиться за пределами блока *switch.*

Пример 10 – Проанализировать значение переменой day

```
int day;
day=8;
    switch(day)
    {case 1:cout<<"Понедельник"<<"\n";break;
    case 2:cout<<"Вторник"<<"\n";break;
    case 3:cout<<"Среда"<<"\n";break;
    case 4:cout<<"Четверг"<<"\n";break;
    case 5:cout<<"Пятница"<<"\n";break;
    case 6:cout<<"Суббота"<<"\n";break;
    case 7:cout<<"Воскресенье"<<"\n";break;
    default:cout<<"Такого дня недели нет"<<"\n";
     }
```

6. ТИПЫ ОПЕРАТОРОВ ЦИКЛОВ

При выполнении программы в некоторых случаях возникает необходимость неоднократного повторения однотипных вычислений над различными данными. Для этого используются циклы.

Для организации циклов в ***C++*** используются следующие три оператора: ***while***, ***for*** и ***do – while***.

6.1. Цикл *while*

Циклом с предусловием.

Используется:

- не известно точное число повторов;
- при этом нет необходимости, чтобы цикл непременно был выполнен хотя бы один раз.

Форма записи:

while (выражение) оператор

Оператор является телом цикла, может быть простой оператор или совокупность операторов, объединенных в блок скобками { }.

Если выражение истинно (не равно нулю), то тело цикла выполняется один раз, затем выражение проверяется заново. Итерации (проверка условия и тело цикла) выполняются до тех пор, пока выражение не станет ложным (равным нулю).

При организации цикла типа *while* в его тело должны быть включены инструкции, изменяющие логику выражения, чтобы оно стало ложным. Иначе, выполнение цикла никогда не закончится.

Пример 11 – Цикл типа while

```
#define YES 1
#define NO 0
void main()
        { int ch, nl, nw, nc, inword;
        inword = NO;
        nl = nw = nc = 0;
        while((ch = getchar())! = EOF)
                {       nc++;
                if (ch == '\n')
        nl++;
                if (ch == ' '||ch == '\t'||ch == '\n')
                inword = NO;
                else
                if (inword == NO){inword = YES; nw++;}
                 }
                cout <<" nc =”<< nc<<” nl=”<<nl<<” nw=”<<nw;            }
```

6.2. Цикл *for*

Цикл с параметрами и используется когда известно точное количество повторов вычислений.

Форма записи:

for (выражение 1; выражение 2; выражение) оператор

Выражение 1 вычисляется первым. Обычно это инициализация счетчика циклов и переменных, вычисляется один раз, когда цикл начинает выполняться.

Выражение 2. предназначено для проверки условия. Если его значение отлично от нуля, то выполняется оператор (тело цикла), если равно нулю – цикл завершается.

Выражение 3 в конце каждого выполнения тела цикла.

В качестве оператора может использоваться простой оператор или совокупность операторов { }.

В круглых скобках после слова for могут отсутствовать все выражения или любое из них, но должны обязательно быть две точки с запятой. Специальное правило для цикла *for*: если отсутствует выражение 2, то результат проверки всегда истина.

Пример12 – Вычислить сумму ряда $x + \frac{x^3}{2} + ... + \frac{x^{2n+1}}{n^2+1}$ *при* $0{,}1 \le x \le 1{,}0$; $n_{max} = 10$ *используя простейшие математические функции. Сумма ряда определяется* $S = S + C_n$, *где* C_n *– n-ый член ряда, который можно определить:*

$$C_n = \frac{a_n}{a_{n-1}} = \frac{x^{2n+1}}{n^2+1} \cdot \frac{(n-1)^2+1}{x^{2(n-1)+1}} = \frac{x^{2n+1}}{n^2+1} \cdot \frac{(n-1)^2+1}{x^{2n-1}} = x^2 \frac{(n-1)^2+1}{n^2+1}$$

```
#include<iostream>
#include<cmath>
#include<iomanip>
using namespace std;
#define MAX 1.0
#define MIN 0.1
#define NUMBER 10
using namespace std;
int main()
        { double  i,x,s,step,c;
        x=MIN;
        step=(MAX-MIN)/NUMBER;
    while(x<=MAX)
                {s=0.0;
                for(i=0;i<=10;i++)
                        {c=pow(x,2.0)*(pow((i-1),2)+1)/(pow(i,2)+1);
                s=s+c;
                          }
cout<< fixed <<setprecision(2)<<"x = "<<x<<setw(8)<<"s = "<<s<<"\n";
x+=step;        }
        }
```

6.3. Цикл do – while

Цикл с постусловием и используется когда неизвестно точное количество повторов, но цикл необходимо выполнить хотя бы один раз.

Отличается от цикла *while* тем, что проверка истинности выражения происходит после выполнения тела цикла.

Форма записи:

do оператор while (выражение)

Пример 13 – Ввод дней месяца с проверкой правильности ввода

```
…..
do cin>>day;
while (day<1||day>31);
cout<<day
……
```

Вложенные циклы конструкции, в которой один цикл выполняется внутри другого. Внутренний цикл выполняется полностью во время каждой итерации внешнего цикла.

Пример 14 - Заполнить экран символами' #'

```
…….
for (i = 1; I <= 25; i++ ) for (k = 1; k <= 80; k++ ) cout << ‘ # ’;
……..
```

В данной программе 25 раз осуществляется вывод по 80 символов.

В программе можно использовать любые комбинации вложенных циклов всех типов: *while, for, do while,* если этого требует логика построения программы.

7. МАССИВЫ

Это набор однотипных объектов с общим именем, которые различаются местоположением в этом наборе (или индексом, присвоенным каждому элементу массива). Элементы массива занимают один непрерывный участок памяти и располагаются последовательно друг за другом.

В языках ***C/C++*** массив не является стандартным типом данных. Он может иметь тип: char, int, float, double и т.д. Допускается создавать массивы массивов, указателей, структур и др.

Свойства массивов:

- в массиве хранятся отдельные значения, которые называются элементами;
- все элементы массива сохраняются в памяти последовательно, и первый элемент имеет нулевое смешение адреса, т.е. нулевой индекс;
- имя массива является константой и содержит адрес первого элемента массива.

Форма объявления массива размерности n:

тип <имя массива>[размер 1] [размер 2]…. [размер n];

тип – базовый тип элементов массива,

[размер 1] [размер 2]…. [размер n] – количество элементов одномерных массивов входящих в многомерный массив.

Чаще всего используются одномерные массивы, форма:

тип <имя массива>[размер]

Например: int A[10], бращение к i элементу массива A[i], i – индекс элемента массива, который начинается с 0 и заканчивается n-1.

Поскольку все элементы массива имеют одинаковый, заранее установленный размер, а имя массива содержит адрес первого его элемента, то нетрудно вычислить адрес любого другого элемента. Но при этом должен соблюдаться еще один принцип – строгой последовательности хранения в памяти всех элементов массива, от нулевого до последнего, причем первый элемент имеет наименьший адрес, а последний – наибольший.

Имя массива представляет собой константное значение, которое не изменяется в ходе выполнения программы, поэтому оно не может размещаться слева от оператора присваивания, то есть не является *левосторонним значением.* Если бы этого ограничения не существовало, программа могла бы изменять содержимое имени, а смысл подобного изменения состоял бы в замене адреса нулевого элемента массива. На первый взгляд кажется, что данное ограничение малозначительно, но на самом деле определенные выражения, выглядящие вполне корректными, оказываются недопустимыми.

Пример 15 – Описания массивов

```
int mas1[492];                    // внешний массив из 492 элементов
void main (void)
{
   double mas2[250];              // массив из 250 чисел типа double
   static char mas3[20];  // статическая строка из 20 символов
   extern mas1[];           // внешний массив, размер указан выше
   int mas4[2] [4];               // двумерный массив из чисел типа int
}
```

В данном примере квадратные скобки [ ] обозначают, что все идентификаторы, после которых они следуют, являются именами массивов. Число, заключенное в скобки, определяет количество элементов массива. Доступ к отдельному элементу массива организуется с использованием номера этого элемента, или индекса. Нумерация элементов массива начинается с нуля и заканчивается *n-1*, где *n* – число элементов массива.

Пример 16 – Описание массива и присваивание начальных значений его элементам

```
……
int mas [2];              //объявление массива
int a=10,b=5;             //объявление переменных
…….
mas[0]=a;
mas[1]=b;
…….
```

Инициализация массива означает присвоение начальных значений его элементам при объявлении. Массивы можно инициализировать списком значений или выражений, отделенных запятой, заключенным в квадратные скобки.

Например: int days[12] = {31,28,31,30,31,30,31,31,30,31,30,31};

Если список инициализируемых значений короче длины массива, то инициализации подвергаются первые элементы массива, а остальные инициализируются нулем.

Массив также можно инициализировать списком без указания в скобках длины массива. При этом массиву присваивается длина по количеству инициализаторов.

Например: char code[ ] = {'a', 'b', 'c'};

В данном примере массив *code* будет иметь длину 3.

Если же массив явно не проинициализирован, то внешние и статические массивы инициализируются нулями. Автоматические же массивы после объявления ничем не инициализируются и содержат неизвестную информацию.

Массив можно инициализировать одним из трех способов:

- при создании массива – используя инициализацию по умолчанию (этот метод применяется только для глобальных и статических массивов);
- при создании массива – явно указывая начальные константные значения;
- в процессе выполнения программы – путем записи данных в массив.

При создании в массив могут быть занесены только константные значения. Впоследствии в массив можно записывать и значения переменных.

Пример 17 – Копирование одно массива в другой

```
void main()
  { int i, m[5] = {1, 2, 3, 4, 5}, mcopy[5];
  for (i=0; i<5; i++) mcopy[i] = m[i];
  cout<<”Исходный массив”<<set(10)<<”Массив копия”;
  for (i=0; i<5; i++)
cout<<”\n m[“<<i<<”] =”<<m[i]<<”\t\t\t mcopy[”<<i<<”]=”<< mcopy[i];
}
```

Многомерный массив это массив, элементы которого одномерные массивы. int A[2][3] = {{…}, {…}, {…}} – 2 строчки и 3 столбца, можно инициализировать одной строчкой: int A[2][3] = {……}.

Пример 18 – Ввод и вывод двухмерного массива

```
#define N 2
#define M 3
void main()
   { float x[N][M];
   int i, j;
   for(i=0;i<N;i++, cout<<”\n”)
   for(j=0;j<M;j++)
         {x[i][j] = sqrt(x[i][j]*2 – 1);
     cout<< x[i][j];
         }
   }
```

8. СТРУКТУРЫ

Это набор взаимосвязанных данных, чаще разных типов, объединенных в единое целое. При использовании структуры ее необходимо объявить, а затем создать элемент этой структуры. Структура описывается с помощью ключевого слова **struct**.

Например: struct STUDENT {
char FIO[20];
int code;
};

STUDENT – имя структуры, с элементами, которые называются полями структуры: символьный массив FIO[20] и переменная целого типа code.

После описания структуры можно создать переменную типа STUDENT или экземпляр данной структуры, например STUDENT stud.

К отдельным элементам структуры можно получить доступ с помощью оператора “*точка*” **(.)** или оператор “*стрелка*” **(- >)**.

Например: stud. code = 12;

Пример 19 – Структура сотрудник

```
int main()
{ struct SOTRUDNIC { char FIO[40];
                     int oclad;
                     int day; };
  int premiy;
SOTRUDNIC str1 = {"Петров И.С.", 2800, 32};
SOTRUDNIC str2 = {"Орлов А.А.", 2900, 42};
SOTRUDNIC str3 = {"Иванов В.К.", 3200, 39};
cout<<"Премия = ";
cin>>premiy;
cout<< str1.FIO;
cout<<"Зарплата = "<<(str1.day*str1.oclad+premiy)<<"\n";
}
```

Можно для ввода структуры использовать следующую конструкцию:

typedef struct { поля структуры
……;} <имя структуры>;

Пример 20 – Структура сотрудник

```
struct PERSON { int age;
                long ss;
                float weight;
                char name[25]; }
int main()
{ int max;
  struct PERSON sister;   // стиль C
  PERSON brother;                 // стиль C++
  cout<<"Введите возрост сестры: ";
  cin>>sister.age;
  cout<<"Введите возрост брата: ";
  cin>>brother.age;
  if (sister.age>brother.age) max=sister.age;
        else max=brother.age;
  cout<<"Максимальный возраст="<<max<<"\n";
}
```

9. УКАЗАТЕЛИ

Это тип переменной, содержащей в памяти адрес того элемента, на который он указывает. При этом имя элемента отсылается к его значению прямо, а указатель косвенно – *косвенная адресация*.

Указатель может указывать на любые объекты: переменные, массивы, классы, структуры и функции.

Описание переменных типа указатель выполняется:

<тип><имя указателя на переменную заданного типа>;*

Например: int *r – указатель на целое число.

***** – операция косвенной адресации.

Для инициализации указателей используется операция присваивания.

Например: int a = 10; int *ptr = &a – инициализация указателя адресом переменной a.

Операция ***** в некотором смысле является обратной операции ***&*** – операция определения адреса.

Например: если vr – имя переменной, то &vr – адрес этой переменной.

Пример 21 – Структура сотрудник

```
void main()       //
/* имя переменной используется только компилятором, в программе используется
адрес*/
   { int i=100;
   int *ptrA,*ptrB;              //отводится память, но там ничего
   ptrA = &i;                    //инициализация ptrA адресом переменной i
   printf("Адрес i=%p\n",&i);
   printf("Значение ptrA=%p\n",ptrA);
   printf("Значение i=%d\n",i);
   printf("Значение по адресу ptrA=%d\n",*ptrA);
   ptrB=(int*)malloc(sizeof(int));
   *ptrB=i;                  //загрузить по адресу
   printf("По адресу *ptrB находится %d\n",*ptrB);
   free(ptrB);}
   }
```

Результат:

Адрес i = 0012FF54
Значение ptrA = 0012FF54
Значение i = 100
Значение по адресу ptrA = 100
По адресу *ptrB находится 100

9.1 Указатели и массивы

В языке ***C***++ указатели связанны с работай массивов. Имя массива является константным указателем на первый элемент массива.

Например: int M[4] – массив

M[i] – обращение к элементу массива по индексу, *(M+i) – обращение к элементу массива по указателю (адрес i элемента + смещение), где *M – адрес начала массива (или адрес первого элемента массива с индексом 0).

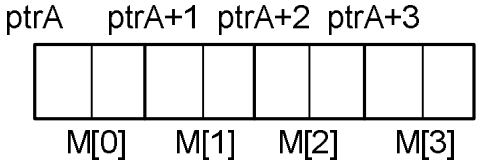

Рис. 2 Размещение массива в памяти

Объявление двумерного массива: int M[4][2]; int *ptr;

Тогда выражение ptr = M указывает на первый столбец первой строки матрицы.

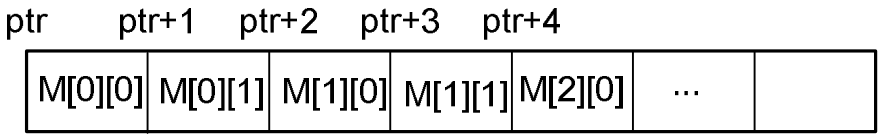

Рис. 3 Размещение двухмерного массива в памяти

Массивы, размер которых становится известен в процессе выполнения программы, называются динамическими. Для работы с этими массивами используются указатели и специальные операторы ***new*** – выделение памяти под динамический объект и ***delete*** – удаление из памяти.

Пример 22 – Выделение памяти для одномерного массива

```
…..
int n;
cin >> n;                  //n – размерность массива
int *mas = new int[n];     // выделение памяти под массив
delete mas;                        // освобождение памяти
```

Пример 23 – Выделение памяти под двухмерный массив

```
…..
    int n, k, i., * *mas;
    cin >> n>> k;              // число строк и столбцов массива
    mas = new * int[n];        // выделение памяти под n указателей на строку
```

```
for (i = 0;i < n; i++)        // выделение памяти для каждой строки по числу столбцов
                              k
mas [i] = new int[k];
for(i = 0; i < n; i++)
delete mas [i];               // освобождение памяти
delete [] mas;
```

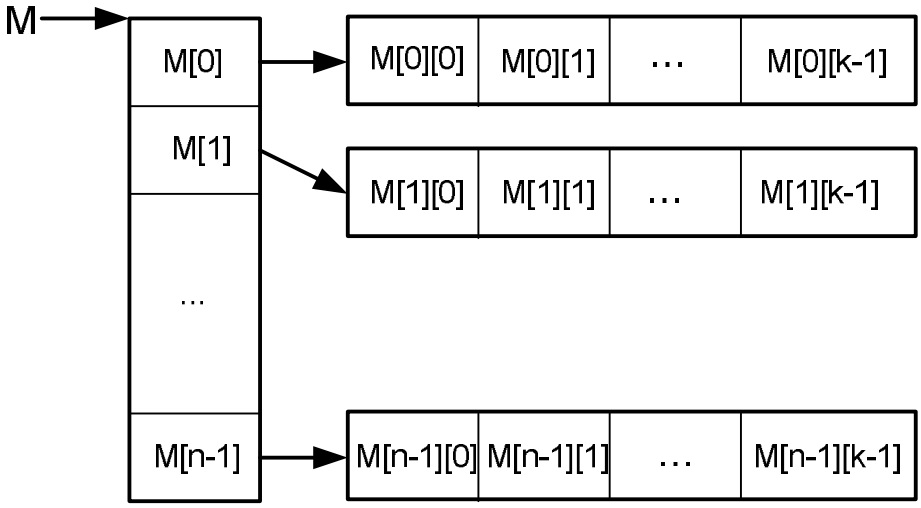

Рис. 4 Размещение двухмерного динамического массива в памяти

Сначала необходимо с помощью операции *new* выделить память под *n* указателей. Выделенная память будет представлять собой вектор, элементом которого является указатель. При этом все указатели располагаются в памяти последовательно друг за другом. После этого необходимо в цикле каждому указателю присвоить адрес выделенной области памяти размером, равным второй границе массива.

Пример 24 – Указатель и структура

```
#include<cstring>
void main()
  { SOTRUDNIK struct { char im[10];
  char ot[10];
  char fa[10];
  float salary;};
  SOTRUDNIK *ptr;
  ptr=(SOTRUDNIK*)malloc(sizeof(SOTRUDNIK));  или
                                          //ptr = new SOTRUDNIK
  strcpy (ptr->im,"Иван");
  strcpy (ptr->ot,"Петрович");
  strcpy (ptr->fa,"Орлов");
  ptr->salary = 4500;
  printf("%s %s %s %f\n", ptr->im, ptr->ot, ptr->fa, ptr->salary);
  free(ptr);
  }
```

Пример 24 – Массив указателей

```
void main()
 { locale::global(locale("rus"));
```

```
int i;
char *a[3]={"МИР!","ТРУД!",NULL};
/*массив из 3-х указателей, значение их адрес*/
char *ptr="МАЙ!";                    // адрес начала МАЙ, был NULL
a[2]=ptr;
for(i=0;i<=2;i++)
printf(" %p %s\n",a[i],a[i]);
   }
```

Результат:

```
00416718 МИР!
00416710 ТРУД!
00416708 МАЙ!
```

10. ФУНКЦИИ

Это именованная часть программы (блок кода, не входящий в основную часть программы), к которой можно обращаться из других частей программы столько раз, сколько потребуется.

С понятием функции в языке ***C++*** связано три компонента:

- описание функции;
- прототип;
- вызов функции.

Описание функции состоит из двух частей: заголовка и тела, имеет следующую форму записи:

тип <имя функции> (список параметров) //заголовок
{операторы тела функции} //тело

тип – тип значения, которое возвращает функция с помощью оператора return, если тип не указан – по умолчания тип int.

список параметров состоит из перечня имен и типов параметров, разделенных запятыми. Функция может не иметь параметров.

операторы тела функции входит оператор ***return***, его может не быть, если функция ничего не возвращает.

Прототип функции может быть до вызова функции вместо описания функции. По форме как заголовок, в конце его «***;***». Параметры функции в прототипе могут иметь имена, но компилятору они не нужны.

Компилятор использует прототип для сравнения типов аргументов с типами параметров. Язык ***C++*** не предусматривает автоматического преобразования типов в случаях, когда аргументы не совпадают по типам с соответствующими им параметрами, то есть язык ***C++*** обеспечивает строгий контроль типов.

При наличии прототипа вызываемые функции могут размещаться в разных файлах с вызывающей функцией.

Если описание функции выполнено до ее вызова, то прототип не требуется.

Вызов функции может быть в виде оператора, если у функции отсутствует возвращаемое значение (пример 25), или в виде выражения (пример 26), если существует возвращаемое значение.

Пример 25 – Передача параметров в функцию по значению

```
int f(int U, int V, int W);     // прототип функции
int main()
   { // тело функции main()
```

```
    int a, b, c, S;
    //вызов функции – в виде выражения
    S = f(a, b, c);          // фактические параметры (аргументы)
…….}
    //описание функции
int f(int U, int V, int W)  // формальными параметры – заголовок
{ // тело функции f
  int S;
  ……
  return S;
  }
```

Значение вычисленного выражения является возвращаемым значением функции. Возвращаемое значение передается в место вызова функции и является результатом ее работы.

Число и типы аргументов должны совпадать с числом и типом параметров функции. При вызове функции параметры подставляются вместо аргументов. Параметры, перечисленные в заголовке функции, называются формальными, а записанные в операторе или выражении вызова – фактические (или аргументы).

Во многих практических задачах результатов может быть несколько, но имя функции является носителем одного результата. Поэтому в качестве параметров и аргументов используются адреса переменных, а не их значения.

Пример 26 – передача параметров в функцию по адресу:

```
void f( int *U, int *V)
{ int c;
c=*U; *U = *V; *V=c;
}
void main()
  { int a, b;
  f(&a, &b);
  …….
  }
```

a и b – аргументы функции, при обращении к функции значения адресов-аргументов передается по значению указателям-параметрам U и V.

В ***C++*** имя массива является константным указателем, то есть при использовании имени массива в качестве параметра функции допускается изменять значения элементов массива (Пример 27).

Например: одномерный массив параметр функции:

```
int sum (int* , int );
void main()
  { …….
  s = sum (mas, n);
  ……}
int sum (int *mas, int n)
  { ……..
  return s;        }
```

Например: двухмерный массив параметр функции:

```
int sum (int* mas, int n, int m)
  { …….
  return s; }
void main()
```

```
{ ……..
s = sum (&mas[0][0], n, m);
……}
```

Пример 27 – Функция заменяе в строке одну букву на другую

```
void zam(int,char[ ]);
void main()
   {const int n=20;
   char m[n]="Надо хорошо учиться";
    cout<<" Замена: "<<"\n";
    zam(n,m);
    for(int i=0; i<n; i++)
    cout<<m[ i ];
    cout<<"\n";
    }
void zam(int k,char c[ ])
    {int i;
    for(i=0;i<k;i++)
    if(c[ i ] == 'o')c[ i ]='a';
    }
```

10.1 Функции с переменным числом параметров

В **C++** можно использовать функции, у которых до выполнения программы число параметров заранее неизвестно. Количество и типы параметров становятся известными при вызове функции. Это функции с переменным числом параметров. Описание:

тип <имя функции> (список явных параметров,…)
{ тело функции }

список явных параметров – параметры, которых известны до вызова функции.

При работе с такими функциями используется 2 положения:

1. один из параметров определяет число дополнительных параметров функции;
2. в список явных параметров последним задается параметр-индикатор, указывающий на окончание списка параметров.

Переход от одного параметра к другому в обоих случаях осуществляется с помощью указателей.

Пример 28 – Функция содержит переменное число параметров и вычисляет сумму значений дополнительных параметров

```
int sum(int, ...);
void main()
  { cout<<" 4+6 = "<< sum(2,4,6);
  cout<<"\n 1+2+3+4+5+6 = "<< sum(6,1,2,3,4,5,6);
  cout<<"\n Параметры отсутствуют, сумма = "<< sum(0) <<"\n";
  }
int sum(int n, ...)
   {int *p=&n;
   int s=0;
   for(int i=1; i<=n; i++)
      { if (n==0)break;
      s+=*(++p);
      }
   return s;  }
```

10.2 Рекурсивные функции

В **C/C++** функции могут вызывать сами себя – *рекурсия*. А функция, которая сама себя вызвала, называется ***рекурсивной***.

Все параметры должны иметь один тип int или double. В практике иногда возникают необходимости программировать рекурсивные алгоритмы. Это требуется при реализации динамических структур данных, таких как стеки, деревья, очереди.

Пример 29 – Использование рекурсивной функции для расчета факториала

```
int fakt(int n)
{int a;
  if( n<0 )return 0;
  if( n==0 )return 1;
  a=n*fakt (n-1);
  return a;
}
void main()
  { int m;
  cout <<"Введите целое число ";
  cin >> m;
  cout<<"Факториал числа "<<m<<" равен "<< fakt(m) <<"\n";
  }
```

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

1. Шилдт Г. Самоучитель C++. 3-е издание: Пер. с англ. – СПБ.: BHV – Санкт-Петербург, 1998. – 688 с.

2. Программирование на C++: Учебное пособие / В.Г. Аверкин, А.И. Бабровский; Под ред. А.Д. Хомоненко – СПБ.: Коронапринт, 1999. – 256 с.

3. Франко П. C++: Учебный курс – СПБ.: Питер, 2002. – 528 с.

4. Макаров В.Л. Программирование и основы алгоритмизации: Учебное пособие – СПб.: СЗТУ, 2003. – 110 с.

5. Программирование и основы алгоритмизации: методические указания / сост. : И.А. Елизаров, С.Б. Путин, С.А. Скворцов, А.А. Третьяков, С.И. Татаренко. – Тамбов: Изд-во Тамбовский государственный технический университет, 2007. – 40 с.

6. Давыдов, В.Г. Программирование и основы алгоритмизации: Учебное пособие – М.: Высш. шк., 2003. – 447 с.

7. Юркин А.Г. Задачник по программированию. – СПб.: «Питер», 2002. – 182с.

8. Павловская Т.А. C/C++. Программирование на языке высокого уровня. Учебник. – М.: «Питер», 2002. -460с.

9. Глушаков С.В., Коваль А.В, Смирнов С.В. Язык программирования C++. Учебный курс. – Харьков: «Фолио», 2001. – 500с.

10. Павловская Т.А., Щупак Ю.А. C\C++. Объектно-ориентированное программирование . – СПб.: «Питер», 2005. -264с.

Учебное текстовое электронное издание

Самарина Ирина Геннадьевна

ПРОГРАММИРОВАНИЕ И ОСНОВЫ АЛГОРИТМИЗАЦИИ ЧАСТЬ 1. КУРС ЛЕКЦИЙ

Учебное пособие

Издается полностью в авторской редакции
0,31 Мб
1 электрон. опт. диск

г. Магнитогорск, 2013 год
ФГБОУ ВПО «МГТУ»
Адрес: 455000, Россия, Челябинская область, г. Магнитогорск,
пр. Ленина 38

ФГБОУ ВПО «Магнитогорский государственный
технический университет им. Г.И. Носова»
Кафедра промышленной кибернетики и систем управления
Центр электронных образовательных ресурсов и
дистанционных образовательных технологий
e-mail: ceor_dot@mail.ru