**Базовые понятия языка C++**

**1.1. Введение в C++**

В настоящее время С++ является языком, наиболее полно представляющим основные парадигмы современного программирования. С++ сочетает в себе три различных принципа программирования:

* процедурное программирование, представленное языком С и позволяющее создавать библиотеки функций,
* объектно-ориентированное программирование (ООП), представленное таким понятием как класс и позволяющее разрабатывать библиотеки классов
* и обобщенное программирование, представленное шаблонами языка С++ и его стандартными библиотеками.

Язык С++ - замечательный язык, обладающий такими свойствами как эффективность, компактность, быстрота выполнения и переносимость программ. А три вышеперечисленные методики программирования С++ делают язык очень мощным, постоянно развивающимся в общем русле эволюции средств программирования и информатики.

Тот факт, что С++ включает в себя как ООП, так и обобщенное программирование наряду с традиционным процедурным подходом, свидетельствует о том, что в С++ ***упор делается на практичность***, а не на идеологию. В этом и состоит одна из причин успеха данного языка. Вторая причина успеха – это двойственность характера языка С++. Элементы языка С обеспечивают доступ к аппаратным средствам на низком уровне. А ООП и обобщенное программирование обеспечивает высокий уровень абстракции данных. Именно сочетанием этих свойств обусловлена популярность С++.

***История создания языка.***

В начале 70-х годов Денис Ритчи, сотрудник компании Bell Laboratories, занимался разработкой операционной системы UNIX. Операционная система – это совокупность программ, управляющих ресурсами компьютера и его взаимодействием с пользователем. Операционная система выводит на экран системные сообщения и управляет выполнением программ. Д. Ритчи нуждался в языке программирования, который мог обеспечить эффективное управление аппаратными средствами и создание компактных, быстро работающих программ. Такие потребности мог удовлетворить язык ассемблера, который тесно связан с внутренним машинным языком компьютера. Однако ассемблер – язык *низкого уровня*, то есть, привязан к определенному типу процессора. Операционная система UNIX предназначалась для работы на компьютерах различных типов (или платформах). А это предполагало использование языка высокого уровня. Язык *высокого уровня* ориентирован на решение задач, а не на конкретное аппаратное обеспечение. Специальные программы, которые называются *компиляторами*, транслируют программу, написанную на языке высокого уровня, в команды внутреннего языка конкретного компьютера. Таким образом, используя, отдельный компилятор для каждой платформы, одну и ту же программу на языке высокого уровня, можно выполнять на разных платформах. ***Нужен был язык, который сочетал бы в себе эффективность и возможность доступа к аппаратным средствам, обеспечиваемую языками низкого уровня, с более общим характером и переносимостью, присущими языкам высокого уровня. На базе имеющихся языков программирования Ритчи разработал язык С.***

###### ***Язык С++*** был создан в начале 80 годов также в компании Bell Laboratories **Бьерном Страуструпом** с целью повышения эффективности языка программирования. Страуструп выбрал в качестве основы язык С, так как он был кратким, хорошо подходил для системного программирования и был широко доступен. **Страуструп добавил в язык С элементы ООП и обобщенного программирования**, не меняя при этом существенно сам язык С. Таким образом, язык **С++ был разработан как расширение языка С**.

Язык С расширяется введением гибких и эффективных средств, предназначенных для построения новых типов. Программист структурирует свою задачу, определив новые типы, которые точно соответствуют предметной области задачи.

Такой метод построения программы обычно называют абстракцией данных. Ключевым понятием С++ является класс.

*Класс* - это определяемый пользователем тип.

Классы обеспечивают:

* упрятывание данных**,**
* ***их*** инициализацию**,**
* неявное преобразование пользовательских типов,
* динамическое задание типов**,**
* контролируемое пользователем управление памятью и
* средства для перегрузки операций.

***Различие между процедурными языками и объектно-ориентированными.***

В общем случае язык программирования базируется на двух основных понятиях – это данные и алгоритмы. Данные представляют собой информацию, которую программа обрабатывает. А алгоритмы – это методы, которые программа использует для обработки данных. Язык С как и большинство языков программирования того времени является ***процедурным*** – это означает, что основной аспект в нем делается на алгоритмах.

***Процедурное программирование*** заключается в том, что сначала определяется ***последовательность действий***, которая должна быть выполнена компьютером, а затем эти действия реализуются с помощью языка программирования. Программа содержит набор процедур, которые компьютер должен выполнить, чтобы получить требуемый результат.

Первые процедурные языки, такие как FORTRAN и BASIC столкнулись с проблемами организационного плана. Во многих старых программах благодаря операторам безусловного перехода алгоритм настолько запутан (их называют программы – «спагетти»), что его трудно читать. Для решения этой проблемы был разработан стиль программирования, который называется ***структурным программированием****.*

* Во-первых, структурное программирование ограничивает ветвление программы, в основном отказываясь от оператора безусловного перехода.
* Во-вторых, алгоритм строится, используя несколько основных структур: структура последовательного выполнения, структуры повторения с предусловием и с постусловием, структура - альтернатива и структура - ветвление.

Язык С поддерживает эти конструкции (операторы-выражения; циклы **for, while, do while**; оператор **if else** и оператор **switch**).

* В-третьих, еще одним новым принципом структурного программирования является проектирование программы ***сверху вниз****.*

Идея заключается в разбиении большой программы на более мелкие, легче решаемые задачи. При этом используются описанные выше структуры. Если полученные задачи по-прежнему остаются слишком обширными, их также следует разделить на более мелкие задачи, используя для алгоритма все те же структурные схемы. Этот процесс продолжается до тех пор, пока программа не будет разделена на маленькие, легко программируемые модули. Разработанные таким образом (***сверху вниз****)* алгоритмы обладает в некотором смысле свойством "правильности", встроенной в них шаг за шагом. Общая структура управления в структурированной программе является ***деревом****.* Такую программу легко читать сверху вниз, а, не прыгая по тексту из конца в начало. Методика структурного программирования отражает процедурный подход, при котором программа рассматривается с точки зрения выполняемых ею действий.

Хотя принципы структурного программирования позволили улучшить понятность и надежность программ, а также облегчить их сопровождение, создание программ больших размеров по-прежнему оставалось нелегкой задачей.

***Объектно-ориентированное программирование (ООП)*** предлагает новый подход к решению этой задачи.

В отличие от процедурного программирования, где главное внимание уделяется алгоритмам, в ООП ***основной акцент делается на данные***. Идея заключается в создании таких форм данных, которые соответствовали специфике поставленной задачи.

Спецификацией, описывающей подобную уникальную форму данных в языке С++ является ***класс****,* а конкретной структурой данных, созданной в соответствии с этой спецификацией, - ***объект***.

В общем случае ***класс*** определяет, какие данные будут представлять объект и какие операции могут выполняться над этими данными.Используя объекты классов, можно приступать к разработке самой программы. Такой процесс продвижения от более низкого уровня организации (классы) к более высокому уровню (программа), называется программированием ***снизу вверх****.*

***Объектно-ориентированное программирование – это объединение данных и методов в описании класса.***

***Сокрытие*** - ***инкапсуляция*** данных позволяет предохранить данные от нежелательного доступа.

***Полиморфизм*** дает возможность создавать множественные определения для операций и функций, а то, какое определение будет конкретно использоваться, зависит от контекста программы.

***Наследование***позволяет создавать новые классы из старых и наследовать данные и методы от классов – родителей.

Таким образом, в ООП используется иной подход к созданию программ в сравнении с процедурным программированием.

***Основное внимание уделяется не алгоритмическому аспекту задач, а созданию новых нужных форм данных, представляющих абстракции общих понятий.***

Кроме того, объектно-ориентированные языки дают возможность включать в программы уже существующие библиотеки классов.

***Понятие обобщенного программирования.***

***Обобщенное программирование*** *–* это еще одна парадигма программирования, поддерживаемая языком С++. Назначение обобщенного программирования такое же как ООП – упростить повторное использование кодов программ и методов абстрагирования общих понятий.

Однако в то время как в ООП основное внимание уделяется данным, ***в обобщенном программировании упор делается на шаблоны алгоритмов*** и у него другая область применения.

ООП – это инструмент для разработки больших программ, тогда как обобщенное программирование обеспечивает выполнение задач общего характера, таких как, например, сортировка данных или поиск.

***Обобщенное программирование – это создание кода программы независимого от типа данных***

В С++ имеются данные различных типов – целые числа, вещественные, символы, строки символов. Кроме того типы определенные пользователем, например, сложные структуры, состоящие из данных нескольких типов. Если требуется, например, сортировать данные различных типов, то обычно требуется создавать отдельную функцию сортировки для каждого типа.

Обобщенное программирование расширяет язык таким образом, что можно один раз написать функцию для обобщенного (то есть неопределенного) типа данных, а затем использовать ее для разнообразных реальных типов данных.

***Однако это можно обеспечить с и помощью шаблонов языка С++.***

***Шаблоны являются инструментами обобщенного программирования, но оно идет дальше по пути обобщения.***

Разнотипные данные могут быть объединены в обычный массив или составлять связанный список или быть элементами любого другого типа контейнеров. Цель обобщенного программирования создать одну функцию в данном случае - сортировки, которая работала бы с массивами, связанными списками, или любым другим типом контейнеров.

***Иначе говоря, функция обработки должна быть независимой не только от типов данных, содержащихся в контейнере, но и от самой структуры контейнера.***

Именно такое обобщенное представление в ***обобщенном программировании*** реализуется с помощью ***итераторов***. Назначение итератора – предоставить единый метод обработки элементов контейнера, не зависящий от вида контейнера и типа элементов в нем.

***Международный стандарт языка C++.***

Язык С++ с момента своего создания приобрел большую популярность у программистов и стал широко распространенным языком. Язык достиг определенного уровня зрелости и претерпел значительные изменения.

В течение многих лет велись работы по выработке стандартов языков С и С++. В 1983 году Национальным институтом стандартизации США (American National Standards Institute –ANSI) был утвержден первый стандарт языка C – ANSI C, который определил не только сам язык С, но и стандартную библиотеку С, которая должна быть включена во все реализации языка С.

В языке С++ тоже используется эта библиотека, обычно называемая стандартной библиотекой С или просто стандартной библиотекой. В дополнении к этому стандарт языка C++ должен представлять стандартную библиотеку классов языка С++.

Работа ANSI и Международной организацией по вопросам стандартизации (International Standards Organization – ISO) над стандартом языка С++ началась в 1990 году В 1998 году был утвержден стандарт ANSI/ISO С++, который согласуется со стандартом ANSI С .

Новый стандарт С был принят ISO и ANSI в 1999 году. Эта версия называется С99. Она включает ряд усовершенствований и несколько новых средств.

Затем в течение многих лет велись работы по выработке новых стандартов языков С и С++, которые завершились в конце 2011г.

**В октябре 2011г. вышел новый стандарт языка С++, обозначаемый как C++11. А в декабре 2011г. был принят новый стандарт и для языка С. Эту версию условно называют С11. Новые стандарты уже сейчас поддерживается свободно распространенным компилятором DJGPP.**

Компиляторы языка С++ разработаны практически для всех аппаратно-программных платформ. Программы на С++ транслируются в исполняемые модули, работающие под управлением операционных систем UNIX (и ее разновидностей, таких как, LINUX, Solaris), Windows, Mac OS. В отличие от языков C# (Си-Шарп), Java и Visual Basic язык C++ позволяет создавать программы, для выполнения которых не требуется устанавливать на компьютер специальное программное обеспечение, создающее среду исполнения программ.

Настоящее учебное пособие в основном будет ориентироваться на версию Borland C++ 3.11.

***Структура языка C++.***

Обобщенная структура языка С++ дана на рис. 1.

# С++

#### Модули

**(main, функции**)

## Типы данных

## Операторы

**обработки данных:**

**о. ввода/вывода**

**о. выражения**

**о. присваивания**

**о. вызовов функций**

**Средства объектно-ориентированного**

**программирования**

#### Сложные

#### Простые

**Массивы**

**Стандартные**

###### Классы

Объекты

**Структуры**

**Арифмет**.

**Организации обработки данных:**

**-о. передачи управления программой.**

**-о. ветвления**

**-о. циклов**

**-о. пустой**

**-о. составной**

###### Объекты

Объекты

**Объединения**

**Веществ**.

**Средства**

**обобщенного**

**программирования**

**Файлы**

**Целые**

**Символ**.

###### Контейнеры

Объекты

**Определенные пользователем**

###### Шаблоны

Объекты

**Перечисления**

###### Итераторы

Объекты

**Указатели**

**Ссылки.**

Рис. 1. Обобщенная структура языка С++

В левой части рисунка представлены средства языка, предназначенные для определения данных, объектов обработки программы. Типы данных определяют свойства данных, их внутреннее представление, возможные операции, которые можно производить с этими данными.

В следующей части рисунка представлены средства языка - операторы, предназначенные,

во-первых, для обработки данных путем, например, получения новых значений объектов программы в операторе присваивания;

во-вторых, для организации процесса обработки данных, например, организации повторяющейся обработки или организация разветвления процесса обработки.

Далее на рисунке представлены модули – относительно самостоятельные фрагменты программы для функционально законченной обработки данных, оформленные в виде функций.

В следующей части рисунка представлены средства и механизмы объектно-ориентированного программирования. Представлен новый тип данных, объединяющий данные и функции их обрабатывающие в единое целое – объект.

В последней части рисунка описаны средства обобщенного программирования, а именно множество контейнеров – структур данных, в которые можно помещать и извлекать данные любых типов, и набора обобщенных алгоритмов, позволяющих выполнять типовые операции над элементами контейнеров, не зависящие от вида контейнера. Абстракцию данных и алгоритмов обеспечивают шаблоны и итераторы.

***Методика создания программ.***

Процесс разработки программ на С++ предполагает разбиение процесса решения задачи на ряд этапов, выполняющих функционально законченную обработку данных и формирование соответствующих функций.

***В результате программа представляет собой совокупность функций, одна из которых главная, называемая*** main***.***

Главная функция может располагаться в любом месте программы, но где бы она не находиласьвыполнение программы начинается и заканчивается именно в главной функции. Для главной функции можно использовать только имя ***main*.**

Определение любой функции, в том числе и главной в С++ состоит из заголовка и тела функции:

***<тип возвращаемого функцией результата> <имя> (список параметров)***

***{ тело функции – последовательность действий функции }***

Приведем пример простой программы:

*#include <iostream.h>*

*int main ( )*

*{cout <*< *"Программа стартовала"<<endl;*

*return 0;*

*}*

В результате выполнения программы в консольном окне экрана выведется фраза: *Программа стартовала.*

В первой строке – команда (директива) препроцессора, обеспечивающая включение в программу средств работы со стандартными потоками ввода/вывода данных. Эти средства подключаются к программе при использовании заголовочного файла c именем *iostream.* Стандартным потоком вывода по умолчанию является вывод на экран дисплея. Стандартный поток ввода обеспечивает чтение данных с клавиатуры.

Вторая строка – это заголовок функции *main.* В общем случае функция С++ вызывается другой функцией, а заголовок функции описывает интерфейс между ней и той функцией, которая ее вызывает. Слово, стоящее перед именем описывает информацию, которую функция передает в вызывающую функцию. Заголовок главной функции описывает интерфейс между функцией *main( )* и операционной системой.

Стандарт языка С++ требует, чтобы определение функции *main* начиналось со следующего заголовка: *int main ( )* - слово *int* указывает, что функция *main()* возвращает целое значение*.*

Возвращаемое функцией***main()***значение должно быть равно нулю, если выполнение программы прошло успешно. Круглые скобки после ***main*** требуются в соответствии с синтаксисом заголовка любой функции. В них помещается необязательный для главной функции список параметров. В данном примере список пуст. Некоторые программисты используют следующий также ***допустимый*** заголовок: ***void main ( ),*** означающий, что функция не возвращает результата.

Тело функции – это заключенная в фигурные скобки последовательность описаний, определений и операторов функции. В теле данной программы описаний и определений нет, а есть только два оператора.

Первый из них: ***cout << "Программа стартовала"<<endl;***

где ***cout*** - имя стандартного выходного потока. Данные для вывода передаются потоку с помощью операции **<<**. То, что нужно вывести, помещается от операции **<<** справа. В данном случае это строка – ***"Программа стартовала",*** заключенная в кавычки последовательность символов. Вслед за строкой помещается еще одна операция вывода **<<,** а затем манипулятор ***endl*** (сокращение от "end of line" – "конец строки"). Его роль - очистить буфер выходного потока и поместить в выходной поток символ перехода на новую строку.

Второй оператор в программе ***return 0*** – оператор возврата. Он завершает выполнение программы и передает в точку ее вызова значение выражения, стоящего в операторе. Так как программа "запускается" на исполнение по команде операционной системы, то возврат будет выполнен к операционной системе.

*Структура программы*

Простая программа на С++ состоит из следующих элементов:

1. препроцессорные директивы, например:

*# include <имя файла>// - включение текстов стандартных файлов*

*#define … // - замены в тексте*

1. объявление глобальных объектов программы (типов, переменных, констант);
2. объявление одной главной функции *main ()*;
3. объявление ряда неглавных функций
4. комментарии.

***Этапы создания исполняемого кода программы.***

Рассмотрим технологию подготовки программ.

Классическая схема подготовки исполняемой программы приведена на рис. 2.

Подготовка программы начинается с редактирования файла, содержащего текст программы, который имеет расширение ".cpp".

**Исходный текст программы**

*\* . cpp prog.cpp*

*Directory (рабочий каталог)*

*\*.h include directory*

##### Препроцессор

**Включаемые файлы**

**Единица трансляции**

*\* . ii* *prog.ii*

##### Компилятор

(Compiler)

Лексический анализ

Синтаксический анализ

Семантический анализ

Генерация кода, оптимизация

*\* . obj prog.obj*

**Объектный код программы программы**

*Output directory*

**Код начальной загрузки программы**

##### Компоновщик

(Linker)

*Library directory*

**Код из библиотечных файлов**

**Исполняемый код программы**

*\* . exe prog.exe*

*Output directory*

**Рис. 2. Схема подготовки исполняемой программы**

Перед шагом компиляции показан этап препроцессорной обработки текста программы. В нашем примере препроцессор обрабатывает директиву *#include <iostream.h>*  и подключает к исходному тексту программы средства ввода/вывода. Результат препроцессорной подготовки при включении специальной опции компилятора помещают в файл с расширением *".ii".* Препроцессор сформирует полный текст программы – единицу трансляции (translation unit).

Затем выполняется компиляция программы, которая включает в себя несколько фаз: лексический, синтаксический, семантический анализ, генерация кода и его оптимизация. В результате компиляции получается объектный модуль - некий "полуфабрикат" готовой программы. Файл объектного модуля имеет стандартное расширение ".obj".

Компоновка (сборка) программы заключается в объединении одного или нескольких объектных модулей программы и объектных модулей, взятых из библиотечных файлов и содержащих стандартные функции. В результате получается исполняемая программа в виде отдельного файла (загрузочный модуль и программный файл) со стандартным расширением -".exe", который затем загружается в память и выполняется.

**1.2. Лексические основы языка.**

***Алфавит*** — это тот набор знаков (символов), который допустим в данном языке.

В алфавит языка С++ входят 96 символов.

* Из них 91 изображаемых символов:
* прописные латинские буквы А..Z;
* строчные латинские буквы а..z;
* арабские цифры 0..9;
* символ подчеркивания \_ (рассматривается как буква).

Эти символы используются для образования ключевых слов и имён языка. В языке С++ прописные и строчные буквы различаются.

* В алфавит входят также 28 специальных символов:

**, . ; : ? ' ! | / \ ~ \* ( ) { } < > [ ] # % & ^ – = " +**

* А также 5 – неизображаемых символов:
* обобщенные пробельные символы (пробел, горизонтальная и вертикальная табуляция, перевод страницы, начало новой строки).

***Комментарии***

Комментарий — это последовательность любых знаков (символов) компьютера, которая используется в тексте программы для её пояснения. Обычно в тексте программы делают вводный комментарий к программе в целом (её назначение, автор, дата создание и т.д.), а далее дают комментарии к отдельным фрагментам текста программы, смысл которых не является очевидным. Компилятор языка программирования игнорирует комментарии, они нужны только для человека. В языке С++ имеется два вида комментариев: однострочные и многострочные.

***Однострочный комментарий*** начинается с символов **//** (две косые черты). Всё, что записано после этих символов и до конца строки, считается комментарием. Например:

**//*Это текст однострочного комментария***

***Многострочный комментарий*** начинается парой символов **/\*** (косая черта и звёздочка) и заканчивается символами **\*/** (звёздочка и косая черта). Текст такого комментария может занимать одну или несколько строк. Всё, что находится между знаками **/\*** и **\*/**, считается комментарием. Например:

***/\* Это текст большого***

***многострочного комментария \*/***

В комментариях символы - это не только литеры из алфавита языка С++, но и любые возможные символы, включая русские буквы.

***Лексемы языка***

Из изображаемых символов алфавита формируются лексемы (tokens) языка. Лексемы - последовательности символов исходного кода программы, имеющие определенное смысловое значение.

В языке С++ имеются следующие категории лексем:

* **идентификаторы (identifier);**
* **ключевые (зарезервированные) слова (keyword);**
* **знаки операций (operator);**
* **константы – литералы (literal);**
* **разделители (знаки пунктуации – punctuator).**

Рассмотрим эти лексические элементы языка подробнее.

***Идентификаторы***

Идентификатор — это имя программного объекта. В идентификаторе могут использоваться латинские буквы, цифры и знак подчеркивания. Прописные и строчные буквы различаются, например, sysop и SYSOP — два различных имени. Первым символом идентификатора может быть буква или знак подчеркивания, но не цифра. Пробелы внутри имен не допускаются. Длина идентификатора по Стандарту не ограничена, но некоторые компиляторы и компоновщики налагают на нее ограничения. Например, компиляторы фирмы Borland различают не более 32-х первых символов. При выборе идентификатора необходимо иметь в виду следующее:

* идентификатор не должен совпадать с ключевыми словами (см. следующий раздел) и с именами используемых стандартных объектов языка;
* не рекомендуется начинать идентификаторы с одного или двух символов подчеркивания, поскольку они могут совпасть с именами системных функций или переменных.

***Ключевые (служебные) слова***

Ключевые слова — это идентификаторы, зарезервированные в языке для специального использования. Эти идентификаторы имеют специальное значение для компилятора. Их можно использовать только в том смысле, в котором они определены. Список ключевых слов С++ приведен в табл. 1.1.

Таблица 1.1

*Список ключевых слов С++*

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

***Константы – литералы и перечисления***

В языке С++ существует несколько видов констант: константы – литералы (неименованные константы), именованные константы, константы перечислений и препроцессорные константы. Рассмотрим константы литералы – лексемы языка.

Константа – литерал – это лексема, представляющая изображение фиксированного числового, символьного или строкового значения. Константы – литералы делятся на пять групп:

**целые (integer literal);**

**вещественные (floating literal – с плавающей точкой);**

**логические (boolean literal - булевские);**

**символьные (character literal - литерные);**

**строковые (string literal - строки**).

Компилятор, выделив константу в качестве лексемы, "по внешнему виду" относит ее к определенной группе, а внутри группы по форме записи и по числовому значению – к тому или иному типу данных.

***Целые константы*** могут быть ***десятичными, восьмеричными и шестнадцатеричными*.**

***Десятичная целая константа*** определена как последовательность десятичных цифр, начинающаяся не с нуля, если это не число нуль, например: 16, 897216, 0, 21. *Отрицательные константы* - это константы без знака, к которым применена операция изменения знака.

***Восьмеричные целые константы*** начинаются всегда с нуля, например 016 имеет десятичное значение 14.

***Шестнадцатеричная константа*** *-* это последовательность шестнадцатеричных цифр, которой предшествует 0x. В набор шестнадцатеричных цифр кроме десятичных входят латинские буквы от ***а*** (или ***А***) до ***f*** (или ***F***). Таким образом, 0х16 имеет десятичное значение 22, a 0xF - десятичное значение 15.

Диапазон допустимых целых положительных значений - от **0** до **4294967295**. Константы, превышающие указанное максимальное значение, вызывают ошибку на этапе компиляции. Абсолютные значения отрицательных констант не должны превышать **2147483648**.

В зависимости от значения целой константы, компилятор по-разному представляет её в памяти ЭВМ. О форме представления данных в памяти ЭВМ говорят, используя понятие ***тип данных****.*

Соответствие между значением целых констант и автоматически выбираемыми для них типами данных отражено в табл. 1.2 для компиляторов семейства IBM PC/XT/AT (16-ти разрядных компиляторов).

Таблица 1.2

*Целые константы и выбираемые для них типы*

*Диапазоны значений констант*

|  |  |  |  |
| --- | --- | --- | --- |
| **Десятичные** | **Восьмеричные** | **Шестнадцатеричные** | **Тип**  **данных** |
| **от 0 до 32767** | **от 00 до 077777** | **от 0х0000 до 0х7FFF** | **int** |
|  | **от 0100000**  **до 0177777** | **от 0х8000**  **до 0xFFFF** | **unsigned int** |
| **от 32768**  **до 2147483647** | **от 0200000**  **до 017777777777** | **от 0x10000**  **до 0x7FFFFFFF** | **long** |
| **от 2147483647**  **до 4294967295** | **от 020000000000**  **до 037777777777** | **от 0x80000000**  **до 0xFFFFFFFF** | **unsigned long** |
| **> 4294967295** | **> 037777777777** | **> 0xFFFFFFFF** | **ошибка** |

В заголовке <limit.h> определены предельные значения целых величин различных типов. Младший тип целых констант – ***int*** имеет диапазон допустимых значений от – **32767** до **32767 .**

Для 32-разрядных компиляторов диапазон значений типа ***int*** обычноравен -**2147483647** и **2147483647.** Для типа ***unsigned int*** ("беззнаковый целый") минимальное значение равно 0, а максимальное - **4294967295.** Предельные значения целых констант даны в заголовке<climit.h>. И в реализациях 32-разрядных компиляторов, в которых не различаются типы ***int* -** "целое"и ***long* -**"длинное целое" при использовании целых чисел, превышающих значение **4294967295**, возникает ошибка.

Если программиста по каким-либо причинам не устаревает тот тип, который компилятор приписывает константе, то он может явным образом изменить тип. Можно явно указать тип, используя суффиксы ***L, l (long)*** и ***U, u* *(unsigned)****.* Например, константа ***64L*** будет иметь тип ***long***, хотя значению 64 должен быть приписан тип ***int***, как это видно из табл. 1.2. Для одной константы можно использовать два суффикса в произвольном порядке. Например, константы ***0x22ul, 0x33Lu*** будут иметь тип ***unsigned long***.

***Перечислимые константы****.* Стандарт относит к целочисленным константам и перечислимые. Перечислимые константы (или константы перечисления) определяются с помощью служебного слова ***enum***. Это целочисленные константы, которым даются уникальные имена по следующему правилу:

***enum { список идентификаторов констант = значения констант }***

Имена констант уникальны, но значения могут повторяться. Кроме того, значения в определении можно опускать. Константы автоматически получат значения, которые будут начинаться от нуля, а затем увеличиваться на 1 для следующей константы. Возможны определения, в которых присутствуют определения констант со значением и без значения. В этом случае константы без значения автоматически получают значения. Правило о последовательном увеличении на 1 значения констант действует и в этом случае. Например, определение:

***enum { a, b =0, c, d, e=2, f };***

вводит следующие константы:

***a = =0, b = =0, c= =1, d = =2, e = =2, f = =3.***

***Вещественные константы***.

Для представления вещественных чисел используются константы, представляемые в памяти компьютера в форме с плавающей точкой. Даже не отличаясь от целой константы по значению, вещественные константы имеют другую форму внутреннего представления в ЭВМ. При операциях с такими константами требуется использование арифметики с плавающей точкой. Компилятор распознает вещественную константу по внешним признакам.

Вещественная константа может включать следующие шесть частей: целая часть (десятичная целая константа); десятичная точка; дробная часть (десятичная целая константа); признак экспоненты "**е**" или "**Е**"; показатель десятичной степени (десятичная целая константа, возможно, со знаком); суффикс ***F*** (или ***f***), либо ***L*** (или ***l***).

При записи констант с плавающей точкой могут опускаться целая или дробная часть (но не одновременно); десятичная точка или символ экспоненты с показателем степени (но не одновременно). Примеры констант с плавающей точкой:

**125. .0 .17 3.141F 1.2е-5 .314159Е25 2.77 2E+6L**

Вещественная константа в экспоненциальном формате представляется в виде мантиссы и порядка. Мантисса записывается слева от знака экспоненты (E или e), порядок — справа от знака. Значение константы определяется как произведение мантиссы и возведенного в указанную в порядке степень числа 10. Обратите внимание, что пробелы внутри числа не допускаются, а для отделения целой части от дробной используется не запятая, а точка.

При отсутствии суффиксов вещественное число имеет форму внутреннего представления, которой соответствует тип данных ***double***. Добавив суффикс ***F*** или ***f***, константе придают тип ***float*** и соответственно, тип ***long double,*** если в ее конце суффиксы ***L*** или ***l***. В табл. 1.3 приведены диапазоны возможных значений и длины внутреннего представления (размеры в битах) данных вещественного типа в конкретной реализации С++ (компилятор DJGPP).

Таблица 1.3

*Данные вещественного типа*

|  |  |  |
| --- | --- | --- |
| Тип данных | Размер в битах | Диапазон значений |
| ***float*** | 32 | от 3.4E-38 до 3.4E+38 |
| ***double*** | 64 | от 1.7E-308 до 1.7E+308 |
| ***long double*** | 96 (80) | от 3.4E-4932 до 1.1E+4932 |

***Булевские (логические) константы.***

Это два литерала типа ***bool: true*** (ИСТИНА) и ***false*** (ЛОЖЬ).Тип ***bool*** и литералы ***true*** и ***false*** были добавлены в последних версиях Стандарта языка. Наряду с логическими литералами продолжают действовать правила, унаследованные из ранних версий языка, в соответствии с которыми значению ЛОЖЬ соответствует число 0, а любое отличное от нуля значение воспринимается в логическом выражении как ИСТИНА.

***Символьные (литерные) константы***

Символьные константы – это один или два символа, заключенные в апострофы. Символьные константы, состоящие из одного символа, занимают в памяти один байт и имеют стандартный тип ***char***. Примеры: **'z', '\*', '\012' *,* '\0', '\n'**– односимвольные константы. Двухсимвольные константы занимают два байта и имеют тип ***int,*** при этом первый символ размещается в байте с меньшим адресом (о типах данных рассказывается в следующем разделе). **'ab'**, **'\x07\x07', '\n\t'** – двухсимвольные константы

Внутри апострофов может быть любой символ, имеющий изображение. Однако в ПК есть символы, не имеющие графического изображения. Это, как правило, управляющие символы, например символ перехода на новую строчку. Для изображения таких символов используется комбинация из нескольких символов, начинающаяся с обратной косой черты.

Последовательности символов, начинающиеся с обратной косой черты, называются управляющими эскейп - последовательностями (escape-sequence).

Эскейп - последовательности используются:

* для записи символов, не имеющих графического изображения (например, \a — звуковой сигнал);
* для записи символов**:** символа апострофа ( **'** ), обратной косой черты **( \** ), знака вопроса ( **?** ) и кавычки ( **"** );
* для записи любого символа с помощью его шестнадцатеричного или восьмеричного кода, например, \073, \0xF5. Числовое значение кода должно находиться в диапазоне от 0 до 255.

В табл. 1.4 приведены допустимые значения эскейп-последовательностей. Управляющая последовательность интерпретируется как одиночный символ. В таблице **000** – строка от одной до трех восьмеричных цифр, **hh** – строка из одной или двух шестнадцатеричных цифр. Последовательность **'\0'** обозначает пустую литеру.

Таблица 1.4

*Допустимые ESC–последовательности в языке С++*

|  |  |  |  |
| --- | --- | --- | --- |
| Изображе-  ние | Внутренний  код | Обозначаемый символ | Действие или  смысл |
| **'\a'** | 0x07 | bel (audible bell) | звуковой сигнал |
| **'\b'** | 0x08 | bs (backspace) | возврат на шаг(забой) |
| **'\f '** | 0x0C | ff (form feed) | перевод страницы |
| **'\n'** | 0x0A | lf (line feed) | перевод строки (LF) |
| **'\r'** | 0x0D | cr (carriage return) | возврат каретки(CR) |
| **'\t** | 0x09 | ht (horizontal tab) | горизонтальная табуляция |
| **'\v'** | 0x0B | vt (vertical tab) | вертикальная табуляция |
| **'\\'** | 0x5C | \ (backslash) | обратная черта |
| **'\''** | 0x27 | ' (single quote) | апостроф |
| **'\"'** | 0x22 | '' (double quote) | кавычка |
| **'\?'** | 0x3F | ? (question mart) | вопр. знак |
| **'\000'** | 000 | octal number | Восьмеричный код символа |
| **'\0xhh'** | hh | hex number | Шестнадцатеричный код |

Для использования внутренних кодов символов нужна таблица, в которой каждому символу компьютера соответствует числовое значение его кода в десятичном, восьмеричном и шестнадцатеричном представлении. На IBM-совместимом ПЭВМ применяется таблица кодов ASCII.

Работу с символьными константами иллюстрирует программа:

*#include<iostream.h>*

*void main()*

*{char c; int i;*

*c = 'ab'; cout << c << '\t';*

*i =* '*ab*'*; cout << i <<'\t';*

*i = c; cout << i << '\t';*

*i =* '*ba*'*;*

*c = i; cout << c << '\t';}*

В результате программы на экран будет выведено:

*a 25185 97 b*

здесь 25185 – двухбайтовое целое число, а 97 – код символа *'a*'.

Если выводить на экран односимвольную константу, то будет выведено изображение символа, но если эту же константу поместить, например, в арифметическое выражение, то значением константы будет ее десятичный внутренний код.

Для 32-разрядного компилятора допустимы константы – несколько символов, заключенных в апострофы, которые называются мультисимвольными (multicharacter literal) и имеют тип *int.*

*Строковые константы*

Строка или строковая константа- это последовательность символов, заключенная в кавычки.

Внутреннее представление строки в памяти таково: все символы размещаются подряд, и каждый символ занимает 1 байт, в котором размещается внутренний код символа. А в конце строки компилятор помещает еще один символ, называемый байтовым нулем '\0'. Этот символ как любой другой занимает в памяти 1 байт, 8 двоичных разрядов, в которых находятся нули.

Среди символов строковой константы могут быть эскейп-последовательности, например:

*''Монография \'' Турбо – Паскаль\''. ''*

Строки, записанные в программе подряд или через пробельные символы, при компиляции конкатенируются (склеиваются). Таким образом, в тексте программы последовательность из строк:

*''Миру - '' ''мир!''*

эквивалентна одной строке: *''Миру – мир!''*

Длинную строковую константу можно размещать на нескольких строках в программе, используя еще и символ переноса строк - '\'

В строке может быть один символ, например, ''A'', которая в отличие от символьной константы 'A' занимает в памяти 2 байта. Строковая константа может быть пустой '''', при этом ее длина равна 1 байту Символьная константа не может быть пустой, запись '' - не допустима.

Кроме непосредственного использования строк в выражениях, строку можно поместить в символьный массив, например, при его инициализации и обращаться к ней по имени массива (раздел 2).

*Знаки операций*

*Знаки операций* – это один из элементов выражений. Выражения есть правило получения значения. Результат операции - это всегда значение.

Знак операции — это один или более символов, определяющих действие над операндами. Внутри знака операции пробелы не допускаются. Операции делятся на унарные, бинарные, тернарные по количеству участвующих в них операндов. Один и тот же знак может интерпретироваться по-разному в зависимости от контекста.

В табл. 1.5 представлены операции, приоритеты (ранги) и ассоциативность операций.

Таблица 1.5

*Операции языка С++ и их приоритеты*

|  |  |  |
| --- | --- | --- |
| Ранг | Операции | Ассоциативность |
| 1 | :: . -> ( ) [ ] |  |
| 2 | ! ~ + - ++ -- & \* (тип) sizeof  new delete тип ( )  typeid dynamic\_cast static\_cast reinterpret\_cast const\_cast |  |
| 3 | .\* ->\* |  |
| 4 | \* / % (мультипликативные) |  |
| 5 | + – (аддитивные) |  |
| 6 | << >> (сдвиги) |  |
| 7 | < <= >= > (сравнения) |  |
| 8 | == != (сравнения) |  |
| 9 | & (поразрядная конъюнкция) |  |
| 10 | ^ (поразрядное исключающее ИЛИ) |  |
| 11 | | (поразрядная дизъюнкция) |  |
| 12 | && (логическая конъюнкция) |  |
| 13 | || (логическая дизъюнкция) |  |
| 14 | ? : (условная операция) |  |
| 15 | = \*= /= %= += -= &= ^= |= <<= >>=  операция присваивания |  |
| 16 | throw |  |
| 17 | , (операция запятая) |  |

Кроме стандартных режимов использования операций язык С++ допускает расширение (перегрузку) их действия, дает возможность распространения действия на объекты классов. Примером такой перегрузки являются операции поразрядных сдвигов >> и <<. Когда слева от них в выражениях находятся входные и выходные потоки, они трактуются как операции извлечения данных из потока >> и вывода данных в поток <<.

В табл. 1.6 дано краткое описание стандартных операций языка С++ .

Таблица 1.6

*Сводка стандартных операций С++*

|  |  |
| --- | --- |
| Унарные операции | Форма |
| 1. & - операция получения адреса  некоторого объекта программы | & lvalue  (lvalue – имя объекта программы) |
| 2. \* - разыменование, доступ по адресу к значению объекта | \* a  ( а – указатель на объект) |
| 3. – - унарный минус | – выражение |
| 4. + - унарный плюс | + выражение |
| 5**. !** – логическое отрицание (НЕ)  **false –** если операнд истинныйи **true –** если операнд ложный | !выражение  !1 ==0, !(-5)==0(ложь)  !0 == 1 (истина) |
| 6. **++** - инкремент, увеличение значения операнда на 1   * префиксная форма - до использования его значения; * постфиксная форма – после использования его значения; | ++ lvalue  lvalue ++ |
| **7.** -- - декремент, уменьшение значения операнда на 1   * префиксная форма; * постфиксная форма | -- lvalue  lvalue-- |
| 8. **sizeof** – размер в байтах внутреннего представления объекта | sizeof выражение  sizeof (тип ) |
| 9. **new** – динамическое выделение памяти | new имя\_типа  new имя\_типа инициализатор |
| 10. **delete** – освобождение динамически выделенной памяти | delete указатель  delete [] указатель |
| 11. **()** - явное преобразование типа | (тип) выражение |
| 12. **()** - функциональная форма преобразования типа | тип (выражение)  тип имеет простое имя |
| 13. **typeid** – операция определения типа операнда | typeid (выражение)  typeid(имя\_типа) |
| 14. **dynamic\_cast -** операция приведения типа с проверкой допустимости при выполнении программы | dynamic\_cast <целевой тип> (выражение) |
| 15. **static\_cast –** операция приведения типов с проверкой допустимости приведения во время компиляции. | static\_cast <целевой тип> (выражение) |
| 16. **reinterpret\_cast –** операция приведения типов без проверки допустимости приведения. | reinterpret\_cast <целевой тип> (выражение) |
| 17. **const\_cast –** операция приведения типов, которая аннулирует действие модификатора const | const\_cast <целевой тип> (выражение) |
| 18**. :: -** операция доступа из тела функциик внешнему объекту | :: имя\_объекта |
| **Бинарные операции** | Форма |
| **Аддитивные:** |  |
| 19. + - сложение арифметических операндов, или сложение указателя с целочисленным операндом | выражение + выражение |
| 20. **–** - вычитание арифметических операндов или указателей | выражение – выражение |
| **Мультипликативные** |  |
| 21. \* -умножение арифметических операндов | выражение \* выражение |
| 22. / -деление операндов арифметических типов. При целых операндах дробная часть результата отбрасывается | выражение / выражение  20/6 равно 3, -20/6 равно -3,  20/(-6) равно -3 |
| 23. % -получение остатка от целочисленного деления | выражение % выражение  13%4 равно 1, (-13)%4 равно -1  13%(-4) равно 1,(-13)%(-4) равно -1 |
| **Операции сдвига** |  |
| 24. << - сдвиг влево битового представления левого целочисленного операнда на количество разрядов равное значению правого операнда | lvalue << выражение |
| 25. >> - сдвиг вправо битового представления левого целочисленного операнда на количество разрядов равное значению правого операнда | lvalue >> выражение |
| **Поразрядные операции** |  |
| 26. & поразрядная конъюнкция (И) битовых представлений целочисленных операндов | выражение& выражение |
| 27. | - поразрядная дизъюнкция (ИЛИ) битовых представлений целочисленных операндов | выражение | выражение |
| 28. ^ - поразрядное исключающее ИЛИ битовых представлений целочисленных операндов | выражение ^ выражение |
| **Операции отношений (сравнения)**  Результат**: true (1)** , если сравнение истиннои **false (0)–** если ложно |  |
| 29. < - меньше | выражение < выражение |
| 30. < = - меньше или равно | выражение <= выражение |
| 31. > - больше | выражение > выражение |
| 32. > = - больше или равно | выражение >= выражение |
| 33. = = - равно | выражение = = выражение |
| 34. ! = - не равно | выражение != выражение |
| **Логические бинарные операции**  Результат**: true (1)** - истинна  и **false (0) –** ложь |  |
| 35. && - конъюнкция (логическое И) скалярных операндов и отношений | выражение && выражение |
| 36. || - дизъюнкция (логическое ИЛИ) скалярных операндов и отношений | выражение || выражение |
| **Операции присваивания** |  |
| 37. = простое присваивание левому операнду значения выражения - операнда из правой части | lvalue = выражение |
| 38 операция = - составное присваивание левому операнду результат операции между левым и правым операндом; операциb **\* / % + - << >> & | ^** | lvalue операция = выражение  эквивалентно  lvalue = lvalue операция (выражение) |
| **Операции доступа к компонентам структурированного объекта** |  |
| 39. **.** (точка) – прямой доступ к компоненту | имя\_объекта.имя\_компонента |
| 40. **–>** косвенный доступ к компоненту структурированного объекта, адресуемого указателем | указатель\_на\_объект–>имя\_ компонента |
| **Операции доступа к адресуемым компонентам класса** |  |
| 41. **.\*-** прямое обращение к компоненту класса по имени объекта и указателю на компонент | имя\_объекта .\* указатель\_ на\_компонент |
| 42. **–>\*** косвенное обращение к компоненту класса через указатель на объект и указатель на компонент | указатель\_на\_объект –>\* указатель\_на\_компонент |
| 43. **:: -** бинарная операциярасширения области видимости | имя\_класса :: имя\_компонента  имя\_пространства\_имен :: имя |
| 44. ( ) – операция круглые скобки – операция вызова функции | имя\_функции (список\_аргументов) |
| 45. [ ] - операция квадратные скобки - индексация элементов массивов | имя\_массива [индекс] |
| 46. **? :** - условная (тернарная) операция; | выражение\_1 ? выражение\_2 : выражение\_3  если выражение\_1 истинно, то значением операции является значение выражения\_2, если выражение\_1 ложно, то значением операции является значение выражения\_3 |
| 47. **,** - операция запятая – это несколько выражений, разделенных запятыми, вычисляются последовательно слева направо; результатом операции является результат самого правого выражения | (список\_выражений) |

***Операция и выражение присваивания***

Операция присваивания обозначается символом '='. Простейший вид операции присвоения:

***l = v***

Здесь ***l*** - выражение, которое может принимать значение, ***v*** - произвольное выражение.

Операция присвоения выполняется справа налево, т.е. сначала вычисляется значение выражения ***v***, а затем это значение присваивается левому операнду ***l***.

Левый операнд в операции присваивания должен быть так называемым **адресным выражением**, которое иначе называют ***lvalue***. Примером адресного, или именующего выражения, является имя переменной. Адресным выражением никогда не являются константы. Не является ***lvalue*** и простое выражение, например, выражение ***a+b.*** Адресное выражение это объект, представляющий некоторый именованный участок памяти, в который можно поместить новое значение.

В языке C++ операция присваивания образует выражение присваивания, т.е. ***a = b*** означает не только засылку в ***a*** значения ***b***, но и то, что ***a = b*** является выражением, значением которого является левый операнд после присвоения. Отсюда следует, что возможна, например, такая запись:

***a = b = c = d = e + 2;***

Если тип правого операнда не совпадает с типом левого, то значение справа преобразуется к типу левого операнда (если это возможно). При этом может произойти потеря значения, например:

***int i; char ch; i=3.14; ch=777;***

Здесь ***i*** получает значение 3, а значение 777 слишком велико, чтобы быть представленным как ***char***, поэтому значение ***ch*** будет зависеть от способа, которым конкретная реализация производит преобразование из большего в меньший целый тип.

Существует так называемая комбинированная операция присваивания вида: ***a оп =b****,* здесь ***оп*** - знак одной из бинарных операций:

***+ - \* / % >> << & | ^ && ||.***

Присваивание ***a оп = b*** эквивалентно ***a = a оп b***.

***Разделители***

*Разделители или знаки пунктуаций* входят в состав лексем.

* *Квадратные скобки* ***[]*** – ограничивают индексы массивов и номера индексированных элементов:

***int A[5] ={ …}; A[3]=5****;*

* *Круглые скобки* ***( )****:* 
  + выделяют условное выражение в условном операторе*:* ***if (x<0) x = -x;***
  + обязательный элемент в определении, описании и вызове функций:

***float F(float x, int n)*** //определение функции

***{тело функции}***

***float F(float , int );* //** описании функции

***F(3.14, 10);* //** вызов функции

* обязательны в определении указателя на функцию и в вызове функции через указатель:

***int (\*pointer) ( );*** //определение указателя

***(\* pointer) ( );* //**вызов функции через указатель на функцию

* + применяются для изменения приоритета операций в выражениях
  + элемент оператора цикла:

***for(int i =0, j = 3; i < j ; i+ =2, j++)***

***{тело цикла};***

* + используются при преобразовании типа:

***(имя типа) операнд ; имя типа(операнд);***

* + в макроопределениях, обрабатываемых препроцессором:

***#define имя(список параметров) (строка замещения)***

* *Фигурные скобки* ***{ }****:*
  + Обозначают начало и конец составного оператора или блока.

Пример составного оператора в условном операторе:

***if (x<y) {x++; y--};***

Пример блока, являющегося телом функции:

***float s (float a, float b)***

***{return a+b;}***

После закрывающей скобки **'}'** не ставится точка с запятой **';'.**

* Используются для выделения списка компонент структур, объединений и классов:

***struct st { char\*b; int c;};***

***union un {char s [2]; unsigned int i; };***

***class m {int b; public : int c, d; m(int);};***

Точка с запятой **';'** обязательна после определений каждого типа.

* Используются для ограничения списков инициализации массивов, структур, объединений, объектов классов при их определении:

***int k[ ] = { 1,2,3,4,5,6,7};***

***struct tt {int ii; char cc;} ss ={777 , ‘\n’};***

* *Запятая* ***','***
* разделяет элементы списков формальных и фактических параметров функций;
* разделяет элементы списков инициализации структурированных объектов;
* разделитель в заголовке цикла ***for*:**

***for(int x=p, y=q, i=2; i < 100; z=x+y, x=y, y=z, i++)***

Запятую - разделитель следует отделять от запятой- операции с помощью круглых скобок:

***int i=1; int m [ ] = { i, ( i=2 , i \* i ) ,i };***

* *Точка с запятой* ***';'*** – завершает каждый оператор и пустой в том числе.
* *Двоеточие* ***':'***
* служит для отделения метки от оператора: ***метка: оператор****;*
* при описании производного класса:

***class x : A, B {список компонентов};***

* + *Многоточие* ***'…'*** – используется для обозначения переменного числа параметров у функции
  + *Звездочка* ***'\*'*** используется как разделитель в определении указателя:

***int\* r;***

* *Знак* ***'='*** при определении объектов программы с инициализацией отделяет имя объекта от инициализирующего значения.
* *Символ* ***'#'*** используется для обозначения директив препроцессора.
* *Символ* ***'&'*** играет роль разделителя при определении ссылок:

***int d; int &c = d;***