[6. Данные.](#_Toc515769973)

[6.1. Идентификаторы.](#_Toc515769974)

[6.2. Ключевые слова.](#_Toc515769975)

[6.3. Символы.](#_Toc515769976)

[6.4. Данные: переменные и константы.](#_Toc515769977)

[6.5. Данные: типы данных.](#_Toc515769978)

[6.5.1. Три целочисленных типа.](#_Toc515769979)

[6.5.1.1. Описание данных целого типа.](#_Toc515769980)

[6.5.1.2. Целые константы.](#_Toc515769981)

[6.5.1.3. Инициализация переменных целого типа.](#_Toc515769982)

[6.5.1.4. Модификатор unsigned.](#_Toc515769983)

[6.5.2. Числа с плавающей точкой.](#_Toc515769984)

[6.5.3. Описание переменных с плавающей точкой.](#_Toc515769985)

[6.5.4. Перечисляемый тип данных (enum).](#_Toc515769986)

[6.5.5. Модификаторы доступа.](#_Toc515769987)

[6.5.5.1. Модификатор const.](#_Toc515769988)

[6.5.5.2. Определение констант через #define.](#_Toc515769989)

[6.5.5.3. Модификатор volatile.](#_Toc515769990)

[6.5.5.4. Cовместное использование const и volatile.](#_Toc515769991)

[6.5.6. Модификаторы pascal, cdecl, near, far и huge.](#_Toc515769992)

[6.5.6.1. Модификатор pascal.](#_Toc515769993)

[6.5.6.2. Модификатор cdecl.](#_Toc515769994)

[6.5.6.3. Модификаторы near, far и huge.](#_Toc515769995)

[6.5.7. Тип данных char](#_Toc515769996)

[6.5.8. Другие типы и размеры данных.](#_Toc515769997)

[6.6. Преобразование типов данных.](#_Toc515769998)

[6.6.1. Явные преобразования типов при помощи операции приведения типа.](#_Toc515769999)

1. Данные.

Программы имеют дело с данными. Мы вводим в компьютер числа, буквы и слова и ожидаем, что он будет проводить над ними какие-то операции. Здесь мы сосредоточим наше внимание на данных различных типов и их свойствах. В соответствии с этим мы будем последовательно останавливаться на каждом из типов, и смотреть, как их можно использовать. Но, поскольку заниматься одним только обсуждением представляется нам не очень веселым делом, мы рассмотрим также небольшие программы обработки данных.

Этот материал в основном посвящен обсуждению двух важнейших классов типов данных: целым числам и числам с плавающей точкой. Язык Си предоставляет программисту возможность использовать несколько разновидностей этих типов. Мы займемся изучением следующих вопросов: что такое типы данных, как их описать, как и когда их использовать. Кроме того, мы обсудим различия между константами и переменными.

## Идентификаторы.

Идентификаторы — это имена, которыми в программе можно обозначать переменные, константы, типы, функции и метки. Идентификатор создается при его появлении в объявлении переменной, типа или функции. В последующих операторах программы идентификатор можно использовать для ссылки на связанный с ним программный объект. Идентификатор — это последовательность из одной или нескольких букв, цифр или символов подчеркивания, причем первой должна быть буква или символ подчеркивания. Идентификаторы могут иметь произвольное число символов, однако компилятор различает только первый 31 символ. (Однако другие программы, работающие с выходной информацией следом за компилятором, например, компоновщик, могут различать еще меньше символов.)

В С и C++ различаются заглавные и строчные буквы. Это значит, что компилятор С рассматривает буквы верхнего и нижнего регистров как различные символы. Например, компилятор считает переменные NAME\_LENGTH и Name\_Length двумя уникальными идентификаторами, представляющими различные ячейки памяти. Эта особенность позволяет создавать различные идентификаторы, имеющие одинаковое произношение, но различный регистр для одной или нескольких букв. Кроме этого, различимость заглавных и строчных букв облегчает понимание текста программы. К примеру: идентификаторы, объявленные в заголовочных файлах в #include, часто записываются только заглавными буквами. Поэтому, когда вы встречаете в исходном файле идентификатор, состоящий из заглавных букв, по его виду вы можете понять, где искать описание данного идентификатора.

Не следует создавать идентификаторы, начинающиеся с символов подчеркивания, — хотя это и не является ошибкой синтаксиса. Такие идентификаторы могут конфликтовать с именами системных процедур или переменных и вызывать ошибки. В результате, не гарантируется переносимость программ с подобными идентификаторами. Идентификаторы, имеющие два последовательных символа подчеркивания (\_), зарезервированы для языка C++ и для стандартных библиотек.

Имеется одно стилистическое соглашение, принятое многими программистами на С: в начале всех идентификаторов указывается сокращенное обозначение типа данных. Например, все идентификаторы для целых начинаются с буквы "i", для чисел с плавающей точкой — с буквы "f”, для строк, оканчивающихся символом null, — с букв "sz", для переменных- указателей — с "р" и так далее. При таком именовании легко по тексту программы видеть не только используемые идентификаторы, но и их тип. При таком подходе легче понять работу конкретного участка программы и выполнить построчную отладку исходного текста. В данной книге в программах используются оба соглашения по именованию переменных, тогда как в реальных программах можно встретить либо один, либо другой формат.

Примеры идентификаторов:

i

itotal

frangel

szfirst\_name

lfrequency

imax

iMax

iMAX

NULL

EOF

Сможете ли вы определить, почему следующие идентификаторы неверны?

1st\_year

#social\_security

Not\_Done!

Все неверны, т.к первый из них начинается с десятичного числа. Второй — начинается с символа #, а последний — заканчивается запрещенным символом.

Взгляните на следующие идентификаторы. Можно ли их использовать?

О

ОО

ООО

На самом деле все четыре идентификатора вполне корректны. В первых трех используется заглавная буква "О". Поскольку в каждом разное число букв, все они уникальные. Четвертый идентификатор состоит из четырех символов подчеркивания (\_). Имеет ли это смысл? Определенно нет. Правильна ли эта запись? Да. Хотя эти идентификаторы отвечают "букве закона", они в значительной мере не соответствуют "духу закона". Дело в том, что все идентификаторы, функции, константы и переменные должны иметь осмысленные (значащие) имена.

Поскольку заглавные и строчные буквы считаются разными символами, каждый из следующих идентификаторов уникален:

MAX\_RATIO

max\_ratio

Max\_Ratio

Такая ситуация может вызвать у начинающего программиста на С определенные трудности. Например, при попытке вызвать функцию printf() и напечатав PRINTF(), он получит от компилятора сообщение об ошибке "неизвестный идентификатор" (unknown identifier). Хотя в Паскале совершенно все равно, как писать: writeln, WRITELN или WriteLn. Имея опыт, вы, вероятно, обнаружите предыдущую ошибку с printf(); a сможете ли вы определить, что неправильно в следующем операторе?

printf ("%D", integer\_value);

Предположив, что integer\_value описана правильно, вы можете подумать, что все в порядке. Помните, однако, о различии в С заглавных и строчных букв — нет формата печати %D, есть только %d.

В более сложных приложениях некоторые компоновщики могут ограничить число и тип символов для глобально видимых обозначений. Кроме этого, компоновщик, в отличие от компилятора, не различает заглавные и строчные буквы. По умолчанию, компоновщик C/C++ LINK видит одинаковыми все символы public и external, записанные как MYVARIABLE, MyVariable и myvariable. Можно, однако, заставить LINK различать регистры, если использовать опцию /NOI. Тогда компоновщик будет считать три приведенных выше переменных уникальными. Используйте справочную утилиту PWB для получения дополнительной информации по применению этого ключа.

И последнее: идентификатор не может выглядеть так же, как ключевое слово языка. В следующем разделе перечислены ключевые слова С и C++.

## Ключевые слова.

Ключевые слова — это предопределенные идентификаторы, имеющие особое значение для компилятора C/C++. Их можно использовать только согласно описанию. Помните, что программные идентификаторы не могут совпадать по написанию с ключевыми словами C/C++. Ключевые слова C/C++ перечислены в таблице.

|  |  |  |
| --- | --- | --- |
| **Ключевые слова Microsoft Visual C/C++** | | |
| \_asm | \_far | \_saveregs |
| auto | \_fastcall | \_self |
| based | \_finally | \_segment |
| break | float | \_segname |
| case | for | short |
| \_cdecl | \_fortran | signed |
| char | goto | sizeof |
| const | \_huge | static |
| continue | if | \_stdcall |
| \_declspec | \_inline | struct |
| default | int | switch |
| dllexport | \_interrupt | thread |
| dllimport | \_leave | \_try |
| do | \_loads | typedef |
| double | long | union |
| else | naked | unsigned |
| enum | \_near | void |
| \_except | \_pascal | volatile |
| \_export | register | while |
| extern | return |  |

|  |  |  |
| --- | --- | --- |
| **Ключевые слова языка С++** | | |
| class | operator | virtual |
| delete | private | \_multi\_inheritance |
| friend | protected | \_single\_ inheritance |
| inline | public | \_virtual\_ inheritance |
| new | this |  |

Далее следуют не ключевые слова; однако, они имеют особое значение в Microsoft С или C++

|  |  |  |
| --- | --- | --- |
| argc | envp | \_setenvp |
| argv | main | *\_*set*\_*new\_handler |
| \_emit | \_setargv |  |

Ключевые слова нельзя переопределять. Однако, при помощи директив препроцессора С можно задать текст, который перед началом компиляции будет заменен ключевыми словами.

## Символы.

В каждом языке для построения значащих операторов используется некоторый набор символов. К примеру, все книги, написанные на английском языке, используют 26 букв алфавита, 10 цифр и знаки пунктуации. Аналогичным образом, программы на С и C++ записываются при помощи набора символов, состоящего из 26 строчных букв алфавита:

abcdefghijklmnopqrstuvwxyz

26 заглавных букв:

ABCDEFGHIJKLMNOPQRSTUVWXYZ

10 цифр:

0123456789

и следующих символов -

+ - \* / = , . \_ : ; ? \ " ' ~ I ! # % $ & () [] { } @

Также в С и C++ используется разделительный символ (пробел, табуляция или перевод строки). Комбинации символов без разделительных символов между ними также считаются допустимыми символами С и C++. Далее следует набор допустимых символов С и C++:

++ — == && || << >> >= <= += -= \*= /= ?: : : /\* \*/ //

## Данные: переменные и константы.

Компьютер, выполняя программу, может заниматься разнообразной деятельностью. Он может складывать числа, сортировать имена, заниматься распознаванием речи и изображения на экране видеодисплея, вычислять орбиты комет, подготавливать список почтовых адресов абонентов, чертить фигуры, делать логические выводы или что-нибудь еще, что только вы можете себе представить. Чтобы заниматься всем этим, программам необходимо работать с «данными» — числами и символами, т. е. объектами, которые несут в себе информацию, предназначенную для использования. Некоторые данные устанавливаются равными определенным значениям еще до того, как программа начнет выполняться, а после ее запуска сохраняют такие значения неизменными на всем протяжении работы программы. Это «константы». Другие данные могут изменяться, или же им могут быть присвоены значения во время выполнения программы; они называются «переменными». Мы уже использовали данные термины, но вы знакомитесь с ним только здесь. Вспомните формулу площади круга при рассмотрении различных алгоритмов.

Различие между переменной и константой довольно очевидно: выполнения программы значение переменной может быть изменено (например, с помощью присваивания), а значение константы — нет. Указанное различие приводит к тому, что обработка переменных компьютером оказывается немного сложнее и требует больше времени, чем обработка констант, но, несмотря на это, он вполне справляется с такой деятельностью.

Константы бывают литеральными и типизованными, причем литеральные константы делятся на: символьные, строковые, целые и вещественные.

Символьные константы представляются отдельным символом, заключенным в одинарные кавычки (апострофы): 'е', '@', '<'.

Строковые константы - это последовательность символов, заключенная в двойные кавычки: “Это пример не самой длинной строковой константы”.

Целые константы бывают следующих форматов:

* десятичные;
* восьмеричные;
* шестнадцатеричные.

Десятичные могут быть представлены как последовательность цифр, начинающаяся не с нуля, например: 123; 2384.

Восьмеричные константы - последовательность восьмеричных цифр (от 0 до 7), начинающаяся с нуля, например 034; 047.

Шестнадцатеричный формат констант начинается с символов 0Х с последующими шестнадцатеричными цифрами (0...9, А...F), например: 0xF4; 0X5D. Буквенные символы при этом могут быть представлены в любом регистре.

Длинные целые константы, используемые в переменных типа long, определяются буквой l или L сразу после константы без пробела: 36L, 012L, 0x52L.

Вещественные константы числа с плавающей запятой могут быть записаны в десятичном формате (24.58; 13.0;.71) или в экспоненциальной форме (1е4; 5е+2; 2.2е-5, при этом может пропускаться целая или дробная часть:.2е4).

Типизованные константы используется как переменные, значение которых не может быть изменено после инициализации.

Символьные константы в C++ занимают в памяти 1 байт и, следовательно, могут принимать значения от 0 до 255 (см табл. 1.2). При этом существует ряд символов, которые не отображаются при печати, - они выполняют специальные действия, возврат каретки, табуляция, и называются символами escape-последовательномти. Термин «escape-последовательность» ввела компания Epson ставшая первой фирмой, которая для управления выводом информации на своих принтерах стала использовать неотображаемые символы. Исторически сложилось так, что управляющие последовательности начинались с кода с десятичным значением 27 (0х1В), что соответствовало символу "Escape" кодировки ASCII.

Escape-символы в программе изображаются в виде обратного слеша, за которым следует буква или символ.

## Данные: типы данных.

Все программы используют какую-то информацию, которая обычно представляется при помощи одного из семи основных типов данных С и C++: текстовый или char, целочисленный или int, с плавающей точкой или float, с плавающей точкой двойной длины или double (long double), перечисляемый или enum, пустой или void и указатели или pointer. Далее следует описание типов:

* Тип char — отдельные символы, например a, Z, ?, 3 и строки, к примеру "Важнее жить, а не ускорять темп жизни" ("There is more to life than increasing its speed"). (Обычно символ имеет длину 8 разрядов или 1 байт и принимает значения в диапазоне от 0 до 255.)
* Тип int — числа, которые используются для счета (1, 2, 7, -45 и 1345). (Обычно имеют длину 16 разрядов, 2 байта или 1 слово и значения в диапазоне от -32,768 до 32,767.) (В Windows 95 и Windows NT целые имеют длину 32 разряда и диапазон значений от - 2,147,483,648 до 2,147,483,647.) (Для того чтобы не было разночтения в тексте и программных примерах сохраняется американский тип нотации при записи десятичных и дробных чисел: в целых числах позиции отделяются запятой, а в дробных — целая часть отделяется от дробной точкой. — Прим. перев.)
* Тип float — числа, имеющие дробную часть, например pi (3.14159), они также могут быть записаны и в экспоненциальном формате (7.563Е21). Их еще называют действительными числами. (Обычно имеют длину 32 разряда, 4 байта или 2 слова и значения в диапазоне от +/-3.4Е-38 до 3.4Е+38.)
* Тип double — числа с плавающей точкой двойной длины имеют расширенный диапазон значений. (Обычно их длина 64 разряда, 8 байтов или 4 слова, а диапазон — от 1.7Е-308 до 1.7Е+ЗО8.) Длинные числа двойной точности (long double) имеют еще большую точность. (Обычно их длина 80 байт или 5 слов и диапазон — от +/-1.18Е-4932 до 1.18Е+4932.)
* Тип enum. Перечисляемый тип данных допускает определение его пользователем.
* Тип void используется для обозначения величин, имеющих нулевую длину и не имеющих значения. Этот тип также может использоваться для создания родовых (обобщенных) указателей.
* Тип pointer не содержит информации в общепринятом смысле — как для других типов; вместо этого, в каждом указателе находится адрес ячейки памяти, где хранятся реальные данные.

|  |  |  |
| --- | --- | --- |
| **Тип** | **Размер, байт** | **Значение** |
| bool | 1 | true или false |
| unsigned short int | 2 | от 0 до 65 535 |
| short int | 2 | от -32 768 до 32 767 |
| unsigned long int | 4 | от 0 до 4 294 967 295 |
| long int | 4 | от-2 147 483 648 до 2 147 483 647 |
| int (16 разрядов) | 2 | от -32 768 до 32 767 |
| int (32 разряда) | 4 | от-2 147 483 648 до 2 147 483 647 |
| unsigned int (16 разрядов) | 2 | от 0 до 65 535 |
| unsigned int (32 разряда) | 4 | от 0 до 4 294 967 295 |
| char | 1 | от 0 до 256 |
| float | 4 | от 1.2е-38 до 3.4е38 |
| double | 8 | от 2.2е-308 до 1.8е308 |
| void | 2 или 4 | - |

Помимо различия между переменными и константами существует еще различие между типами данных. Некоторые данные в программе являются числами, некоторые — буквами, или, более обобщенно, символами. Компьютер должен иметь возможность идентифицировать и обрабатывать требуемым образом данные любого из этих типов. В языке Си предусмотрено использование нескольких основных типов данных. Если величина есть константа, то компилятор обычно может распознать ее тип только по тому виду, в каком она присутствует в программе. Однако в случае переменной необходимо, чтобы ее тип был объявлен в операторе описания. Дополнительные детали, относящиеся к типам данных, мы будем сообщать вам по мере изложения. Рассмотрим основные типы данных, имеющиеся в языке Си. В стандарте языка Си используется семь ключевых слов, указывающих на различные типы данных. Приведем список этих ключевых слов:

int

long

short

unsigned

char

float

double

Первые четыре ключевых слова используются для представления целых, т. е. целых чисел без десятичной дробной части. Они могут появляться в программе по отдельности или в некоторых сочетаниях, как, например, unsigned short. Следующее слово char предназначено для указания на буквы и некоторые другие символы, такие, как #, $, % и &. Последние два ключевых слова используются для представления чисел с десятичной точкой. Типы, обозначаемые этими ключевыми словами, можно разделить на два класса по принципу размещения в памяти машины. Первые пять ключевых слов определяют «целые» типы данных, в то время как последние два — типы данных с «плавающей точкой».

В этом месте у некоторых из вас могут появиться недоуменные вопросы: «Целые типы данных? Типы данных с плавающей точкой?» Не пугайтесь. Если эти термины кажутся вам непривычными или непонятными, мы дадим краткое объяснение их смысла.

Биты, байты и слова. Термины «бит», «байт» и «слово» обычно используются для описания как элементов данных, которые обрабатывает компьютер, так и элементов памяти. Здесь мы займемся рассмотрением второго смысла этих терминов.

Наименьшая единица памяти называется бит. Она может принимать одно из двух значений: 0 или 1. (Иначе говоря, бит может находиться в состояниях «включен» или «выключен»; эта фраза совершенно аналогична первому высказыванию.) В один бит нельзя поместить достаточное количество информации, но в машине содержится большое число битов; дело в том, что бит — основной «строительный блок», из которых создается память компьютера.

Байт — более удобный элемент памяти. В большинстве машин байт состоит из 8 бит. Поскольку каждый бит можно установить либо в состояние 0, либо в состояние 1, всего в байтовом формате можно представить 256 (два в восьмой степени) различных комбинаций из нулей и единиц. Такие комбинации можно использовать, например, для представления целых чисел в диапазоне от 0 до 255 или для кодирования набора символов. Это можно получить при помощи «двоичного кода», в котором для представления чисел используются только нули и единицы.

При современном подходе к проектированию компьютеров слово - самый естественный элемент памяти. В 8-разрядных микрокомпьютерах, таких, как ЭВМ фирмы Sinklair или первые модели машин фирмы Apple, слово занимает как раз 1 байт. Многие более новые персональные вычислительные системы, такие, как IBM PC и Lisa фирмы Apple, являются 16-разрядными. Это означает, что размер слова у них 16 бит, т. е. 2 байта. Большие компьютеры могут иметь 32-, 64-разрядные слова или даже более длинные. Совершенно очевидно, что чем длиннее слово, тем больше информации можно туда поместить. Обычно в компьютерах предусмотрена возможность объединять вместе два или более слов для того, чтобы помещать в память элементы данных большей длины, но этот процесс сильно замедляет работу компьютера.

В наших примерах мы предполагаем, что длина слова равна 16 бит, если мы не оговорили противного.

Для человека различие между целым числом и числом с плавающей точкой выражается в способе записи. Для компьютера различие выражается в способе занесения этих чисел в память. Давайте рассмотрим по очереди каждый из двух классов чисел.

**Замечание**. Переменная типа void не имеет значения и служит для согласования синтаксиса.

### Три целочисленных типа.

Давайте теперь рассмотрим некоторые специфические особенности основных типов данных, используемых в языке Си. Для каждого типа мы покажем, как описать переменную, как представить константу и как лучше всего использовать данные этого типа. В некоторых компиляторах с языка Си не реализована обработка всех типов данных; поэтому вам необходимо свериться с руководством по языку Си, имеющимся в комплекте вашей машины, чтобы посмотреть, какие из типов доступны для использования.

В языке Си имеется несколько целых типов, поэтому у вас есть возможность вносить изменения в свою программу, чтобы она удовлетворяла требованиям конкретной машины или определенного задания. Если вы не хотите заботиться о таких деталях, то, вообще говоря, вы можете просто остановиться на типе int и не думать больше о других возможностях.

Все данные типов int, short и long являются «числами со знаком», т. е. допустимыми значениями переменных этих типов могут быть только целые числа — положительные, отрицательные и нуль. Один бит используется для указания знака числа, поэтому максимальное число со знаком, которое можно представить в слове, меньше, чем максимальное число без знака. Например, в формате 16-битного слова можно представить любые целые числа без знака из диапазона от 0 до 65535. Точно так же 16-битное слово можно использовать для представления целых чисел со знаком из диапазона от — 32768 до + 32767. Заметим, что длины диапазонов в обоих случаях одинаковые.

Язык Си предоставляет пользователям возможность выбора размера элемента памяти (одного из трех) для представления целых чисел. Типу int обычно соответствует стандартная длина слова, принятая на используемой машине. При этом гарантируется, что размер элементов памяти, отводимых под данные типа short и long, будет соответственно не больше и не меньше длины элемента памяти, выделяемого типу int. В некоторых вычислительных системах один или оба этих типа реализованы точно так же, как int. Все зависит от того, какое представление лучше соответствует архитектуре конкретной ЭВМ.

В Microsoft Visual C++ имеется три типа целых чисел. Кроме стандартного типа int, компилятор поддерживает также short int и long int. Последние чаще всего сокращаются до short и long. Хотя язык С не зависит от аппаратных средств (если говорить о синтаксисе и подобных вещах), типы данных, используемые в С, являются машинно-зависимыми. Следовательно, на практике длина short, int и long зависит от реализации. Единственное, что гарантируется во всех компиляторах С, — то, что переменные типа short не могут быть больше (по значению) переменных типа long. В Microsoft Visual C/C++ и для short, и для fat отводится 2 байта. (В Windows 95 и Windows NT целые имеют длину 32 разряда.) Для хранения переменной типа long отводится 4 байта.

У целого числа никогда не бывает дробной части и, согласно правилам языка Си, десятичная точка в его записи всегда отсутствует. В качестве примера можно привести числа 2, —23 и 2456. Числа вида 3.14 и 2/3 не являются целыми. Представив целое число в двоичном виде, его нетрудно разместить в памяти машины.

Например, число 7 в двоичном виде выглядит как 111. Поэтому, чтобы поместить это число в 1-байт слово, необходимо первые 5 бит установить в 0, а последние 3 бит — в 1.

04_01

Описание данных целого типа.

При описании данных необходимо ввести только тип, за которым должен следовать список имен переменных. Ниже приведены некоторые возможные примеры описаний:

int erns;

short stops;

long Johns;

int hogs, cows, goats;

В качестве разделителя между именами переменных необходимо использовать запятую; весь список должен оканчиваться символом «точка с запятой». Вы можете собрать в один оператор описания переменных с одним и тем же типом или, наоборот, разбить одно описание на несколько операторов. Например, описание

int erns, hogs, cows, goats;

будет давать тот же самый эффект, что и два отдельных описания типа int в предшествующем примере. При желании вы даже могли бы использовать четыре различных описания данных типа int — по одному для каждой переменной. Иногда вам могут встретиться сочетания ключевых слов, как, например, long int или short int. Эти комбинации являются просто более длинной записью ключевых слов long и short.

Целые константы.

Согласно правилам языка Си, число без десятичной точки и без показателя степени рассматривается как целое. Поэтому 22 и -273 — целые константы. Но число 22.0 нецелое, потому что в его записи имеется десятичная точка, и число 22ЕЗ тоже нецелое, поскольку в записи использован порядок. Кроме того, указывая целое число, нельзя использовать запятые. Нужно записать 23456 вместо 23,456.

Если вы хотите ввести некоторую константу типа long, то можете это сделать, указав признак L или l в конце числа. Использование прописной буквы L более предпочтительно, поскольку ее труднее спутать с цифрой 1. Примером такой константы служит число 212L. Очевидно, что само по себе число 212 не очень большое, но добавление признака L гарантирует, что в памяти для него будет отведено необходимое число байтов. Это может оказаться важным для достижения совместимости, если данное число должно использоваться вместе с другими переменными и константами типа long.

Вполне возможно, что вам уже ничего больше не нужно знать про то, как записывают константы, но в языке Си имеются еще и два других способа.

Первый: если целое начинается с цифры 0, оно интерпретируется как «восьмеричное» число. Восьмеричные числа — это числа, представляемые «по основанию восемь» (т. е. их запись состоит из комбинаций степеней числа восемь). Например, 020 — это удвоенная первая степень основания восемь, т. е. восьмеричный эквивалент числа 16. При отсутствии в первой позиции нуля это просто обыкновенное (десятичное) число 20.

Второй: целое, начинающееся с символом 0х или 0Х интерпретируется как шестнадцатеричное число, т. е. число, записываемое по основанию 16. Поэтому запись 0x20 представляет собой удвоенную первую степень числа 16, или 32.

Восьмеричные и шестнадцатеричные числа чрезвычайно популярны среди программистов. Поскольку 8 и 16 являются степенями числа 2, а 10 — нет, использование этих систем счисления при работе на машине является вполне естественным. Например, число 65536, которое часто возникает при программировании на 16-Разрядных компьютерах, в шестнадцатеричной записи имеет вид 10000.

Инициализация переменных целого типа.

Константы часто применяются при «инициализации» переменных. Это означает присваивание переменной некоторого значения перед началом обработки. Ниже приводятся примеры использования инициализации:

erns = 1024;

stops = -3;

Johns = 12345678;

Если захотите, вы можете инициализировать переменную в операторе описания. Например:

int hogs = 23;

int cows = 32, goats =14;

short dogs, cats = 92;

Заметим, что в последней строке была инициализирована только переменная cats. При невнимательном чтении может создаться впечатление, что переменная dogs тоже инициализирована значением 92, поэтому лучше избегать смешивания инициализируемых и неинициализируемых переменных в одном операторе описания.

Рекомендации. Какие переменные целого типа со знаком лучше всего использовать? Одной из целей введения в язык трех классов целых чисел, имеющих различные размеры, было предоставить возможность согласования типа переменной с требованиями задачи. Например, если переменная типа int занимает одно слово, а переменная типа long — два, то тип long позволяет обрабатывать большие числа. Если в вашей задаче такие большие числа не используются, то незачем и вводить в программу переменную типа long, так как, если вместо числа, занимающего одно слово памяти, используется число, занимающее два слова, работа машины замедляется. Вообще говоря, необходимость введения данных типа long целиком зависит от вашей вычислительной системы, поскольку под данные типа int на одной машине может отводиться больше памяти, чем под данные типа long на другой. В конце мы еще раз хотим напомнить вам, что обычно вполне достаточно использовать переменную типа int.

Переполнение, возникающее при обработке целых чисел. Что происходит, когда в процессе обработки данных появляется значение, лежащее вне того диапазона чисел, которому соответствует данный целый тип? Давайте присвоим некоторой переменной целого типа наибольшее возможное значение, выполним операцию сложения и посмотрим, что произойдет.

/\* переполнение \*/

main()

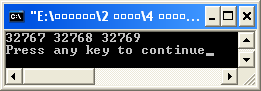
{

int i = 32767;

printf("%d %d %d\n", i, i+1, i+2);

}

Ниже приведен результат работы этой программы, выполненной на нашей вычислительной системе:



Целая переменная i ведет себя здесь как одометр в машине (прибор для определения пройденного расстояния). Когда его показания достигают максимума, данная величина «сбрасывается», и все начинается сначала. Основное отличие состоит в том, что показания одометра растут, начиная с нуля, а значения нашей переменной типа int — с величины — 32768.

Заметим, что при этом вам не сообщают, что переменная i превысила максимальное значение. Для регистрации подобных событий вы должны использовать свои программные средства.

Описанный подход не вытекает непосредственно из правил языка Си, а является довольно распространенным способом реализации.

Модификатор unsigned.

Все компиляторы С и C++ позволяют объявлять некоторые типы как беззнаковые. В настоящее время модификатор unsigned (без знака) можно использовать с четырьмя типами данных: char, short int и long int. Когда один из этих типов объявляется беззнаковым, диапазон представляемых им значений можно рассматривать как цифры на счетчике пройденного пути в автомобиле. Отсчет начинается со значения 000..., достигает максимума 999..., а затем сбрасывается в 000.... Отображаются только положительные целые числа. Подобным образом, данные беззнакового типа могут иметь только положительные значения в диапазоне от 0 до максимального для данной длины слова числа.

Предположим, к примеру, что разрабатывается новый тип данных my\_octal, и переменные типа my\_octal могут быть только 3-разрядными. Кроме этого, пусть числа типа my\_octal по умолчанию имеют знак. Поскольку переменная этого типа может иметь значения разрядов от 000 до 111 (или от 0 до 7 в десятичной системе), вы можете столкнуться с проблемой, если захотите представлять и положительные, и отрицательные значения. Положительные и отрицательные числа в диапазоне от 0 до 7 получить нельзя, так как один из трех разрядов необходим для представления знака числа. Поэтому диапазон типа my\_octal является подмножеством. Когда старший значащий разряд равен 0, число — положительное, когда 1 — отрицательное. Таким образом, диапазон переменных типа my\_octal от —4 до +3, что показано в таблице.

|  |  |
| --- | --- |
| **Уникальные комбинации**  **нулей и единиц** | **Десятичный**  **эквивалент** |
| 000 | +0 |
| 001 | +1 |
| 010 | +2 |
| 011 | +3 |
| 100 | -4 |
| 101 | -3 |
| 110 | -2 |
| 111 | -1 |

Если же, однако, для переменной типа my\_octal использовать модификатор unsigned, можно обеспечить диапазон от 0 до 7, поскольку для получения большего диапазона положительных чисел можно объединить старший разряд с двумя младшими, а не использовать его для указания знака числа (см. таблицу).

|  |  |
| --- | --- |
| **Уникальные комбинации**  **нулей и единиц** | **Десятичный**  **эквивалент** |
| 000 | +0 |
| 001 | +1 |
| 010 | +2 |
| 011 | +3 |
| 100 | +4 |
| 101 | +5 |
| 110 | +6 |
| 111 | +7 |

Эта простая аналогия справедлива для всех допустимых типов данных С, с которыми можно использовать unsigned. Длина и диапазон значений для основных типов данных С сведены в таблице.

В следующей таблице перечислены модификаторы типов данных, показанные во всех допустимых полных и сокращенных вариантах.

|  |  |  |
| --- | --- | --- |
| **Тип** | **Длина (байтов)** | **Диапазон значений** |
| char | 1 | от -120 до 127 |
| int | 2 | от -32768 до 32767 |
| short | 2 | от -32768 до 32767 |
| long | 4 | от –2 147 493 648 до2 147 483 647 |
| unsigned char | 1 | от 0 до 255 |
| unsigned int | 2 | от 0 до 65 535 |
| unsigned short | 2 | от 0 до 65 535 |
| unsigned long | 4 | от 0 до 4 294 967 295 |
| float | 4 | от 3.4Е-38 до 3.4Е+38 |
| double | 8 | от 1.7Е-308 до 1.7Е+308 |
| long double | 10 | от 1.1Е-4932 до 1.1Е+4982 |
| pointer | 2 | (near,based) |
| pointer | 4 | (far.huge) |

Обычно данный тип служит модификатором одного из трех ранее описанных типов. Поэтому мы можем использовать комбинации ключевых слов unsigned int или unsigned long как обозначения типов. Для указания типа unsigned int достаточно привести только ключевое слово unsigned. Некоторые вычислительные системы никак не обеспечивают аппаратную реализацию типа unsigned long; кроме того, существуют модели микропроцессоров, в которых unsigned - специальный тип фиксированного размера.

Целые беззнаковые константы записываются точно так же, как и обычные целые константы, с тем лишь исключением, что использование знака — запрещено.

Целые переменные без знака описываются и инициализируются совершенно аналогично тому, как это делается в случае обычных целых переменных. Ниже приведено несколько примеров:

unsigned int students;

unsigned players;

unsigned short ribs = 6;

Применение данного типа при введении в программу некоторой переменной гарантирует, что она никогда не станет отрицательной. Кроме того, если вы имеете дело только с положительными числами, вы сможете воспользоваться тем, что данные указанного типа могут принимать большие значения, чем данные эквивалентного типа со знаком. Обычно это применяется при адресации памяти и организации счетчиков.

### Числа с плавающей точкой.

В большинстве проектов разработки программного обеспечения оказывается вполне достаточным использовать данные целых типов. Однако в программах вычислительного характера часто применяются числа с плавающей точкой. В языке Си такие данные описываются типом float; они соответствуют типу real в Фортране и Паскале. Указанный подход, как вы могли заметить при внимательном чтении, позволяет представлять числа из гораздо более широкого диапазона, включая и десятичные дроби. Числа с плавающей точкой совершенно аналогичны числам в обычной алгебраической записи, используемой при работе с очень большими или малыми числами. Давайте рассмотрим ее подробнее.

Алгебраическая запись числа представляет собой произведение некоторого десятичного числа на степень, основание которой равно десяти. Ниже приведено несколько примеров.

|  |  |  |
| --- | --- | --- |
| Число | Алгебраическая запись | Запись для ввода в машину |
| 1 000 000 000 | 1,0\*109 | 1.0е9 |
| 123 000 | 1,23\*105 | 1.23е5 |
| 322.56 | 3,2256\*102 | 3.2256е2 |
| 0.000056 | 5,6\*10-5 | 5.6е-5 |

В первом столбце числа изображены в обычной записи, во втором приведена соответствующая алгебраическая запись, а в третьем столбце числа показаны в том виде, в котором они обычно представляются при вводе в машину и при выводе из нее — с символом е, за которым следует показатель степени по основанию десять (порядок).

Обычно для размещения в памяти числа с плавающей точкой отводится 32 бита — 8 бит для представления порядка и знака и 24 бита — для мантиссы (т. е. коэффициента при степени десяти). Важным фактом, который вам необходимо знать, является то, что такой способ дает возможность представлять числа с точностью до 6—7 десятичных цифр в диапазоне ±(1037 — 1038). Это может оказаться удобным, если вам понадобится обрабатывать числа того же порядка, что масса Солнца (2.0еЗ0 кг) или заряд протона (1.6е—19 Кл). (Многим нравится использовать подобные числа.)

Во многих ЭВМ предусматривается обработка данных типа double (вычислений с двойной точностью), когда для представления чисел используется удвоенное число битов, чаще всего 64. В некоторых машинах все 32 добавочных бита используются для хранения мантиссы. Это увеличивает число значащих цифр и уменьшает ошибку округления. В других машинах некоторое число битов из дополнительного набора используется для хранения большего порядка: это расширяет диапазон представления чисел.

Другой способ определения данных типа double заключается в использовании ключевых слов long float.

Числа с плавающей точкой более или менее соответствуют тому, что математики называют «вещественными числами». Они включают в себя числа, расположенные между целыми. Вот некоторые из них: 2.75, 3.16Е7, 7.00 и 2е-8. Очевидно, что любое число с плавающей точкой можно записать несколькими способами. Более полное обсуждение «Е-нотации» будет проведено дальше, а мы только кратко поясним, что запись вида «3.16Е7» означает число, полученное в результате умножения 3.16 на 10 в седьмой степени, т. е. на 1 с семью нулями. Число 7 называется «порядком» (показателем степени при основании 10).

Наиболее существенным моментом здесь является то, что способ кодирования, используемый для помещения в память числа с плавающей точкой, полностью отличается от аналогичной схемы для размещения целого числа. Формирование представления числа с плавающей точкой состоит в его разбиении на дробную часть и порядок; затем обе части раздельно помещаются в память. Поэтому число 7.00 из вышеприведенного списка нельзя поместить в память тем же способом, что и целое число 7, хотя оба имеют одно и то же значение. В десятичной записи (точно так же как и в двоичной) число «7.0» можно было бы записать в виде «0.7Е1»; тогда «0.7» будет дробной частью, а «1» — порядком. Для размещения чисел в памяти машины будут, конечно, использоваться двоичные числа и степени двойки вместо степеней десяти. Здесь мы остановимся лишь на различиях, связанных с практическим использованием чисел этих двух типов.

1. Целые числа не имеют дробной части, в то время как числа с плавающей точкой могут представлять как целые, так и дробные числа.
2. Числа с плавающей точкой дают возможность представлять величины из более широкого диапазона, чем целые.
3. При некоторых арифметических операциях, например при вычитании одного большого числа из другого, использование чисел с плавающей точкой приводит к большей потере точности.
4. Операции над числами с плавающей точкой выполняются, как правило, медленнее, чем операции над целыми числами. Однако сейчас уже появились микропроцессоры, специально ориентированные на обработку чисел с плавающей точкой, и в них эти операции выполняются довольно быстро.

_04_02

Ошибки округления чисел с плавающей точкой. Возьмите некоторое число. Добавьте к нему 1, а затем вычтите из полученной суммы исходное число. Что у вас получится? У нас получилась 1. Но вычисления, производимые над числами с плавающей точкой, могут дать и совершенно неожиданный результат:

/\*ошибка вычислений\*/

main()

{

float a, b;

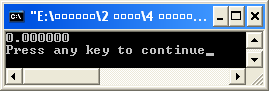
b = 2.0е20 +1.0;

а = b - 2.0е20;

printf(" %f \n", a);

}

Результат



Причина появления такого странного результата состоит в отсутствии достаточного числа разрядов для выполнения операций с требуемой точностью. Число 2.0е20 записывается как двойка с последующими двадцатью нулями, и, добавляя к нему 1, мы пытаемся изменить 21-ю цифру. Чтобы выполнить эту операцию корректно, программа должна иметь возможность поместить в память число, состоящее из 21 цифры. Но число типа float (т. е. с плавающей точкой) путем изменения порядка можно увеличить или уменьшить лишь на 6 или 7 цифр. Попытка вычисления оказалась неудачной. С другой стороны, если бы мы использовали, скажем, число 2.0е4 вместо 2.0е20, мы смогли бы получить правильный ответ, поскольку в этом случае мы пытались бы изменить 5-ю цифру, и точность представления чисел типа float оказалась бы вполне достаточной для этого.

### Описание переменных с плавающей точкой.

Переменные с плавающей точкой описываются и инициализируются точно таким же образом, что и переменные целого типа. Ниже приведено несколько примеров:

float noah, jonah;

double trouble;

float planck = 6.63e-34;

Константы с плавающей точкой

Правила языка Си допускают несколько способов записи констант с плавающей точкой. Наиболее общая форма записи константы — это последовательность десятичных цифр со знаком, включающая в себя десятичную точку, затем символ е или Е и показатель степени по основанию 10 со знаком. Вот два примера:

-1.56Е+12 2.87е-3

Знак + можно не писать. Разрешается также опускать либо десятичную точку, либо экспоненциальную часть, но не одновременно. Кроме того, можно не писать дробную или целую часть, но не обе сразу. Ниже приведено еще несколько правильно записанных констант с плавающей точкой:

3.14159 .2 4е16 .8Е-5 100.

Использовать пробелы при записи констант запрещается

1.56Е+ 12 — НЕПРАВИЛЬНО

В процессе обработки константы с плавающей точкой рассматриваются в формате с удвоенной точностью. Предположим, например, что переменная some типа float получает свое значение в результате выполнения оператора

some = 4.0\*2.0;

В этом случае константы 4.0 и 2.0 размещаются в памяти как данные типа double, т. е. для каждой из них (обычно) отводится 64 бит. Их произведение (равное 8) вычисляется с помощью операции умножения, выполняемой с двойной точностью, и только после этого производится усечение результата до нормального размера, соответствующего типу float. Все это обеспечивает максимальную точность ваших вычислений.

Переполнение и потеря значимости при обработке чисел с плавающей точкой. Что произойдет, если значение переменной типа float выйдет за установленные границы? Например, предположим, что вы умножаете 10е38 на 100 (переполнение) или делите 10е — 37 на 1000 (потеря значимости). Результат целиком зависит от реакции вашей вычислительной системы. В нашей системе при возникновении состояния «переполнение» результат операции заменяется максимально допустимым числом, а при потере значимости — нулем. В других системах в подобной ситуации могут выдаваться предупреждающие сообщения, выполнение задачи можно приостановить, или вам будет предоставлена возможность предпринять что-нибудь самому. Если этот вопрос окажется для вас существенным, вам необходимо будет свериться с правилами, действующими для вашей ЭВМ. В случае если вы не сможете найти никакой информации, не бойтесь пробовать другие возможности.

### Перечисляемый тип данных (enum).

При объявлении переменной перечисляемого типа с ней связывается набор именованных целочисленных констант, называемых перечисляемыми константами. В каждый момент времени переменная может иметь значение одной из констант, и к ней можно обращаться по имени. К примеру, следующее описание создает перечисляемый тип air\_supply, перечисляемые константы EMPTY, USEABLE и FULL, и перечисляемую переменную instructor\_tank:

enum air\_supply {EMPTY,

USEABLE,

FULL=5 } instructor\_tank;

Все константы и переменные имеют тип int, каждая константа автоматически получает по умолчанию некоторое значение, если не указывается какое-либо другое. В предыдущем примере константа EMPTY имеет по умолчанию целое значение ноль, так как она — первая в списке и не переопределена особо. Значение константы USEABLE равно 1, поскольку она следует непосредственно за константой с нулевым значением. Константа FULL инициализирована отдельно и имеет значение 5; если бы вслед за ней в списке была еще одна константа, эта новая константа была бы равна 6.

После создания типа air\_supply вы можете в дальнейшем описать другую переменную, student\_tank, следующим образом:

enum air\_supply student\_tank;

После этого оператора можно выполнить следующие:

instructor\_tank = FULL;

student\_tank = EMPTY;

При этом переменная instructor\_tank получит значение 5, а переменная student\_tank — значение 0.

**ПРИМЕЧАНИЕ.**

При описании дополнительных перечисляемых переменных в C++ не обязательно повторять ключевое слово enum. В компиляторе C++ допустимы оба варианта записи.

Часто делают ошибку, считая air\_supply переменной. Это — "тип" данных, который можно использовать в дальнейшем для создания дополнительных перечисляемых переменных, аналогичных instructor\_tank или student\_tank.

Так как instructor\_tank — название перечисляемой переменной типа air\_supply, instructor\_tank может использоваться в левой части оператора присваивания и получать значение. Это и происходит при явном присваивании переменной значения перечисляемой константы FULL. EMPTY, USEABLE и FULL — имена констант; это не переменные, и их значения не могут меняться.

Можно выполнять проверки переменных и сравнивать их с константами. В следующей законченной программе на С используются приведенные выше описания:

/\*06ENUM.C\*/

//Программа на С, демонстрирующая использование перечисляемых переменных

#include <stdio.h>

main()

{

enum air\_supply { EMPTY,

USEABLE,

FULL=5 } instructor\_tank;

enum air\_supply student\_tank;

instructor\_tank = FULL;

student\_tank = EMPTY;

printf("The value of instructor\_tank is %d\n",instructor\_tank);

if (student\_tank < USEABLE) {

printf("Refill this tank.\n");

printf("Class is cancelled.\n");

exit(0);

}

if (instructor\_tank >= student\_tank)

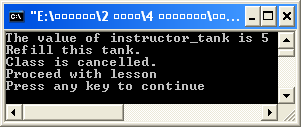
printf("Proceed with lesson\n");

else

printf("Class is cancelled'\n");

return(0);

}



В С тип enum эквивалентен типу int. Технически это означает возможность присваивать в программе целые значения непосредственно перечисляемым переменным. C++ осуществляет более строгую проверку типов и не позволяет выполнять такие смешанные операции.

### Модификаторы доступа.

Модификаторы const и volatile являются новыми в С и C++. Они были добавлены в стандарте ANSI С для того, чтобы различать переменные, которые никогда не изменяются (const), и переменные, значение которых может измениться непредсказуемо (volatile).

Модификатор const.

Иногда необходимо использовать некоторою величину, которая не изменяется на протяжении всей программы. Такая величина называется константой. Например, если в программе вычисляется площадь круга и длина окружности, часто будет использоваться константа pi=3.14159. При финансовых расчетах константой может быть процентная ставка. В подобных случаях можно улучшить читаемость программы, если дать константам осмысленные имена.

Использование значащих имен позволяет также избежать ошибок. Предположим, что некоторое постоянное значение (не переменная-константа) используется повсюду в программе. Опечатка может привести к появлению неправильных значений в одном или нескольких операторах. Однако если константа имеет собственное имя, опечатку может заметить компилятор, поскольку для неправильного идентификатора будет, скорее всего, отсутствовать объявление.

Предположим, вы пишете программу, в которой неоднократно используется значение pi. Можно было бы объявить переменную с именем pi и дать ей начальное значение 3.14159. Однако программа не должна иметь возможность изменения значения константы. К примеру, если вы по ошибке поставите "pi" слева от знака равенства, значение pi изменится, и все последующие вычисления будут неправильными. В С и C++ имеется механизм, позволяющий избежать подобных ошибок: вы можете задавать константы, значения которых неизменны.

В С и C++ для объявления константы перед ключевым словом (int, float или double) пишется const. К примеру:

const int iMIN=1, iSALE\_PERCENTAGE=25;

const float fbase\_change=32.157;

int irow\_index=1,itotal=100,lobject;

double ddistance=0,dvelocity;

Поскольку значение константы нельзя изменить, она должна инициализироваться при объявлении. Целые константы iMIN и iSALE\_PERSENTAGE объявлены со значениями 1 и 25, соответственно; константа fbase\_change имеет тип float и значение 32.157. Кроме этого, объявлены целые переменные (не константы) irow\_index, itotal и iobject. Для irow\_index и itotal заданы начальные значения 1 и 100, соответственно. И наконец, объявлены переменные (не константы) типа double: ddistance и dvelocity. Переменная ddistance получила нулевое начальное значение.

Константы и переменные используются в программе одинаково. Единственное отличие — в том, что начальные значения констант изменить нельзя. То есть, константы не являются выражениями ivalues: они не могут находиться слева от знака равенства. (Выражения, указывающие на ячейки памяти, называются ivalues. Выражения, указывающие на изменяемые величины, называются модифицируемыми ivalues. Примером модифицируемого ivalues может быть имя переменной, объявленной без модификатора const.)

Обычно в операциях присваивания значение операнда, стоящего справа, записывается в ячейку памяти, занимаемую операндом, стоящим слева. Поэтому в операции присваивания операнд, стоящий слева (или единственный операнд в унарных операциях присваивания), должен быть выражением, указывающим на модифицируемую ячейку памяти.

Определение констант через #define.

В С и C++ имеется еще один метод для задания констант: директива компилятора #define. Посмотрим на пример. Предположим, что в начале программы стоит оператор:

#define SALES\_TEAM 10

Оператор состоит из слова #define, за которым следует два набора символов, разделенных пробелами. При компиляции выполняется несколько проходов по программе. Первый из них осуществляет препроцессор компилятора, выполняющий, в частности, директивы #include и #define. Когда он встречает директиву #define, он заменяет каждое вхождение строки SALESTEAM исходного текста на число 10.

Другими словами, при наличии директивы #define препроцессор заменяет каждое вхождение первой последовательности символов "SALESTEAM" в программе на вторую последовательность — "10". Кроме того, нельзя изменить значение SALES\_TEAM, так как такой переменной не объявлено. При данном синтаксисе SALES\_TEAM имеет все признаки константы. Обратите внимание на то, что оператор #define не имеет в конце точки с запятой. Если за значением 10 будет стоять точка с запятой, все вхождения SALES\_TEAM будут замены на "10;". При выполнении директивы первая последовательность символов заменяется на вторую.

Все рассмотренные до сих пор программы достаточно короткие и могут храниться в одном файле. Если оператор, подобный #define SALES\_TEAM, появился в начале файла, замена "SALES\_TEAM" на "10" выполняется во всей программе. Следует учитывать, что директива компилятора действует только в том отдельном файле, в котором она записана. Выше описывались два метода описания констант: при помощи ключевого слова const и при помощи директивы компилятора #define. Во многих программах результат от использования любого метода, по сути, одинаков. С другой стороны, при использовании модификатора const создается переменная", значение которой нельзя изменить. Позднее вы узнаете, как объявлять переменные так, чтобы они существовали только в некоторых частях программы. То же самое можно сказать и о константах, объявленных с ключевым словом const. Поэтому объявления с модификатором const в чем-то гибче директивы #define. Следует добавить, что директива #define имеется в стандартном языке С и, следовательно, уже знакома программистам на С.

Оператор, позволяющий определять длину окружности, можно было бы записать в следующем виде:

circ = 3.14 \* diameter;

Приведенная здесь константа 3.14 — известное число пи. Чтобы ввести ту или иную константу в программу, нужно указать ее фактическое значение, как было сделано выше. Однако существуют веские причины использовать вместо этого «символические константы»; например, мы могли бы применять оператор:

circ = pi \* diameter;

а позже компилятор подставил бы в него фактическое значение константы.

В чем достоинства такого метода? Во-первых, имя говорит нам больше, чем число. Сравним два оператора

owed = 0.015 \* housevl;

owed = taxrate \* housevl;

Если мы изучаем большую программу, то второй вариант будет нам более понятен.

Во-вторых, предположим, что некоторая константа использовалась в нескольких местах программы и впоследствии возникла необходимость изменить ее значение — ведь в конце концов и налоговые тарифы (taxrate) меняются, и, к примеру, некое законодательное собрание приняло однажды закон впредь считать число пи равным 3 1/7. (Весьма вероятно, что окружности пришлось при этом скрываться от правосудия!) В таком случае требуется только изменить определение символической константы, а не отыскивать каждый случай ее появления в программе.

Теперь осталось выяснить, как можно создать такую символическую константу? Первый способ заключается в том, чтобы описать некоторую переменную и положить ее равной требуемой константе. Мы могли бы сделать это следующим образом:

float taxrate;

taxrate = 0.015;

Такой способ подходит для небольшой программы, в других же случаях он несколько неэкономичен, поскольку каждый раз при использовании переменной taxrate компьютер должен будет обращаться к той ячейке памяти, которая отведена данной переменной. Это служит примером подстановки «во время выполнения», так как она производится именно при выполнении программы. К счастью, в языке Си имеется и другой, лучший способ.

Этот способ реализуется с помощью препроцессора языка Си. Мы уже видели, как препроцессор использует директиву #include для включения информации из другого файла в программу. Кроме того, препроцессор дает нам возможность задавать константы. Для этого в начало файла, содержащего вашу программу, необходимо добавить только одну строку, аналогичную следующей:

#define TAXRATE 0.015.

При компиляции программы каждый раз, когда появится переменная TAXRATE, она будет заменяться величиной 0.015. Это называется подстановкой «во время компиляции». К тому моменту, когда вы начнете выполнение своей программы, все подстановки будут уже сделаны.

Несколько замечаний по поводу формата. Сначала идет ключевое слово #define. Оно должно начинаться с самой левой позиции. Потом следует символическое имя константы, а затем ее величина. Символ «точка с запятой» не используется, поскольку это не оператор языка Си. Почему имя TAXRATE пишется прописными буквами? В процессе использования языка Си выработалась традиция писать константы прописными буквами. Если при просмотре программы вам встретится имя, написанное прописными буквами, вы сразу поймете, что имеете дело с константой, а не с переменной. Это еще один способ улучшить читаемость программы. Ваша программа будет работать даже и тогда, когда вы будете писать константы строчными буквами, но при этом вы должны чувствовать свою вину, поскольку нарушаете традицию. Приведем простой пример:

/\* пицца \*/

# include <stdio.h>

#define PI 3.14159

main()/\* изучение вашей пиццы \*/

{

float area, circum, radius;

printf("Чемy равен радиус вашей пиццы?\n");

scanf("%f", &radius);

circum=2\*PI\*radius;

area = PI \* radius \* radius;

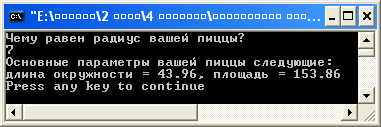
printf("Основные параметры вашей пиццы следующие:\n");

printf("длина окружности = %1.2f, площадь = %1.2f\n",circum, area);

}

04_05

Использование спецификации %1.2f в операторе printf() приведет к тому, что при печати результаты будут округлены до двух десятичных цифр. Мы понимаем, конечно, что написанная выше программа может и не отражать ваши собственные вкусы, касающиеся пиццы, но во множестве программ, посвященных этому вопросу, она займет свое скромное место. Вот один из примеров ее выполнения.



Директиву #define можно также использовать для определения символьных и строковых констант. Необходимо использовать знак «апостроф» в первом случае и кавычки — во втором. Примеры, приведенные ниже, вполне правомерны:

#define ВЕЕР '\007'

#define ESS 'S'

/\* X091.C \*/

#include "E:\LECTURE\AlgorithmProgramming\Tests\X09.h"

#include <stdio.h>

void main(void) /\* Определение константы во внешнем файле x09.h \*/

{

float r;

double cir;

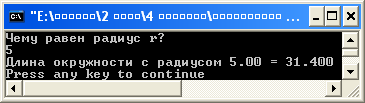
printf("Чему равен радиус r?\n ");

scanf("%f",&r);

cir = 2\*PI\*r;

printf("Длина окружности с радиусом %1.2f = %1.3f\n",r,cir);

}



/\* E:\LECTURE\AlgorithmProgramming\Tests\X09.h \*/

#define PI 3.14159

#define NULL '\0'

#define OOPS " Ну вот, вы и сделали это!"

А теперь мы хотим обрадовать лентяев. Предположим, вы разрабатываете целый пакет программ, использующих один и тот же набор констант. Вы можете произвести следующие действия:

1. Соберите все ваши директивы #define в один файл и назовите его, например, const.h.
2. В начало каждого файла, содержащего программу, включите директиву #include "const.h."

Тогда, если вы будете выполнять программу, препроцессор прочтет файл с именем const.h и использует все директивы #define вашей программы. Получилось так, что символ .h в конце имени файла напомнит вам, что этот файл является «заголовком», т. е. в нем содержится вся информация, которая должна попасть в начало вашей программы. Самому препроцессору безразлично, используете ли вы символ .h в имени файла или нет.

Язык Си — искусный фокусник: создание псевдоимен. Возможности директивы #define не исчерпываются только символическим представлением констант. Рассмотрим, например, следующую программу:

#include "alias.h"

program begin

whole yours, mine then

spitout(" Введите, пожалуйста, целое число.\n") then

takein(" %d" , &yours) then

mine=yours times TWO then

spitout(" %d в два раза больше вашего числа!\n" , mine) then

end

Странно, текст что-то смутно напоминает, язык немного похож на Паскаль, но программа не похожа на Си-программу. Секрет лежит, конечно, в файле с именем alias.h. Давайте посмотрим, что в нем содержится?

alias.h.

#define program main()

#define begin {

#define end }

#define then ;

#define takein scanf

#define spitout printf

#define TWO 2

#define times \*

#define whole int

Этот пример иллюстрирует, как работает препроцессор. Он просматривает вашу программу и проводит поиск элементов, определяемых директивами #define. Обнаружив такие элементы, он полностью заменяет их. В нашем примере во время компиляции все слова then заменяются символами «точка с запятой», end — } и т. д. Результирующая программа будет полностью идентична той, которую мы могли бы получить, если бы с самого начала писали ее в обычных терминах языка Си.

Эту мощную возможность языка можно использовать для задания макрокоманд, являющихся одним из вспомогательных средств программирования. Мы вернемся к обсуждению этой темы позднее.

Теперь необходимо упомянуть о некоторых ограничениях. Например, части программы, заключенные в кавычки, закрыты для подстановок. Операторы, приводимые ниже, служат иллюстрацией такого положения:

#define MN "минимифидианизм"

printf("OH глубоко верил в MN.\n");

Распечатка будет выглядеть так:

Он глубоко верил в MN.

Однако после выполнения оператора

printf(" Он глубоко верил в %s.\n" , MN);

мы получим следующий результат:

Он глубоко верил в минимифидианизм

В последнем случае константа с именем MN находилась вне кавычек и поэтому была заменена соответствующим значением.

Препроцессор языка Си является полезным вспомогательным средством, поэтому при написании программ старайтесь пользоваться преимуществами, которые он предоставляет. По мере изложения мы покажем вам дополнительные возможности применения препроцессора.

Модификатор volatile.

Ключевое слово volatile означает, что переменная может неожиданно измениться в результате каких-то внешних событий. Например, следующее Описание указывает, что переменная event\_time может менять свое значение независимо от программы:

volatile int event\_time;

Подобное описание необходимо, если, к примеру, переменная event\_time обновляется аппаратными средствами таймера реального времени. Программа, содержащая эту переменную, может прерываться аппаратурой таймера, что изменяет значение event\_time.

Объект данных должен объявляться как volatile, если это либо аппаратный регистр, отображенный в памяти, либо объект, общий для нескольких процессов, что типично для многозадачной операционной среды.

Cовместное использование const и volatile.

Модификаторы const и volatile можно использовать с любыми другими типами данных (например, с char или float), а также совместно. Следующее описание указывает, что значение переменной constant\_event\_time в программе изменяться не будет:

const volatile constant\_event\_time;

Однако модификатор volatile предупреждает компилятор о том, что тот не должен делать допущений относительно значения переменной в любой момент времени. Следовательно, следует отметить два момента. Во-первых, компилятор выдаст сообщение об ошибке в любой строке программы, в которой будет сделана попытка изменить значение переменной constant\_event\_time. Во-вторых, компилятор не удалит переменную constant\_event\_time из циклов, так как внешний процесс во время работы программы тоже может менять переменную.

### Модификаторы pascal, cdecl, near, far и huge.

Первые два модификатора, pascal и cdecl, чаще всего используется в сложных приложениях. Microsoft Visual C/C++ позволяет создавать программы, которые с легкостью могут вызывать процедуры, написанные на разных языках. Также возможна связь и в обратном направлении. Например, можно написать программу на Паскале, вызывающую процедуру C++. При подобном смешении языков необходимо учитывать два момента: имена идентификаторов и способ передачи параметров.

Когда Microsoft Visual C/C++ компилирует программу, все глобальные идентификаторы программы (функции и переменные) помещаются в выходной файл с объектным кодом для последующей компоновки. По умолчанию компилятор сохраняет эти идентификаторы, используя те же буквы, которые использовались в описаниях (заглавные, строчные или и те, и другие). Кроме этого, в начале идентификатора добавляется символ подчеркивания (\_). Так как встроенный компоновщик Microsoft Visual C/C++ различает (по умолчанию) заглавные и строчные буквы, предполагается, что все внешние идентификаторы, объявленные в программе, сохраняют символ подчеркивания и тот вид, который они имели при объявлении (название и регистр букв).

Модификатор pascal.

Язык Паскаль использует отличную от С и C++ последовательность вызова процедур. В Паскале (так же, как в ФОРТРАНе) аргументы функции передаются в порядке слева направо, а списки аргументов, имеющие переменную длину, не разрешены. Кроме этого, в Паскале аргументы из стека должна удалять вызванная, а не вызывающая функция (когда управление возвращается от вызванной функции).

Программы на С и C++ могут создавать такую вызывающую последовательность двумя способами. Во-первых, можно использовать ключ компиляции /Gc, при котором для всех содержащихся вызовов и описаний функций по умолчанию создается вызывающая последовательность паскалевского типа. Во-вторых, в программе на С можно явно переопределить вызывающую последовательность (по умолчанию, С-типа), если при описании функции указать ключевое слово pascal.

Как указывалось выше, при генерации вызова функции в С к имени функции по умолчанию добавляется символ подчеркивания и функция объявляется внешней. Также сохраняется написание ее названия. Однако, если используется ключевое слово pascal, символ подчеркивания не добавляется и идентификатор (функции или переменной) преобразуется в заглавные буквы.

Следующий фрагмент программы иллюстрирует использование функции с ключевым словом pascal. (Это же ключевое слово можно использовать, чтобы добиться совместимости с программами на ФОРТРАНе.)

float pascal pfcalculate (int iscore, int iweight)

{

}

Как видно из следующего примера, переменные также могут использоваться с ключевым словом pascal:

#define TABLESIZE 30

float pascal pfcalculate(int iscore, int iweight)

{

}

float pascal pfscore\_table[TABLESIZE];

main ()

{

int iscore 95, iweight = 10;

pfscore\_table[0] = pfcalculate(iscore,iweight);

return(0);

}

В этом примере переменная pfscore\_table описана как глобальная при помощи модификатора pascal. В функции main() также показано использование внешней ссылки на функцию типа pascal. Поскольку обе функции, main() и pfcalculate(), находятся в одном исходном файле, функция pfcalculate() — глобальная по отношению к main().

Модификатор cdecl.

Если при компиляции программ на С или C++ использовать ключ /Gz, все ссылки на функции и переменные генерируются в соответствии с типом вызовов языка Паскаль. Однако могут быть случаи, когда необходимо гарантировать, что некоторые используемые в программе идентификаторы сохраняют свое написание (различие заглавных и строчных букв) и имеют в начале символ подчеркивания. Чаще всего, это те идентификаторы, которые используются в другом файле на С.

Для обеспечения такой совместимости с языком С (сохранение различий букв при написании и начального символа подчеркивания) можно использовать ключевое слово cdecl. Когда cdecl используется при описании функции, это влияет также на способ передачи параметров.

Обратите внимание на то, что все функции С и C++, имеющие прототипы в заголовочных файлах Microsoft Visual C/C++, — например, stdio.h — имеют тип cdecl. Это гарантирует возможность компоновки с библиотечными функциями, даже если при компиляции использовалась опция /Gz. Следующий пример компилировался с ключом /Gz; в нем показано, как переписать предыдущий пример для обеспечения совместимости с С:

#define TABLESIZE 30

float cdecl cfcalculate(int iscore,int iweight)

{

}

float cdecl cfscore\_table[TABLESIZE];

main ()

{

int iscore 95, iweight = 10;

cfscore\_table[0] = cfcalculate(iscore,iweight);

return(0);

}

Модификаторы near, far и huge.

Три модификатора near, far и huge влияют на работу операции косвенной ссылки (\*); другими словами, они модифицируют размер указателей на объекты данных. Указатель near имеет длину 2 байта, указатель far — 4 байта, указатель huge — также 4 байта. Отличие указателей far и huge заключается в том, что указатель huge работает с формой адреса. Эта концепция детально описана в дальнейшем.

### Тип данных char

Этот тип определяет целые числа без знака в диапазоне от 0 до 255. Обычно такое целое размещается в одном байте памяти. В машине используется некоторый код для перевода чисел в символы и обратно. В большинстве компьютеров это код ASCII.

Описание символьных переменных. Для описания символьной переменной применяется ключевое слово char. Правила, касающиеся описания более чем одной переменной и инициализации переменных, остаются теми же, что и для других основных типов. Поэтому строки, приведенные ниже, являются допустимыми операторами.

char response;

char intable, latan;

char isma = 'S';

Символьные константы. В языке Си символы заключаются в апострофы. Поэтому, когда мы присваиваем какое-то значение переменной broiled типа char, мы должны писать

broiled = 'T'; /\*ПРАВИЛЬНО\*/

а не

broiled = Т; /\*НЕПРАВИЛЬНО\*/

Если апострофы опущены, компилятор «считает», что мы используем переменную с именем Т, которую забыли описать.

В стандарте языка Си принято правило, согласно которому значениями переменной или константы типа char могут быть только одиночные символы. В соответствии с этим последовательность операторов, указанная ниже, является недопустимой, поскольку там делается попытка присвоить переменной bovine значение, состоящее из двух символов:

char bovine; bovine = 'ox'; /\*НЕПРАВИЛЬНО\*/

Если вы посмотрите на таблицу кода ASCII, то увидите, что некоторые из «символов» в ней не выводятся на печать. Например, при использовании в программе символа номер 7 терминал компьютера издает звуковой сигнал. Но как использовать символ, который невозможно набрать на клавиатуре? В языке Си для этого имеются два способа.

*В первом способе* используется сам код ASCII. Вы должны только указать номер символа вместе с предшествующим знаком «обратная косая черта». Мы уже делали это в нашей программе «золотой эквивалент». Вот эта строка

beep = '\007';

Здесь имеются два важных момента, которые вы должны отчетливо представлять себе. Первый — это то, что последовательность знаков заключается в апострофы точно так же, как это делается с обычным символом. Второе — то, что номер символа должен быть записан в восьмеричном виде. При записи последовательности знаков мы можем случайно пропустить нули в первых позициях; в этом случае для представления кода «сигнал» мы могли бы использовать '\07' или даже '\7'. Но ни в коем случае не опускайте в записи последние нули! Последовательность символов '\020' можно записать в виде '\20', но не '\02'.

При использовании кода ASCII необходимо отметить различие между числами и символами, обозначающими числа. Например, символу "4" соответствует код ASCII, равный 52. Это символ " 4" , а не число 4.

*Во втором способе* представления «неудобных» знаков используются специальные последовательности символов. Они называются управляющими последовательностями и выглядят следующим образом:

\n новая строка

\t табуляция

\b шаг назад

\r возврат каретки

\f подача бланка

\\ обратная косая черта (\)

\’ апостроф (')

\” кавычки (")

При присваивании символьной переменной эти последовательности тоже должны быть заключены в апострофы. Например, мы могли бы написать оператор

nerf = ' \n';

а затем вывести на печать переменную nerf; это приведет к продвижению на одну строку вперед на печатающем устройстве или на экране дисплея.

Первые пять управляющих последовательностей являются общепринятыми символами, предназначенными для управления работой печатающего устройства: символ «новая строка» вызывает переход к новой строке; символ «табуляция» сдвигает курсор или печатающую головку на некоторое фиксированное число позиций 5 или 8; символ «шаг назад» производит сдвиг назад на одну позицию; символ «возврат каретки» осуществляет возврат к началу строки; символ «подача бланка» вызывает протяжку бумаги на одну страницу. В последних трех управляющих последовательностях символы \, ', " можно считать символьными константами [поскольку они служат для определения символьных констант и непосредственно используются в операторе printf(), применение их самих в качестве символов могло бы привести к ошибке]. Если вы хотите вывести на печать строку:

Запомните, “ символ \ называется обратная косая черта”

оператор будет выглядеть так:

printf(" Запомните, \" символ \\ называется обратная косая черта. \" \n");

Здесь у вас могут возникнуть два вопроса. Во-первых, почему мы не заключили управляющие последовательности в апострофы? Во-вторых, в каких случаях необходимо использовать код ASCII и когда управляющие последовательности, которые мы только что обсуждали? (Мы надеемся, что у вас возникли как раз эти вопросы, потому что мы собираемся отвечать именно на них.)

1. Когда символ является частью строки символов, заключенной в кавычки, он входит туда без апострофов независимо от того, является ли он управляющим или нет. Заметим, что все остальные символы в нашем примере (3,а, п, о, м, н и т. д.) тоже присутствуют в этой строке без кавычек. Строка символов, заключенная в кавычки, называется символьной строкой или цепочкой.
2. Если у вас есть возможность выбора одной из двух форм записи некоторой специальной управляющей последовательности, скажем ' \f', или эквивалентного кода из таблицы кодов ASCII — ' \016', то рекомендуем использовать ' \f'. Во-первых, это более наглядно. Во-вторых, лучше согласуется с требованием переносимости программ, поскольку даже в том случае, когда в системе не используется код ASCII, обозначение ' \f будет продолжать «работать».

04_03

*Программа.* Ниже приводится короткая программа, позволяющая узнавать номер кода символа даже в том случае, если на вашей машине не используется код ASCII.

main() /\* определяет номер кода символа \*/

{

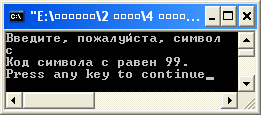
char ch;

printf("Введите, пожалуйста, символ \n");

scanf(" %c",&ch); /\* ввод пользователем символа \*/

printf("Koд символа %с равен %d.\n", ch, ch);

}



При работе с этой программой не забывайте нажимать клавишу [ввод] или [возврат] после ввода символа. Затем функция scanf() прочтет введенный символ; знак амперсанд (&) указывает, что символ должен быть присвоен переменной ch. Функция printf() выводит на печать величину ch дважды: первый раз как символ (в соответствии со спецификацией %с), а затем как десятичное целое число (в соответствии со спецификацией %d).

### Другие типы и размеры данных.

Этот раздел завершает рассмотрение основных типов данных. Некоторым читателям их число может показаться слишком большим. Остальные могут полагать, что описанных типов недостаточно; например, им захочется иметь булев тип или строковый, тип данных. В языке Си они отсутствуют, но, несмотря на это, он вполне подходит для написания программ, связанных с обработкой логических данных или строк.

В языке Си имеются и другие типы данных, построенные с использованием основных типов. Они включают в себя массивы, указатели, структуры и объединения. (Указатели используются функцией scanf(); признаком этого в данном случае служит префикс &.)

04_04

Размеры данных. Приведем таблицу размеров данных для некоторых распространенных вычислительных систем.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Размер слова** | **DEC PDP-11** | **DEC VAX** | **Interdata** | **IBM PC** |
|  | **16 бит** | **32 бита** | **32 бита** | **16 бит** |
| char | 8 | 8 | 8 | 8 |
| int | 16 | 32 | 32 | 16 |
| short | 16 | 16 | 16 | 16 |
| long | 32 | 32 | 32 | 32 |
| float | 32 | 32 | 32 | 32 |
| double | 64 | 64 | 64 | 64 |
| Диапазон порядка(double) | ±38 | ±38 | ±76 | -307 - 308 |

Как обстоит дело на вашей машине? Попробуйте выполнить нижеследующую программу:

/\* X05.C \*/

#include <stdio.h>

void main(void) /\* определение размера для данных \*/

{

int count;

printf("Данные типа int занимают %d байта.\n",sizeof(int));

printf("Данные типа char занимают %d байта.\n",sizeof(char));

printf("Данные типа long занимают %d байта.\n",sizeof(long));

printf("Данные типа short занимают %d байта.\n",sizeof(short));

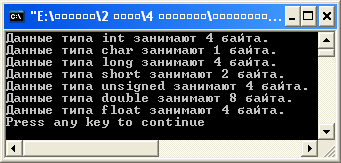
printf("Данные типа unsigned занимают %d байта.\n",sizeof(unsigned));

printf("Данные типа double занимают %d байта.\n",sizeof(double));

printf("Данные типа float занимают %d байта.\n",sizeof(float));

}

В языке Си имеется встроенная операция sizeof, которая позволяет определить размер объектов в байтах. Результат работы этой программы на нашей системе выглядит так:



Мы определили размеры данных только четырех типов, но вы легко можете модифицировать эту программу и найти размер объекта любого другого интересующего вас типа.

Использование типов данных. Во время разработки программы вам необходимо составить список требуемых переменных и указать при этом, какого они должны быть типа. Скорее всего вы будете использовать тип int или, возможно, float для определения чисел и тип char для символов. Описывайте эти данные в самом начале тела функции, в которой они используются. Имена переменных выбирайте таким образом, чтобы они указывали на их смысл. При инициализации переменной следите за тем, чтобы тип константы соответствовал типу переменной.

int apples = 3; /\*ПРАВИЛЬНО\*/

int oranges = 3.0; /\*НЕПРАВИЛЬНО\*/

Язык Си «рассматривает» такие несоответствия менее жестко, чем, скажем, Паскаль, но в любом случае лучше учиться избегать дурных привычек.

## Преобразование типов данных.

Во всех описанных до сих пор программах все переменные и числа, использованные в отдельных операторах, имели одинаковый тип: к примеру, int или float. Можно написать операторы, выполняющие действия с переменными различных типов. Такие операции называются смешанными. В отличие от некоторых других языков программирования, в С и C++ выполняется автоматическое преобразование типов.

В общем случае неявное преобразование типов сводится к участию в выражении переменных разного типа (так называемая арифметика смешанных типов). Если подобная операция осуществляется над переменными базовых типов, она может повлечь за собой ошибки, в случае, например, если результат занимает в памяти больше места, чем отведено под принимающую переменную, неизбежна потеря значащих разрядов.

Данные различных типов хранятся в памяти по-разному. Предположим, что запоминается число 10. Его представление зависит от типа числа. То есть, комбинация нулей и единиц в памяти будет одна, если 10 хранится как целое число, и другая — если оно хранится как число с плавающей точкой.

Предположим, что выполняется следующая операция, в которой переменные fresult и fvalue имеют тип float, а переменная ivalue — тип int:

fresult = fvalue \* ivalue;

Следовательно, это операция смешанного типа. При выполнении оператора значение ivalue будет преобразовано в число с плавающей точкой, а затем выполнится умножение. Компилятор распознает смешанные операции и генерирует код для выполнения следующих операций. Целочисленное значение переменной ivalue считывается из памяти. Затем оно преобразуется в формат с плавающей точкой, и полученное число умножается на действительное значение fvalue, а результат в формате числа с плавающей точкой запоминается в fresult. Другими словами, компилятор выполняет преобразование автоматически. Заметьте, что значение ivalue не изменилось при выполнении оператора, сохранился и его тип int.

Как было показано, в смешанных операциях над числами типа int и float. Для вычислений значение типа int преобразуется в значение типа float. Процесс преобразования не затрагивает хранимое целое значение. Теперь давайте рассмотрим смешанные операции с двумя разными типами переменных.

Сначала вы должны узнать, что на практике существует иерархия преобразований, что означает следующее: для выполнения вычислений объект более низкого приоритета временно преобразуется к типу с более высоким приоритетом. Иерархия преобразований имеет следующий вид, следуя от высших приоритетов к низшим:

double float long int short

К примеру, тип double имеет более высокий приоритет, чем тип int. Когда некоторый тип данных преобразуется в другой, более высокий, значение числа и его точность не изменяются.

Давайте посмотрим, что происходит при преобразовании типа float в тип int. Предположим, что описаны переменные ivalue1 и ivalue2 типа int и переменные fvalue и fresult типа float. Рассмотрим следующую последовательность операторов:

ivalue1 = 3;

ivalue2 = 4;

fvalue = 7.0;

fresult = fvalue + ivalue1/ivalue2;

Операция ivalue1/ivalue2 — не смешанная; она представляет собой деление двух целых чисел, и ее результат 0, так как при выполнении целочисленного деления дробная часть (в этом примере 0.75) отбрасывается. Поэтому значение fresult равно 7.0.

Что будет, если переменную ivalue2 описать как float? В этом случае переменная fresult получит значение с плавающей точкой 7.75, поскольку операция ivalue1/ivalue2 будет иметь смешанный тип. Тогда значение ivalue1 временно преобразуется в число с плавающей точкой 3.0 и результат деления будет равен 0.75. После сложения с fvalue получим 7.75.

Важно знать, что тип значения, стоящего в левой части оператора присваивания, определяет тип результата операции. Например: предположим, что переменные fx и fy объявлены как float, а переменная iresult объявлена int. Рассмотрим следующие операторы:

fx = 7.0;

fy = 2.0;

iresult = 4.0 + fx/fy;

Результат операции fx/fy равен 3.5; при сложении с 4.0 получается число с плавающей точкой 7.5. Однако это значение нельзя присвоить переменной iresult, так как она имеет тип int. Поэтому число 7.5 преобразуется в целое; при этом дробная часть отбрасывается. Полученное округленное число преобразуется из формата с плавающей точкой в целочисленный формат, и в результате переменная iresult получает целое значение 7.

### Явные преобразования типов при помощи операции приведения типа.

Как вы видели, компилятор С и C++ автоматически меняет формат переменной при выполнении смешанных операций, в которых задействуются различные типы данных. Однако, в некоторых случаях, несмотря на то, что автоматическое преобразование не выполняется, преобразование типов было бы желательным. В таких случаях необходимо особо указать, что требуется изменение типа. Кроме того, подобные явные указания облегчают другим программистам понимание используемых операторов. В языке С имеется несколько процедур, позволяющих указывать необходимость преобразования типа.

Одна из таких процедур называется приведением типа. Если нужно временно изменить формат некоторой переменной, достаточно перед ее именем указать в скобках имя типа, который необходимо получить. Например: если переменные ivalue1 и ivalue2 объявлены как int, а переменные fvalue и fresult как float, то три следующих оператора будут эквивалентны:

fresult = fvalue + (float)ivalue1/ivalue2;

fresult = fvalue + ivalue1/(float)ivalue2;

fresult = fvalue + (float)ivalue1/(float)ivalue2;

Все три оператора выполняют преобразование в тип с плавающей точкой и деление переменных ivalue1 и ivalue2. Если любая из этих переменных приведена в тип float, то — согласно описанным выше правилам стандартного преобразования типов в смешанных арифметических операциях — выполняется деление чисел с плавающей точкой. Третий оператор явно указывает, какие операции должны выполняться.

Резюме: основные типы данных.

*Ключевые слова.* Данные основных типов вводятся в программу при помощи следующих семи ключевых слов: int, long, short, unsigned, char, float, double.

*Целые числа со знаком.* Данные этих типов могут принимать положительные и отрицательные значения.

int: основной целый тип, используемый в вычислительной системе;

long или long int: может содержать целое значение, не меньшее максимальной величины, допускаемой типом int, или даже большее;

short или short int: максимальное целое число типа short не больше, чем максимальное целое число типа int, а может быть, и меньше.

Обычно числа типа long бывают больше чисел типа short, а тип int реализуется как один из двух указанных типов. Например, компилятор Lattice С на IBM PC под данные типов short и int отводит 16 бит, а под данные типа long — 32 бита. Все зависит от конкретной системы.

*Целые числа без знака.* Данные этих типов принимают только положительные значения или нуль. Это расширяет диапазон возможных положительных значений. При указании типа используйте ключевое слово unsigned: unsigned int, unsigned long, unsigned short. Просто unsigned соответствует написанию unsigned int.

*Символы.* Эти знаки соответствуют типографским символам, таким, как А, &, + и т.д. Обычно под каждый символ отводится 1 байт памяти.

char: ключевое слово, используемое для указания данных этого типа.

*Числа с плавающей точкой.* Данные этих типов могут принимать положительные и отрицательные значения. float: основной тип данных с плавающей точкой в системе; double или long float при размещении в памяти чисел с плавающей точкой такого типа отводится (возможно) элемент памяти большего размера; при этом может допускаться либо боль. шее число значащих цифр, либо большее значение порядка.

Резюме: как описывать простые переменные

1. Выбрать требуемый тип данных.
2. Выбрать имя для переменной.

Для оператора описания использовать нижеследующий формат: спецификация-типа имя-переменной; Спецификация-типа формируется из одного или более ключевых слов. Вот несколько примеров:

int erest;

unsigned short cash;

Вы можете описать в одном операторе несколько переменных одного типа, разделяя их имена запятыми:

char ch, unit, ans;

В операторе описания вы имеете возможность инициализировать переменную: float mass = 6.0E24;

**Что вы должны были узнать**. Здесь мы рассмотрели довольно большой материал. Суммируя его, мы обратим основное внимание на практическую сторону тех вопросов, которые здесь обсудили. Так же как и раньше, мы дадим краткие примеры. Ниже приводится сводка тех фактов, которые вы должны были узнать.

Что такое основные типы данных языка Си: int, short, long, unsigned, char, float, double.

Как описать переменную любого типа: int beancount, float root-beer; и т. д.

Как записать константу типа int: 256, 023, OXF5 и т. д.

Как записать константу типа char: 'r', 'U, '\007', '?' и т. д.

Как записать константу типа float: 14,92, 1,67е —27 и т. д.

Что такое слова байты и биты.

В каких случаях используются различные типы данных.

**Вопросы и ответы.**

Рассмотрение приводимых ниже вопросов должно помочь вам глубже усвоить материал.

**Вопросы**

1. Какими типами вы будете пользоваться при обработке данных -следующего вида:
   1. Население Рио-Фрито.
   2. Средний вес картины Рембрандта.
   3. Наиболее часто встречающаяся буква в тексте.
   4. Сколько раз указанная буква встречается в тексте.
2. Определите тип и смысл (если он есть) каждой из следующих констант:
   1. '\b'
   2. 1066
   3. 99.44
   4. ОХАА
   5. 2.0е30
3. Вирджила Анн Ксенопод (ВАКС) написала программу с множеством ошибок. Помогите ей обнаружить их.

#include <stdio.h>

main

{

float g; h;

float tax, rate;

g = e21;

tax = rate\*g;

}

**Ответы**

1. a. int, возможно short; население выражается целым числом

б. float; маловероятно, что среднее окажется целым числом

в. char

г. int, возможно unsigned

1. a. char, символ «шаг назад»

б. int, историческая дата2'

в. float, степень чистоты после мытья

г. Шестнадцатеричное число типа int; десятичное значение 170

д. float; масса Солнца в кг

1. Строка 1: правильная

Строка 2 должна содержать пару круглых скобок вслед за именем main, т. е.

main()

Строка 3: нужно использовать { , а не (

Строка 4: g и h должны разделяться запятой, а не точкой с запятой

Строка 5: правильная

Строка 6: (пустая) правильная

Строка 7: перед е должна стоять по крайней мере одна цифра

Например: 1е21 или 1.0e21

Строка 8: правильная

Строка 9: нужно использовать }, а не )

Недостающие строки: первая — переменной rate нигде не присваивается значение. Вторая — переменная h нигде не используется. Кроме того, программа никак не информирует нас о результатах вычислений. Ни одна из этих ошибок не будет служить препятствием для выполнения программы (хотя при компиляции может быть выдано предупреждение о неиспользуемой переменной), но все ошибки существенно снижают эффективность программы и без того уже ограниченную.