Лекция 2. Базовые типы данных, операции и операторы C++

План лекции:

- Переменные
- Базовые типы данных
- Статическая типизация и преобразования типов
- Переменные и константы
- Операции и операторы

ПЕРЕМЕННЫЕ

Как и во многих языках программирования, в C++ для хранения данных используются **переменные**. Переменная имеет тип, имя и значение. Тип определяет, какие значения может иметь переменная, какие операции с ней можно производить и сколько байт в памяти она будет занимать.

Перед использованием любую переменную надо определить. Синтаксис определения переменной выглядит следующим образом:

```
тип_переменной имя_переменной;
```

Простейшее определение переменной:

```
int age;
```

Здесь определена переменная age, которая имеет тип **int**. Поскольку определение переменной представляет собой инструкцию, то после него ставится точка с запятой.

Переменные

Имя переменной может представлять последовательность символов латинского алфавита, чисел и знака подчеркивания. При этом имя должно начинаться либо с алфавитного символа, либо со знака подчеркивания.

```
1 int _age33;
```

Также стоит учитывать, что C++ - регистрозависимый язык, а это значит, что регистр символов имеет большое значение. В следующем коде будут определяться две разные переменные:

```
1 int age;
2 int Age;
```

- Поэтому переменная Age не будет представлять то же самое, что и переменная age.
- Кроме того, в качестве имени переменной нельзя использовать ключевые слова языка C++

Инициализация переменных

После определения переменной можно присвоить некоторое значение:

```
1  int age;
2  age = 20;
```

Однако также можно сразу при определении переменной дать ей некоторое начальное значение. Данный прием называется **инициализацией**, то есть присвоение переменной начального значения:

```
1  #include <iostream>
2
3  int main()
4  {
5      int age = 28;
6      std::cout<<"Age = " << age;
7      return 0;
8  }
```

БАЗОВЫЕ ТИПЫ ДАННЫХ

- Правила языка Си++ требуют, чтобы в программе у всех переменных был задан тип данных.
- В языке различают понятия ***"тип данных"*** и ***модификатор типа***.
- Тип данных - это, например, целый, а модификатор - со знаком или без знака.

Базовые типы данных

Все типы данных в C++ за исключением void могут быть разделены на три группы:

- символьные (char, wchar_t, char16_t, char32_t)
- целочисленные (short, int, long, long long)
- типы чисел с плавающей точкой (float, double, long double).

Базовые типы данных

Переменная типа *char* имеет размер 1 байт. Переменной типа *char* кроме чисел могут присваиваться символьные константы. Символьная константа – это символ, заключенный в апострофы, например: '&', ‘a’, ‘4’, ‘K’ и т.д. Символ '0', например, имеет в кодировке ASCII значение 48.

Данные char занимают один байт и меняются в диапазоне:

- ***signed char*** (или просто char) -128..127
- ***unsigned char*** 0..255

Если переменная типа char должна содержать русские буквы, то её необходимо объявлять как unsigned char.

Базовые типы данных

- Ключевое слово ***float*** позволяет определить переменные вещественного типа
- Их значения имеют дробную часть, отделяемую точкой, например: -5.6, 31.28 и т.д.
- Вещественные числа могут быть записаны также в форме с плавающей точкой, например: 3.128e+1.
- Число перед символом "e" называется ***мантиссой***, а после "e" - ***порядком***. **Переменная типа *float* занимает в памяти 32 бита**. Она может принимать значения в диапазоне от 3.4e-38 до 3.4e+38.

Базовые типы данных

- Ключевое слово ***double*** позволяет определить вещественную переменную двойной точности
- Она занимает в памяти в два раза больше места, чем переменная типа float (т.е. ее размер **64 бита**).
- Переменная типа *double* может принимать значения в диапазоне от 1.7e-308 до 1.7e+308.

Базовые типы данных

- Ключевое слово ***void*** (не имеющий значения) используется для нейтрализации значения объекта, например, для объявления функции, не возвращающей никаких значений

Базовые типы данных

В стандарте ANSI языка C++ имеются следующие модификаторы типа:

- ***unsigned*** – без знака, принимает неотрицательные значения
- ***signed*** – со знаком, принимает положительные и отрицательные значения
- ***short*** - влияет на размер выделяемой памяти
- ***long*** - влияет на размер выделяемой памяти

Базовые типы данных

Модификаторы записываются перед спецификаторами типа, например: ***unsigned char***.

- Если после модификатора опущен спецификатор, то компилятор предполагает, что этим спецификатором является ***int***
- Модификатор типа ***signed*** указывает, что переменная может принимать как положительные, так и отрицательные значения. Как правило, при этом самый левый бит области памяти, выделяемой для хранения значения, используется для представления знака. Если этот бит установлен в 0, то значение переменной считается положительным. Если в 1, то значение переменной считается отрицательным

Базовые типы данных

В языке C++ определены следующие базовые типы данных:

- **bool**: логический тип. Может принимать одно из двух значений **true** (истина) и **false** (ложь). Размер занимаемой памяти для этого типа точно не определен, обычно 1 байт (8 бит)
- **char**: представляет один символ в кодировке ASCII. Занимает в памяти 1 байт. Может хранить любое значение из диапазона от -128 до 127, либо от 0 до 255
- **signed char**: представляет один символ. Занимает в памяти 1 байт (8 бит). Может хранить любое значение из диапазона от -128 до 127
- **unsigned char**: представляет один символ. Занимает в памяти 1 байт (8 бит). Может хранить любое значение из диапазона от 0 до 255
- **wchar_t**: представляет расширенный символ. На Windows занимает в памяти 2 байта (16 бит), на Linux - 4 байта (32 бита). Может хранить любой значение из диапазона от 0 до 65 535 (при 2 байтах), либо от 0 до 4 294 967 295 (для 4 байт)
- **char16_t**: представляет один символ в кодировке Unicode. Занимает в памяти 2 байта (16 бит). Может хранить любой значение из диапазона от 0 до 65 535
- **char32_t**: представляет один символ в кодировке Unicode. Занимает в памяти 4 байта (32 бита). Может хранить любой значение из диапазона от 0 до 4 294 967 295

Базовые типы данных

- **short**: представляет целое число в диапазоне от –32768 до 32767. Занимает в памяти 2 байта (16 бит). Данный тип также имеет синонимы **short int**, **signed short int**, **signed short**.
- **unsigned short**: представляет целое число в диапазоне от 0 до 65535. Занимает в памяти 2 байта (16 бит). Данный тип также имеет синоним **unsigned short int**.
- **int**: представляет целое число. В зависимости от архитектуры процессора может занимать 2 байта (16 бит) или 4 байта (32 бита). Диапазон предельных значений соответственно также может варьироваться от –32768 до 32767 (при 2 байтах) или от −2 147 483 648 до 2 147 483 647 (при 4 байтах). Но в любом случае размер должен быть больше или равен размеру типа short и меньше или равен размеру типа long. Данный тип имеет синонимы **signed int** и **signed**.
- **unsigned int**: представляет положительное целое число. В зависимости от архитектуры процессора может занимать 2 байта (16 бит) или 4 байта (32 бита), и из-за этого диапазон предельных значений может меняться: от 0 до 65535 (для 2 байт), либо от 0 до 4 294 967 295 (для 4 байт). В качестве синонима этого типа может использоваться **unsigned**
- **long**: представляет целое число в диапазоне от −2 147 483 648 до 2 147 483 647. Занимает в памяти 4 байта (32 бита).

  У данного типа также есть синонимы **long int**, **signed long int** и **signed long**

Базовые типы данных

- **unsigned long**: представляет целое число в диапазоне от 0 до 4 294 967 295. Занимает в памяти 4 байта (32 бита). Имеет синоним **unsigned long int**.
- **long long**: представляет целое число в диапазоне от −9 223 372 036 854 775 808 до +9 223 372 036 854 775 807. Занимает в памяти, как правило, 8 байт (64 бита).

  Имеет синонимы **long long int**, **signed long long int** и **signed long long**.
- **unsigned long long**: представляет целое число в диапазоне от 0 до 18 446 744 073 709 551 615. Занимает в памяти, как правило, 8 байт (64 бита).

  Имеет синоним **unsigned long long int**.
- **float**: представляет вещественное число ординарной точности с плавающей точкой в диапазоне +/- 3.4E-38 до 3.4E+38. В памяти занимает 4 байта (32 бита)
- **double**: представляет вещественное число двойной точности с плавающей точкой в диапазоне +/- 1.7E-308 до 1.7E+308. В памяти занимает 8 байт (64 бита)
- **long double**: представляет вещественное число двойной точности с плавающей точкой не менее 8 байт (64 бит). В зависимости от размера занимаемой памяти может отличаться диапазон допустимых значений.
- **void**: тип без значения

Числовые пределы

Два стандартных включаемых файла
<limits. h >
<float. h >
определяют числовые ограничения или минимальное и максимальное значения, которые могут храниться в переменной данного типа. Эти минимальные и максимальные значения гарантированно переносимы на любой C++ компилятор, использующий то же представление данных, что и ANSI C.

Числовые пределы

Например,

SHRT_MAX	Максимальное значение для переменной типа **short**.	32767
USHRT_MAX	Максимальное значение для переменной типа **unsigned short**.	65 535 (0xffff)
INT_MIN	Минимальное значение для переменной типа **int**.	-2147483648
INT_MAX	Максимальное значение для переменной типа **int**.	2147483647

Символьные типы

Определим несколько переменных:

```
1  char c ='d';
2  wchar_t d ='c';
```

Переменная типа char в качестве значения принимает один символ в одинарных кавычках: char c ='d'. Также можно присвоить число из указанного выше в списке диапазона: char c = 120. В этом случае значением переменной с будет тот символ, который имеет код 120 в таблице символов ASCII.

Целочисленные типы

```
1  short a = -10;
2  unsigned short b= 10;
3  int c = -30;
4  unsigned int d = 60;
5  long e = -170;
6  unsigned long f = 45;
7  long long g = 89;
```

Типы чисел с плавающей точкой

```
1  float a = -10.45;
2  double b = 0.00105;
3  long double c = 30.890045;
```

Размеры типов данных

В выше приведенном списке для каждого типа указан размер, который он занимает в памяти. Однако стоит отметить, что предельные размеры для типов разработчики компиляторов могут выбирать самостоятельно, исходя из аппаратных возможностей компьютера. Стандарт устанавливает лишь минимальные значения, которые должны быть. Например, для типов int и short минимальное значение - 16 бит, для типа long - 32 бита. При этом размер типа long должен быть не меньше размера типа int, размер типа int - не меньше размера типа short, а размер типа long double должен быть больше double.

Размеры типов данных

Однако бывают ситуации, когда необходимо точно знать размер определенного типа. И для этого в C++ есть оператор **sizeof()**, который возвращает размер памяти в байтах, которую занимает переменная:

```
#include <iostream>

int main()
{
    long double number = 2;
    std::cout << "sizeof(number) =" << sizeof(number);
    return 0;
}
```

Консольный вывод при компиляции в Windows для long double будет равен:
sizeof(number) = 8

Спецификатор auto

Иногда бывает трудно определить тип выражения. И согласно последним стандартам можно предоставить компилятору самому выводить тип объекта. Для этого применяется спецификатор **auto**. При этом, если мы определяем переменную со спецификатором auto, эта переменная должна быть обязательно инициализирована каким-либо значением:

```
1 auto number = 5;
```

На основании присвоенного значения компилятор выведет тип переменной. Неинициализированные переменные со спецификатором auto не допускаются.

СТАТИЧЕСКАЯ ТИПИЗАЦИЯ И ПРЕОБРАЗОВАНИЯ ТИПОВ

C++ является статически типизированным языком программирования. То есть если мы определили для переменной какой-то тип данных, то в последующем мы этот тип изменить не сможем. Соответственно переменная может получить значения только того типа, который она представляет. Однако нередко возникает необходимость присвоить переменной значения каких-то других типов. И в этом случае применяются преобразования типов.

Статическая типизация и преобразования типов

Ряд преобразований компилятор может производить неявно, то есть автоматически. Например:

```
#include <iostream>

int main()
{
    int code = 'g';
    char letter = 103;
    std::cout << letter << " in ASCII is " << code << "\n";
    return 0;
}
```

В данном случае числовой переменной типа int присваивается символ 'g'. Этот символ будет автоматически преобразовываться в число. По факту переменная получит числовой код этого символа в таблице ASCII.

Переменной letter, наоборот, присваивается число, хотя эта переменная представляет тип char. Это число будет преобразовываться в символ, и в итоге переменная letter будет хранить символ, чей код в таблице ASCII равен 103, то есть символ 'g'.

Результатом этой программы будет следующий консольный вывод: q in ASCII is 103

Таблица специальных управляющих последовательностей

Управляющая последовательность	Наименование
\a	Звонок
\b	Возврат на шаг
\t	Горизонтальная табуляция
\n	Перевод строки
\v	Вертикальная табуляция
\r	Возврат каретки
\f	Перевод страницы
\”	Кавычки
\’	Апостроф
\\	Обратный слеш

Статическая типизация и преобразования типов

Как выполняются преобразования:

- Переменной типа **bool** присваивается значение другого типа. В этом случае переменная получает **false**, если значение равно 0. Во всех остальных случаях переменная получает **true**
- Числовой или символьной переменной присваивается значение типа **bool**. В этом случае переменная получает 1, если значений равно **true**, либо получает 0, если присваиваемое значение равно **false**.
- Целочисленной переменной присваивается дробное число. В этом случае дробная часть после запятой отбрасывается.
- Переменной, которая представляет тип с плавающей точкой, присваивается целое число. В этом случае если целое число содержит больше битов, чем может вместить тип переменной, то часть информации усекается.

Статическая типизация и преобразования типов

- Переменной беззнакового типа (unsigned) присваивается значение не из его диапазона. В этом случае результатом будет остаток от деления по модулю. Например, тип **unsigned char** может хранить значения от 0 до 255. Если присвоить ему значение вне этого диапазона, то компилятор присвоит ему остаток от деления по модулю 256 (так как тип unsigned char может хранить 256 значений). Так, при присвоении значения -1 переменная типа unsigned char получит 256 - |-1/256| = 255
- Переменной знакового типа (signed) присваивается значение не из его диапазона. В этом случае результат не определен. Программа может работать нормально, выдавая адекватный результат, а может работать некорректно.

Опасные и безопасные преобразования

Те преобразования, при которых не происходит потеря информации, являются безопасными. Как правило, это преобразования от типа с меньшей разрядностью к типу с большей разрядностью. В частности, это следующие цепочки преобразований:

- **bool -> char -> short -> int -> double -> long double**
- **bool -> char -> short -> int -> long -> long long**
- **unsigned char -> unsigned short -> unsigned int -> unsigned long**
- **float -> double -> long double**

Преобразование типов

- Если один из операндов в выражении имеет тип **long double**, то остальные тоже преобразуются к типу **long double**.
- В противном случае, если один из операндов в выражении имеет тип **double**, то остальные тоже преобразуются к типу **double**.
- В противном случае, если один из операндов в выражении имеет тип **float**, то остальные тоже преобразуются к типу **float**.
- В противном случае, если один из операндов в выражении имеет тип **unsigned long**, то остальные тоже преобразуются к типу **unsigned long**.
- В противном случае, если один из операндов в выражении имеет тип **long**, то остальные тоже преобразуются к типу **long**.
- В противном случае, если один из операндов в выражении имеет тип **unsigned**, то остальные тоже преобразуются. к типу **unsigned**.
- В противном случае все операнды преобразуются к типу **int**. При этом тип char преобразуется в int со знаком; тип unsigned char в int, у которого старший байт всегда нулевой; тип signed char в int, у которого в знаковый разряд передается знак из char; тип short в int (знаковый или беззнаковый).

Это автоматическое преобразование типов

Преобразование типов

В языке C++ можно явно указать тип любого выражения. Для этого используется операция преобразования ("приведения") типа. Она применяется следующим образом:

(тип) выражение

Рассмотрим пример:

```
int a = 30000;
float b;
........
b = (float) a * 12;
```

(переменная a целого типа явно преобразована к типу float; если этого не сделать, то результат будет потерян, т.к. **a * 12 > 32767**).

Опасные и безопасные преобразования

Примеры безопасных преобразований:

```
1  short a = 'g'; // преобразование из char в short
2  int b = 10;
3  double c = b; // преобразование из int в double
4  float d = 3.4;
5  double e = d; // преобразование из float в double
6  double f = 35; // преобразование из int в double
```

Но также есть опасные преобразования. При подобных преобразованиях мы потенциально можем потерять точность данных. Как правило, это преобразования от типа с большей разрядностью к типу с меньшей разрядностью. В ряде случаев компиляторы при компиляции выдают предупреждение, тем не менее программа может быть успешно скомпилирована. В других случаях компиляторы не выдают никакого предупреждения. Собственно в этом и заключается опасность, что программа успешно компилируется, но тем не менее существует риск потери точности данных.

ПЕРЕМЕННЫЕ И КОНСТАНТЫ

- ***вещественные,*** например 123.456, 5.61e-4. Они могут снабжаться суффиксом F (или f), например 123.456F, 5.61e-4f;
- ***целые,*** например 125;
- ***короткие целые,*** в конце записи которых добавляется буква (суффикс) H (или h), например 275h, 344H;
- ***длинные целые,*** в конце записи которых добавляется буква (суффикс) L (или l), например 361327L;
- ***беззнаковые,*** в конце записи которых добавляется буква U (или u), например 62125U;

Переменные и константы

- ***восьмеричные,*** в которых перед первой значащей цифрой записывается нуль (0), например 071;
- ***шестнадцатеричные,*** в которых перед первой значащей цифрой записывается пара символов нуль-икс (0x), например 0x5F;
- ***символьные*** - единственный символ, заключенный в одинарные кавычки, например ‘A', '2', '.' и т.д.
- Символы, не имеющие графического представления, можно записывать, используя специальные комбинации, например \n (код 10), \0 (код 0). Эти комбинации выглядят как два символа, хотя фактически это один символ.
- Так же можно представить любой двоичный образ одного байта: '\NNN', где NNN - от одной до трех восьмеричных цифр. Допускается и шестнадцатеричное задание кодов символов, которое представляется в виде: '\x2B', '\x36' и т.п.;

Переменные и константы

- ***строковые*** - последовательность из нуля символов и более, заключенная в двойные кавычки, например: "Это строковая константа". Кавычки не входят в строку, а лишь ограничивают ее. Строка представляет собой массив из перечисленных элементов, в конце которого помещается байт с символом '\0'. Таким образом, число байтов, необходимых для хранения строки, на единицу превышает число символов между двойными кавычками;
- ***константное выражение,*** состоящее из одних констант, которое вычисляется во время трансляции (например: a=60+301);
- **типа *long double,*** в конце записи которых добавляется буква L (или l), например: 1234567.89L.

Директива define

Директива препроцессора define имеет вид

#define имя значение_подстановки

Например,

```
#define max 100
#define rmax (max-1)
```

- Имя, указанное в define, в области его видимости заменяется в тексте программы значением подстановки. Вместо имени max, после препроцессорной обработки, в тесте программы появится 100, а вместо rmax – (100-1).

Модификатор const

Если в объявлении имени присутствует модификатор **const**, то объект, с которым сопоставлено данное имя, рассматривается в области существования этого имени как константа.

Например:

- const int i = 50; // то же что и const i = 50
- const double pi = 3.14159;

Такие именованные константы в программе изменять нельзя.

ОПЕРАЦИИ И ОПЕРАТОРЫ

- Любое выражение языка состоит из операндов (переменных, констант и др.), соединенных знаками операций.
- **Знак операции** - это символ или группа символов, которые сообщают компилятору о необходимости выполнения определенных арифметических, логических или других действий.
- Операции выполняются в строгой последовательности. Величина, определяющая преимущественное право на выполнение той или иной операции, называется ***приоритетом***.

Операции и операторы

Знак операции	Назначение операции
( )	Вызов функции
[ ]	Выделение элемента массива
.	Выделение элемента структуры
->	Выделение элемента структуры
!	Логическое отрицание
~	Поразрядное отрицание
-	Изменение знака
++	Увеличение на единицу
--	Уменьшение на единицу

Операции и операторы

Знак операции	Назначение операции
&	Взятие адреса
*	Обращение по адресу
(тип)	Преобразование типа (т.е. (float) a)
sizeof( )	Определение размера в байтах
*	Умножение
/	Деление
%	Определение остатка от деления
+	Сложение
-	Вычитание
<<	Сдвиг влево
>>	Сдвиг вправо

Операции и операторы

Знак операции	Назначение операции
<	Меньше, чем
<=	Меньше или равно
>	Больше, чем
>=	Больше или равно
= =	Равно
!=	Не равно
&	Поразрядное логическое "И"
^	Поразрядное исключающее "ИЛИ"
|	Поразрядное логическое "ИЛИ"
&&	Логическое "И"
||	Логическое "ИЛИ"
?:	Условная (тернарная) операция
=	Присваивание
+=, - =, *=, /=, %=, <<=, >>=, &=,|=, ^=	Составные операции присваивания (например, a *= b (т.е. a = a * b) и т.д.)
,	Операция запятая

Операции и операторы

- Для исключения путаницы в понятиях "операция" и "оператор", отметим, что **оператор** - это наименьшая исполняемая единица программы.
- Различают операторы ***выражения***, действие которых состоит в вычислении заданных выражений (например: a = sin(b)+c; j++;), операторы ***объявления***, ***составные операторы***, ***пустые операторы***, операторы ***метки***, ***цикла*** и т.д.
- Для обозначения конца оператора в языке C++ используется ***точка с запятой***.

Операции и операторы

- **Комментарий** – это последовательность символов, которая игнорируется компилятором языка C++. Комментарий имеет следующий вид: /*<символы>*/. Комментарии могут занимать несколько строк, но не могут быть вложенными. Кроме того, часть строки, следующая за символами //, также рассматривается как комментарий.
- Удачно подобранный и написанный набор комментариев является существенной частью хорошей программы.

Операции и операторы

Составной оператор ограничивается фигурными скобками {...}

Все другие операторы заканчиваются точкой с запятой. Блок отличается от составного оператора наличием определений в теле блока.

Рассмотрим **операцию присваивания** (=). Выражение вида

```
x = y;
```

присваивает переменной ***x*** значение переменной ***y***. Операцию "=" разрешается использовать многократно в одном выражении, например:

```
x = y = z = 100;
```

Операции и операторы

- Различают ***унарные***, ***бинарные*** *и* ***тернарные*** *операции*. У первых из них один операнд, а у вторых – два, у третьих три.

Операции и операторы

Арифметические операции задаются следующими символами: +, -, *, /, %. Последнюю из них (**%** – остаток от деления) нельзя применять к переменным вещественного типа. Например:

```
a = b + c;
x = y - z;
r = t * v;
s = k / l;
p = q % w;
```

Операции и операторы

Логические операции отношения задаются следующими символами: **&&** ("И"), **||** ("ИЛИ"), **!** ("НЕ"), **>**, **>=**, **<**, **<=** , **==** (равно), **!=** (не равно).

- Традиционно эти операции должны давать одно из двух значений: **истину** или **ложь**.
- В языке C++ принято следующее правило: **истина** - это любое ненулевое значение; **ложь** - это нулевое значение.
- Выражения, использующие логические операции и операции отношения, возвращают 0 для ложного значения и 1 для истинного.

x	y	x&&y	x||y	!x
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Операции и операторы

- ***Битовые операции*** можно применять к переменным, имеющим типы ***int***, ***char***, а также их вариантам (например, ***long int***).
- Их нельзя применять к переменным типов ***float***, ***double***, ***void*** (или более сложных типов).
- Эти операции задаются следующими символами:

  ~ (поразрядное отрицание),

  << (сдвиг влево),

  >> (сдвиг вправо),

  & (поразрядное "И"),

  ^ (поразрядное исключающее "ИЛИ"),

  | (поразрядное "ИЛИ").

Операции и операторы

Примеры: если **a = 0000 1111** и **b = 1000 1000**,

~a = 1111 0000,

a << 1 = 0001 1110,

a >> 1 = 0000 0111,

a & b = 0000 1000,

a ^ b = 1000 0111,

a | b = 1000 1111.

Операции и операторы

В языке предусмотрены две ***нетрадиционные операции инкремента*** (++) и декремента (--).

Они предназначены для увеличения и уменьшения на единицу значения операнда. Операции ++ и -- можно записывать как перед операндом, так и после него. В первом случае (++n или --n) значение операнда (n) изменяется перед его использованием в соответствующем выражении, а во втором (n++ или n--) - после его использования.

Операции и операторы

Рассмотрим две следующие строки программы:

```
a = b + c++;
a1 = b1 +  ++c1;
```

Предположим, что b = b1 = 2, c = c1 = 4.

Тогда после выполнения операций:

a = 6, b = 2, c = 5, a1 = 7, b1 = 2, c1 = 5.

Операции и операторы

Широкое распространение находят также выражения с еще одной нетрадиционной тернарной или условной операцией “**?:**”.

В формуле **y = x ? a: b**;

- y = a, если x не равно нулю (т.е. истинно), и y = b, если x равно нулю (ложно).

Выражение **y = (a>b) ? a: b**;

- позволяет присвоить переменной y значение большей переменной (a или b), т.е. y = max(a, b).

Операции и операторы

- Еще одним отличием языка является то, что выражение вида a = a + 5; можно записать в другой форме: a += 5;. Вместо знака + можно использовать и символы других бинарных операций.

Операторы условных и безусловных переходов

Для организации условных и безусловных переходов в программе на языке C++ используются операторы:

if else, *switch* и *goto*.

Оператор ***if*** записывается следующим образом:

```
if (проверка_условия) оператор_1; else оператор_2;
```

- Если условие в скобках принимает истинное значение, выполняется оператор_1, если ложное - оператор_2. Если вместо одного необходимо выполнить несколько операторов, то они заключаются в фигурные скобки. В операторе if слово else может отсутствовать.
- Слово **else** всегда относится к ближайшему предшествующему ему **if** **!!!** (для вложенных)

09:41

Операторы условных и безусловных переходов

Пример: ввести три числа и вывести максимальное из них по значению

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main() {
6     int a, b, c;
7      cin >> a >> b >> c;
8      if(a >= b && a >= c){
9          cout << a << endl;
10     }
11     else if(b >= a && b >= c){
12         cout << b << endl;
13     }
14     else{
15         cout << c << endl;
16     }
17
18     return 0;
19
20 }
```

Операторы условных и безусловных переходов

- Оператор ***switch*** позволяет выбрать одну из нескольких альтернатив. Он записывается в следующем формальном виде:

```
switch (выражение)
{
        case константа_1:  операторы_1;
                           break;

        case константа_2:  операторы_2;
                           break;
        ........           ........
        default:           операторы_default;
}
```

Операторы условных и безусловных переходов

- ***Оператор безусловного перехода*** можно представить в следующей форме:

```
goto метка;
```

- ***Метка*** - это любой идентификатор, после которого поставлено двоеточие. Оператор *goto* указывает на то, что выполнение программы необходимо продолжить начиная с оператора, перед которым записана метка. Метку можно поставить перед любым оператором в той функции, где находится соответствующий ей оператор *goto*. Ее не надо объявлять.
- Современным стилем программирования рекомендуется использовать оператор безусловного перехода в основном для выхода сразу из всех вложенных циклов, в иных случаях целесообразно воспользоваться операторами циклов и условных переходов

Операторы условных и безусловных переходов

- ***Оператор return*** завершает выполнение функции, в которой он задан и возвращает управление в вызывающую функцию. Управление передается в вызывающую функцию в точку, непосредственно следующую за вызовом завершенной функции.
- Если оператор **return** присутствует в функции main(), то он вызывает прерывание выполнения программы.

Операторы цикла

Циклы организуются, чтобы выполнить некоторый оператор или группу операторов определенное число раз. В языке Си три оператора цикла: ***for***, ***while*** и ***do while***. Первый из них формально записывается, в следующем виде:

```
for (выражение_1; выражение_2; выражение_3) тело_цикла
```

Тело цикла составляет либо один оператор, либо несколько операторов, заключенных в фигурные скобки { ... } (после блока точка с запятой не ставится). В выражениях 1, 2, 3 фигурирует специальная переменная, называемая управляющей. По ее значению устанавливается необходимость повторения цикла или выхода из него.

Выражение_1 присваивает начальное значение управляющей переменной, **выражение_3** изменяет его на каждом шаге, а **выражение_2** проверяет, не достигло ли оно граничного значения, устанавливающего необходимость выхода из цикла.

Операторы цикла

Пример:

```
for (i = 1; i < 10; i++)
{    ...
}
```

```
for (ch = 'a'; ch != 'p';) scanf ("%c", &ch);
/* Цикл будет выполняться до тех пор, пока с клавиатуры
   не будет введен символ 'p' */
```

Любое из трех выражений в цикле **for** может отсутствовать, однако точка с запятой должна оставаться. Таким образом, for (; ;) {...} - это бесконечный цикл, из которого можно выйти лишь другими способами.

Операторы цикла

Оператор ***while*** формально записывается в таком виде:

```
while (выражение) тело_цикла
```

Выражение в скобках может принимать ненулевое (истинное) или нулевое (ложное) значение. Если оно истинно, то выполняется тело цикла до тех пор, пока выражение в скобках не примет ложное значение. Если оно ложно при входе в цикл, то его тело не выполнится ни разу.

Операторы цикла

Вводится последовательность чисел. Признак конца ввода - число 777. Вывести среднее арифметического значение последовательности чисел *(реализация на языке C++):*

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

int main() {
    float sum = 0;
    float counter = 0;
    int a;
    cin >> a;
        if (a == 777)
     cout << "There is no one element!" << endl;
        else {
    while (a != 777){
        ++counter;
        sum += a;
        cin >> a;
    }
    float mid = sum / counter;
    cout << mid << endl;
        }
    return 0;
}
```

Операторы цикла

Вводится последовательность чисел. Признак конца ввода - число 777. Вывести среднее арифметического значение последовательности чисел *(реализация на языке Си):*

```c
#include<stdio.h>

int main() {
    int x, sum = 0, count = 0;
    // float average;
    scanf("%d", &x);
    if (x == 777)
        printf("There is no one element!\n");
    else {
        while (x != 777) {
            sum += x;
            count++;
            scanf("%d", &x);
        }
        // average = (float)sum/(float)count;
        printf("Average of elements is: %f\n", (float)sum/(float)count);
    }
    return 0;
}
```

Операторы цикла

Оператор ***do while*** формально записывается следующим образом:

```
do {тело_цикла} while (выражение);
```

Основным отличием между циклами *while* и *do while* является то, что тело в цикле *do while* выполняется по крайней мере один раз. Тело цикла будет выполняться до тех пор, пока выражение в скобках не примет ложное значение. Если оно ложно при входе в цикл, то его тело выполняется ровно один раз.

Операторы цикла

Допускается вложенность одних циклов в другие, т.е. в теле любого цикла могут появляться операторы *for*, *while* и *do while*.

В теле цикла могут использоваться операторы ***break*** и ***continue***. Оператор *break* обеспечивает немедленный выход из цикла, оператор *continue* вызывает прекращение очередной и начало следующей итерации.

Пример (Гипотеза Коллатца)

Берём любое натуральное число n. Если оно чётное, то делим его на 2, а если нечётное, то умножаем на 3 и прибавляем 1 (получаем 3n + 1). Над полученным числом выполняем те же самые действия, и так далее.

Гипотеза Коллатца заключается в том, что какое бы начальное число n мы ни взяли, рано или поздно мы получим единицу.

Необходимо построить в программе описываемую последовательность для любого введенного числа n.

Пример (Гипотеза Коллатца)

```
1  #include <iostream>
2
3  using namespace std;
4
5    void collatz( unsigned long val ) { // гипотеза Коллатца
6       cout << val << " => [";
7       while( true ) {
8          val = val & 1 ? 3 * val + 1 : val >> 1;
9          cout << val << ( val > 1 ? "," : "]\n" );
10         if( 1 == val ) return;
11 } }
12   int main() {
13      unsigned long d;
14      cout << "Введите натуральное число: ";
15          cin >>d;
16         collatz( d );
17      }
```

Здесь условный оператор записан в виде тернарной или условной операции “**?:**”, а деление на 2 выполняется сдвигом числа на один разряд вправо.
Для проверки числа на четность используется поразрядная логическая операция ”И”, с помощью которой проверяется младший бит числа.

Литература и Интернет-ресурсы

Основная литература (О)- доступная из электронной библиотеки РЭУ:

Алгоритмизация и программирование : Учебное пособие / С.А. Канцедал. - М.: ИД ФОРУМ: НИЦ ИНФРА-М, 2013. - 352 с.: ил.; 60x90 1/16. - (Профессиональное образование). (переплет) ISBN 978-5-8199-0355-1 - Режим доступа: http://znanium.com/catalog/product/391351

А.В.Кузин, Е.В.Чумакова. Программирование на языке Си. – М.: Форум, НИЦ ИНФРА-М, 2015. – 144 с.: ISBN 978-5-00091-066-5

Воронцова Е.А. Программирование на C++ с погружением: практические задания и примеры кода - М.:НИЦ ИНФРА-М, 2016. - 80 с.: 60x90 1/16 - Режим доступа: http://znanium.com/catalog/product/563294

Нормативно-правовые документы:

ГОСТ 19.701-90 Схемы алгоритмов, программ, данных и систем

Дополнительная литература (Д):

Царев, Р. Ю. Программирование на языке Си [Электронный ресурс] : учеб. пособие / Р. Ю. Царев. - Красноярск : Сиб. федер. ун-т, 2014. - 108 с. - ISBN 978-5-7638-3006-4 - Режим доступа: http://znanium.com/catalog.php?bookinfo=510946

Основы программирования на языке С: Учебное пособие / В.Г. Дорогов, Е.Г. Дорогова; Под общ. ред. проф. Л.Г. Гагариной - М.: ИД ФОРУМ: ИНФРА-М, 2011. - 224 с.: 60x90 1/16. - (Высшее образование). (переплет) ISBN 978-5-8199-0471-8 - Режим доступа: http://znanium.com/catalog/product/225634

Литература и Интернет-ресурсы

Перечень электронно-образовательных ресурсов

Курс "Информатика и программирование" – авторы Комлева Н.В., Иванов Е.А. https://www.rea.ru/ru/org/managements/electedu/Documents/EOS/EOR_12.04.2019.pdf

Перечень информационно-справочных систем

http://www.garant.ru - Консультант Плюс;

http://www.consultant.ru/ - Гарант.

Перечень ресурсов информационно-телекоммуникационной сети "Интернет", необходимых для освоения дисциплины (модуля)

http://ocw.mit.edu

http://www.citforum.ru

http://intuit.ru

http://rsdn.org

http://computersbooks.net/index.php?id1=4&category=language-programmer&author=podelskiy-vv&book=2003