Элементы языка C++.

Алфавит

Множество символов языка C++ включает:

- буквы латинского алфавита
  - * большие **A, B, C, …, Z**
  - * маленькие **a, b, c, …, z**
- цифры **0, 1, 2, …, 9**
- пробел
- знаки препинания . :
- скобки ( ) [ ] { }
- знаки | ^ ? _
- двойные и одинарные кавычки " '

C++ чувствителен к регистру. Это означает, что "a" и "A" — совсем разные буквы.

Список ключевых слов языка C++:

asm	char	delete	extern
auto	class	do	float
break	const	double	for
case	continue	else	friend
catch	default	enum	goto
if	protected	static	typedef
inline	public	struct	union
int	register	switch	unsigned
long	return	template	virtual
new	short	this	void
operator	signed	throw	volative
private	sizeof	try	while

Зарезервированные (ключевые) слова **запрещается** использовать в качестве пользовательских имен переменных.

Комментарии

Комментарии не обрабатываются компилятором, и поэтому не влияют на выполнение программы.

В языке C++ можно записывать комментарии двух видов:

- **// комментарий kommentaarid**

или

- **/* комментарий**
  комментарий
  комментарий */

Переменные

В процессе написания программы приходится использовать ***переменные***.

Каждая переменная имеет **имя, тип, размер и значение.**

Имя переменной (идентификатор) является ее названием. Имя переменной может состоять из латинских букв, цифр и символа подчеркивания.

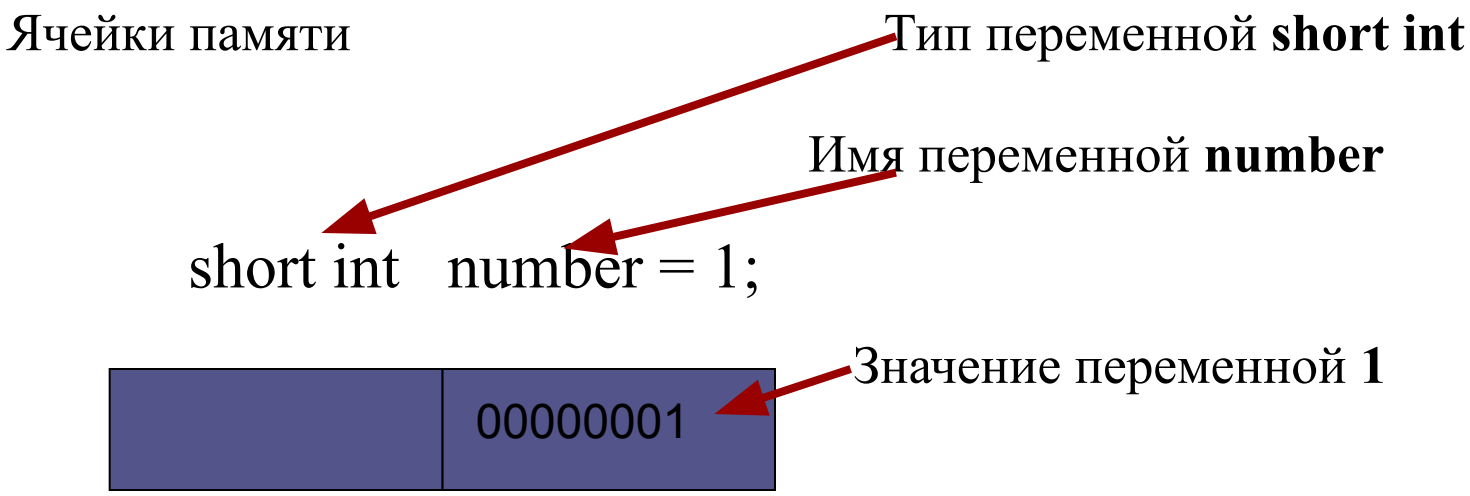
Например, **birth_date salary summa3 summa_33**
_1Tase _Tase_1

Тип определяет, какие символы или числа записаны в ячейку памяти под этим именем.

Размер указывает максимальную величину или точность задания числа.

Значение определяет содержимое ячейки памяти.

Переменная — это именованная область памяти, к которой программист имеет доступ из программы по имени переменной.



Типы данных

1. ***Встроенные типы данных,*** т. е. типы, предопределенные в языке программирования.
2. ***Составные типы*** обладают той особенностью, что в языке предопределены средства распознавания таких типов и некоторый набор операций, дающих возможность доступа к компонентам составных значений.
3. 3. ***Указательные типы*** дают возможность работы с типизированными множествами абстрактных адресов переменных, содержащих значения некоторого типа. В языках с более слабой типизацией (например, C и C++) допускаются практически неограниченные манипуляции с указателями.

В C++ реализован набор базовых типов данных:

Имя типа	Размер в байтах	Допустимое значение
char	1 байт	От **-128** до **127**
int	Зависит от реализации	2 или 4 байта
short int	2 байта	От **-32 768** до **32 767**
long int	4 байта	От **-2 147 483 648** до **2 147 483 647**
unsigned char	1 байт	От **0** до **255**
unsigned int	Зависит от реализации	
unsigned short int	2 байта	От **0** до **65 535**
unsigned long int	4 байта	От **0** до **4 294 967 295**
float	4 байта	От **~3.4e-38** до **~3.4e + 38**
double	8 байтов	От **~1.7e – 308** до **~1.7e + 308**
long double	10 байтов	От **~3.4e – 4932** до **~1.1e + 4932**
bool	1 байт	false (0), true (любое число, отличное от нуля)

Объявление переменной.

```
unsigned int A;
int B;          /* подразумевается signed int B */
unsigned C;  /* подразумевается unsigned int C*/
```

Можно по одному типу определять несколько переменных через запятую.

Например:

```
int A, B, kokku, tulemus_1;
char  C;  /* подразумевается mõeldakse unsigned int C*/
```

Если объект определен как глобальный, C++ гарантирует, что он будет инициализирован нулевым значением.

Если же переменная локальная либо динамическая, ее начальное значение не определено, т. е. она может содержать некоторое случайное значение.

NB!!! Рекомендуется явно указывать начальное значение переменной

```
int  A = 5,
     B = 0,
     kokku = 100,
     tulemus_1 = 6;
```

Преобразование типов

При выполнении операций производится **автоматическое преобразование типов**, чтобы привести операнды выражений к общему типу или чтобы расширить короткие величины до размера целых величин, используемых в машинных командах.

- операнды типа **float** преобразуются к типу **double**

- если один операнд — **long double**, то второй преобразуется к этому же типу

- если один операнд — **double**, то второй также преобразуется к типу **double**

- любые операнды типа **char** и **short** преобразуются к типу **int**

- любые операнды **unsigned char** или **unsigned short** преобразуются к типу **unsigned int**

- если один операнд — **unsigned long**, то второй преобразуется к типу **unsigned long**

- если один операнд — long, то второй преобразуется к типу long

- если один операнд — unsigned int, то второй операнд преобразуется к этому же типу.

Таким образом, при вычислении выражений операнды преобразуются к типу того операнда, который имеет наибольший размер.

Константы

Константы, в отличие от переменных, не могут изменяться программой.

Записываются они по следующим правилам:

- вещественные константы можно записать
  а) в обычной форме (0.54)
  в) в экспоненциальной форме 0.5e1, -0.12345e + 2, 987e-5)
- целочисленные константы можно записать
  а) в десятичной (-2 6)
  в) шестнадцатеричной системе счисления (0x1a);
- символьные константы записываются в одинарных кавычках ('5', 'A', '\t', '\r');
- строковые константы записываются в двойных кавычках ("Line").

Константы можно определить одним из следующих способов:

1. записать в выражении

c = a + 7;

2. с помощью ключевого слова const.

const тип Long=5;

Тогда в выражениях вместо константы 5 указывается идентификатор Long.

3. с помощью директивы препроцессора

#define Long 5

Директива заменяет каждое появление символов Long на 5.

Арифметические и логические операции

Арифметическое сложение	+
Арифметическое вычитание	-
Умножение	*
Деление	/
Отрицание	!
Присваивание	=
Вычисление остатка	%
Логическое умножение	&& и
Логическое сложение	| | или
Проверка на равенство	= =
Проверка на неравенство	! =
Проверка на больше	>
Проверка на меньше	<
Проверка на больше или равно	>=
Проверка на меньше или равно	<=

Директивы препроцессора

Обработка программы препроцессором происходит перед ее компиляцией.

Управление работой препроцессора осуществляется при помощи его директив.

Директивы препроцессора выполняются до трансляции программы.

Директивы препроцессора позволяют изменить текст программы, например, заменить некоторые лексемы в тексте, вставить текст из другого файла, запретить трансляцию части текста и т. п.

#include включает в текст программы содержимое указанного файла.

#define создаёт псевдонимы

#undef отмена действия директивы #def ine.

#error печатает в процессе компиляции сообщение об ошибке

#typedef позволяет задать синоним для встроенного либо пользовательского типа данных.

Все директивы препроцессора начинаются со знака #. После директив точка с запятой не ставится.

Директива *#include*

Директива #include включает в текст программы содержимое указанного файла.

Эта директива имеет две формы

#include *"имя_файла"* ***поиск файла осуществляется в соответствии с заданным маршрутом, а при его отсутствии — в текущем каталоге.***

#include <*имя_файла*> ***поиск файла производится в стандартных каталогах операционной системы, задаваемых командой path.***

Директива **#include** имеет следующие свойства

- она может быть вложенной, т. е. во включаемом файле тоже может содержаться директива #include.

- она используется для включения в программу *заголовочных файлов*.

```
#include <iostream>      //подключает заголовочный файл для
                         //работы с потоковым вводом и
//выводом информации.
```

Директива *#define*

Служит для замены часто использующихся констант, ключевых слов, операторов или выражений некоторыми идентификаторами (псевдонимами).
Идентификаторы, заменяющие текстовые или числовые константы, называют ***именованными константами.***
Идентификаторы, заменяющие фрагменты программ, называют ***макроопределениями,*** причем макроопределения могут иметь аргументы.

Директива #define

#define *идентификатор текст*

```
#define My_type unsigned short int
#define    Pi   3.14
```

Директива *#undef*

Директива #undef используется для отмены действия директивы #def ine.

```
#undef My_type
```

Директива отменяет действие текущего определения #def #define для указанного идентификатора.
Не является ошибкой использование директивы #undef для идентификатора, который не был определен директивой #define.

Директива *#error*

Директива препроцессора #error имеет следующий синтаксис:

#error msg

Директива печатает в процессе компиляции сообщение об ошибке вида:

Error: *filename* line# : Error directive: *msg*

Здесь *msg* — сообщение, заданное директивой #error.

Директива **#typedef**

Данная директива позволяет задать синоним для встроенного либо пользовательского типа данных.
Например:

```
#typedef double Wages;
#typedef  int Coll;
```

Имена, определенные с помощью директивы #typedef, можно использовать точно так же, как спецификаторы типов:

```
#typedef   double   Wages;  //тип  double теперь называется Wages
main()
{    //объявляем переменные типа Wages
     Wages i,  j,  k;
     i=j+k;
}
```

Ввод/вывод

В ходе работы приложения необходимо вводить некоторые данные и получать ответ — так называемые выходные данные.
В C++ ввод и вывод на экран осуществляется при помощи команд **cin** и **cout** соответственно.
Чтобы использовать команды **cin** и **cout**, необходимо подключить заголовочный файл:

#include <iostream.h> или **#include <iostream>**

Форматы команд:

cout<<element1<<element2<<. . .<<elementN;

cin>>element1>>element2>>. . .>>elementN;

В качестве элемента могут использоваться следующие величины:

1. **переменная** одного из указанных выше типов

cout<<My_res;

2. **строковая константа**, т. е. текст в двойных кавычках.

cout<<“Programmi tulemused:”;

3.числовая константа

cout<<5;

4.выражение, заключенное в круглые скобки

cout<<(a+b);

5.ключевое слово **endl**, которое означает, что

после этого слова информация будет выводиться с новой строки,

cout<< endl;

В тексте можно использовать *управляющие символы*, признаком которых является обратный слэш «\».

\n означает, что информация, выводимая после него, будет размещена на следующей строке.

\t табуляция

\f перевод страницы

\r возврат каретки

\v вертикальная табуляция

\’ одиночная кавычка

\” двойные кавычки

\? вопросительный знак

**** обратный слэш

\xhhh шестнадцатиричное число

Есть ли отличие **\n** от **endl** ?


```
#include <iostream>   // Заголовочный файл
int main() {
// 2 переменные 2 muutujad
      int a=2,   b=5;
      cout<<"\n summa "<<(a+b)<<"\n raznost “<<(a-b);
//  или
      cout<<endl<<" summa “<<(a+b) <<endl<<"
raznost"<<(a-b);
return 0;    //  Завершает программу lõpeb programmi  }
```

Результат выполнения вывода обоими способами будет одинаковый.

printf()

Функция вывода printf () имеет следующий синтаксис:

printf (строка, список*);*

Строка:

- **любая последовательность символов**, включая пробелы, которые точно в таком же виде будут отображаться при выводе на экран;
- **управляющие символы или символьные константы с обратным слэшем**:
  - \n — переход на следующую строку;
  - \r — возврат каретки, т. е. переход в начало текущей строки;
  - \t — горизонтальная табуляция, т. е. вывод с определенных стандартных позиций;
  - \a — непродолжительный звуковой сигнал;
- **форматы, или спецификаторы формата**. Форматы указывают способ вывода соответствующих элементов списка и зависят от типов выводимых значений:
  - %f — для вывода вещественных чисел;
  - %d — для вывода целых чисел;
  - %e или %e — для вывода вещественных чисел в экспоненциальной форме, например, o.se-4 или o.5e-4;

%g — выбирает в зависимости от значения числа формат % f или % e;

%д — выбирает в зависимости от значения числа формат %f или % e;

%x или %x — для вывода шестнадцатеричных чисел;

%c — для вывода символа;

%s — для вывода строки.

Перед спецификаторами форматов можно использовать **модификаторы форматов** для изменения способа представления выводимых значений: %nd, где целая константа n определяет ширину поля для выводимого целого числа.

Например, printf ("%5d%05d", i, number) выводит справа поля шириной 5 со стоящими впереди пробелами для i и нулями для number.

Вся строка с указанными выше элементами записывается, как правило, в виде текстовой константы. Можно использовать для этих целей переменную строкового типа.

В **списке** функции printf () можно записать через запятую выражения, значения которых выводятся. Как частный случай выражения можно записать константу или переменную. Между элементами списка и спецификаторами формата с символом % должно быть соответствие в количестве, порядке следования и типе.

Любой из элементов строки или список могут отсутствовать.

Инкремент и декремент.

++ Инкремент – увеличение значения переменной на 1.

-- Декремент – уменьшение значения переменной на 1.

A++; увеличит A на 1

A--; уменьшит A на 1

Равносильны следующие выражения:

A++; **A = A + 1;** **A + =1;**

Префикс и постфикс

Операторы инкремента и декремента работают в 2-х вариантах:

- префиксном **++A;**
- постфиксном **A++;**

Префиксный оператор вычисляется до присваивания, а постфиксный – после.

Префиксный оператор работает так: инкрементируем значение, а затем считываем его.

Постфиксный оператор работает так: считываем значение, а затем инкрементируем его.

Пример. x – целое число, равное 5.

Если написано **int a = ++x;** , то сначала увеличиваем x на 1 (это 6), а затем присваиваем “a”. Следовательно, значение переменной “a” теперь равно 6 и значение x тоже равно 6.

Если после этого написать **int b = x++;** , то переменной “b” присваивается значение x=6, а затем возвращаемся к **x** и инкрементируем её, т.е. **x** стало равным 7.

Объявить три переменные типа int и присвоить первой числовое значение, вторая переменная равна первой переменной увеличенной на 3, а третья переменная равна сумме первых двух.

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

int main()
{
int first = 4;
int second = first + 3;
int third = first + second;

cout << "first = " << first << endl;
cout << "second = " << second << endl;
cout << "third = " << third << endl;

return 0;
}
```

Объявить переменные, для подсчета общего количества предметов для сервировки стола. Например чашки, такое же количество блюдец и ложек.

```
   #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8  setlocale(LC_ALL, "rus");
9  int cups = 6; // количество чашек
1  int spoons = cups; // количество ложек равно кол-тву чашек
0  int saucers = cups; // блюдца
1
1  int amount = cups + spoons + saucers;  // общее количество
1
2  cout << "Всего " << amount << " предметов" << endl;
1
3  return 0;
1  }
4
```

Создайте 4 переменные с разными типами данных и предложите пользователю ввести в них значения. После ввода, отобразите их на экране.

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

int main()
{
setlocale(LC_ALL, "rus");

int digit = 0;
double digit2 = 0;
char symbol = 0;
bool trueOrFalse = 0;

cout << "Введите целое число: ";
cin >> digit;
cout << "Введите вещественное число: ";
cin >> digit2;
cout << "Введите символ: ";
cin >> symbol;
// в переменную типа bool  с помощью cin можно ввести
// только числа 0 (интерпретируется как false) и 1 (true)
cout << "Введите 0 или 1: ";
cin >> trueOrFalse;

cout << endl << endl;
cout << "Целое число: " << digit << endl;
cout << "Вещественное число: " << digit2 << endl;
cout << "Символ: " << symbol << endl;
cout << "bool: " << trueOrFalse << endl;

return 0;
}
```

Самостоятельно

Создайте 5 переменных типа char, предложите пользователю ввести слово из пяти букв и покажите эти символы (слово) на экран. (Символы вводить латиницей, т.к. кирилица будет отображаться некорректно. Почему? Это мы рассмотрим в одном из наших следующих уроков)

s");