

Модуль 2. Простейшие программы. Ввод-вывод. Операции в выражениях. Оператор присваивания.

**Структура простейшей программы,
переменные, операции, выражения,
преобразования базовых типов, вводные
замечания о средствах ввода-вывода,
особенности вывода кириллицы в
консольное окно.**

Описание идентификаторов

[класс памяти] [const] тип имя [инициализатор] ;

инициализатор: = значение или (значение)

Примеры описаний:

```
short int a = 1;  
const char C = 'C';  
char s, sf = 'f';  
float c = 0.22, sum;
```

Область видимости

Каждый идентификатор имеет область действия (**potential scope**) и область видимости (**scope**), которые, как правило, совпадают (кроме случая описания такого же имени во вложенном блоке).

- Область видимости начинается в точке описания.

```
const int i = 2; { int i[i]; }
```

Имя, описанное внутри блока, локально по отношению к этому блоку. Имя, описанное вне любого блока, имеет глобальную область видимости.

Область действия и класс памяти зависят не только от собственно описания, но и от места его размещения в тексте программы.

Класс памяти

auto — *автоматическая* переменная. Память выделяется в стеке и при необходимости инициализируется каждый раз при выполнении оператора, содержащего ее определение. Освобождение памяти - при выходе из блока

extern — переменная определяется в другом месте программы. Extern double PI=3.1415926;

static — *статическая* переменная. Время жизни — постоянное. Инициализируется один раз при первом выполнении оператора, содержащего определение переменной. В зависимости от расположения оператора описания статические переменные могут быть *глобальными и локальными*.

register — аналогично auto, но память выделяется по возможности в регистрах процессора.

Область видимости. Пример 1

```
int a;                // 1
main() {
    int b;            // 2
    extern int x;      // 3
    static int c;      // 4
    a = 1;            // 5
    int a;            // 6
    a = 2;            // 7
    ::a = 3;          // 8
}
int x = 4;            // 9
```

Область видимости. Пример 2

```
int a;           // глобальная переменная

int main(){
    int b;       // локальная переменная
    static int c = 1; // локальная статическая переменная
}
```

	Глобальная	Локальная	Статическая
Размещение	с-т данных	с-т стека	с-т данных
Время жизни	вся прогр.	блок	вся прогр.
Область видимости	файл	блок	блок
Обнуление	да	нет	да

Пространства имен

В каждой области действия различают пространства имен, в пределах которых идентификатор должен быть уникальным. В разных категориях имена могут совпадать, например:

```
struct Node{  
    int Node;  
    int i;  
}Node;
```

В C++ определено четыре отдельных класса идентификаторов, в пределах которых имя должно быть уникальным:

1. имена переменных, функций, типов typedef и констант перечислений;
2. имена типов перечислений, структур, классов и объединений;
3. элементы каждой структуры, класса и объединения;
4. метки.

Пример 1 - простейшая программа

```
#include <stdio.h>

int main() {
    int i;
    printf("Введите целое число\n");
    scanf("%d", &i);
    printf("Вы ввели число %d, спасибо!", i);
}
```

В первой строке записана директива препроцессора, вставляющая содержимое файла `stdio.h` с описаниями используемых в программе элементов стандартной библиотеки ввода/вывода в начало нашей программы. Строка 2 представляет собой заголовок главной (и единственной) функции программы. В 3 строке описывается переменная целого типа с именем `i`. В 4 строке используем функцию вывода с именем `printf`. Она выводит приглашение к набору на клавиатуре числа и курсор переводится на следующую строку в соответствии с управляющей последовательностью `\n`. В 5 строке функция с именем `scanf` читает набранное число и записывает его в ячейку памяти с именем `i`. Знак `&` обозначает операцию получения адреса переменной `i`. Это нужно чтобы функция `scanf` могла записать в эту переменную введенное с клавиатуры значение. В 6 строке функция вывода выводит благодарность за ввод числа и само число из ячейки памяти `i`.

Стандартные функции ввода-вывода

// ВВОД

int scanf (“спецификации”, список адресов вводимых величин) ;

// ВЫВОД

int printf(“текст со спецификациями”, список выводимых величин);

Спецификации:

%d, %i – для целых десятичных чисел

%f - для величин типа float,

%lf - для величин типа double

%c – для символов

%s – для строк

%u – для беззнаковых целых чисел

%e,%E – для чисел с плавающей точкой

\n - управляющий символ новая строка;

\t – табуляция;

\a – звуковой сигнал

Модификатор

%[-]m[.p]C

- выравнивание по левому краю

m – минимальная ширина поля

p – количество цифр после запятой

C – спецификация формата

Пример 2 - целые форматы

```
#include <stdio.h>

int main(){
    int int1 = 45, int2 = 13;
    printf("int1 = %d| int2 = %3d| int2 = %4d|\n",
        int1, int2, int2);
    printf("int1 = %x| int2 = %3x| int2 = %4o|\n",
        int1, int2, int2);
}
```

```
int1 = 45| int2 = 13| int2 = 13 |
int1 = 2D| int2 = d| int2 = 15|
```

Пример 3 - вещественные форматы

```
#include <stdio.h>
int main() {
float f = 3.621;
double dbl = 2.23;
printf("f = %f| f = %4.2f| f = %6.1f|\n", f, f, f);
printf("f = %g| f = %e| f = %+E|\n", f, f, f);
printf("dbl = %5.2lf| dbl = %e| dbl = %4.1G|\n",
      dbl, dbl, dbl);
}
```

```
f = 3.621000| f = 3.62| f = 3.6|
f = 3.621| f = 3.621000e+000| f = +3.621000E+000|
dbl = 2.23| dbl = 2.230000e+000| dbl = 2|
```

Пример 4 - форматы символов и строк

```
#include <stdio.h>

int main(){
char ch = 'z', *str = "I am a student";
printf("ch = %c| ch = %3c|\n", ch, ch);
printf("str = %-18s|\nstr = %s|\n", str, str);
}
```

```
ch = z| ch =   z|
str = I am a student |
str = I am a student|
```

Пример 5 – Поточковый ВВОД-ВЫВОД

```
#include <iostream.h>

int main() {
    int i;
    cout << "Введите целое число\n";
    cin >> i;
    cout << "Вы ввели число" << i << ", спасибо!";
}
```

Особенности вывода кириллицы

Русские символы, набираемые в окне редактора среды, имеют другую кодировку, и поэтому в консольном окне могут быть видны как "иероглифы". Если хочется использовать русские буквы, проще всего воспользоваться функцией `setlocale`, унаследованной из языка C (она описана в заголовочном файле `<locale>`):

```
setlocale( LC_ALL, "Russian" );
setlocale( LC_ALL, "rus" );
```

Чтобы русификация консольного окна работала, необходимо, чтобы в операционной системе Windows в настройках языков и стандартов была установлена кириллица по умолчанию.

Выражения. Основные операции C++

Примеры выражений:

$(a + 0.12) / 6$, $x \&\& y \parallel !Z$,

$(t * \sin(x) - 1.05e4) / ((2 * k + 2) * (2 * k + 3))$, $a = b = c$,
 $b++$

Унарные операции

$++$ $--$ $sizeof$ \sim $!$ $-$ $+$ $\&$ $*$ new $delete$ (type)

Бинарные операции

$*$ $/$ $\%$ $+$ $-$ $<<$ $>>$ $<$ $<=$

$>$ $>=$ $==$ $!=$ $\&$ $^$ $|$ $\&\&$ \parallel $=$ $*=$ $/=$ $\%=$ $+=$ $-=$ $<<=$

$>>=$ $\&=$ $|=$ $^=$ $throw$,

Тернарная операция

$?:$

Приоритеты операций

Операция	Краткое описание
Унарные операции	
::	доступ к области видимости
.	выбор
->	выбор
[]	индексация
()	вызов функции
<тип>()	конструирование
++	постфиксный инкремент
--	постфиксный декремент
typeid	идентификация типа
dynamic_cast	преобразование типа с проверкой на этапе выполнения
static_cast	преобразование типа с проверкой на этапе компиляции
reinterpret_cast	преобразование типа без проверки
const_cast	константное преобразование типа

Приоритеты операций

sizeof	размер объекта или типа
--	префиксный декремент
++	префиксный инкремент
~	поразрядное отрицание
!	логическое отрицание
-	арифметическое отрицание (унарный минус)
+	унарный плюс
&	взятие адреса
*	адресация
new	выделение памяти
delete	освобождение памяти
(<тип>)	преобразование типа

Приоритеты операций

.*	выбор
->*	выбор
Бинарные и тернарная операции	
*	умножение
/	деление
%	остаток от деления
+	сложение
—	вычитание
<<	сдвиг влево
>>	сдвиг вправо

<	меньше
<=	меньше или равно
>	больше
>=	больше или равно
==	равно
!=	не равно
&	поразрядная конъюнкция (И)
^	поразрядное исключающее ИЛИ
	поразрядная дизъюнкция (ИЛИ)
&&	логическое И
	логическое ИЛИ
? :	условная операция (тернарная)
=	присваивание
*=	умножение с присваиванием
/=	деление с присваиванием

<code>%=</code>	остаток от деления с присваиванием
<code>+=</code>	сложение с присваиванием
<code>-=</code>	вычитание с присваиванием
<code><<=</code>	сдвиг влево с присваиванием
<code>>>=</code>	сдвиг вправо с присваиванием
<code>&=</code>	поразрядное И с присваиванием
<code> =</code>	поразрядное ИЛИ с присваиванием
<code>^=</code>	поразрядное исключающее ИЛИ с присваиванием
<code>throw</code>	исключение
<code>,</code>	последовательное вычисление

Операции выполняются в соответствии с *приоритетами*. Для изменения порядка выполнения операций используются круглые скобки. Если в одном выражении записано несколько операций одинакового приоритета, унарные операции, условная операция и операции присваивания выполняются *справа налево*, остальные — *слева направо*.

Операции инкремента и декремента

```
#include <stdio.h>

int main() {
    int x = 3, y = 3;
    printf("Значение префиксного выражения: %d\n", ++x);
    printf("Значение постфиксного выражения: %d\n", y++);
}
```

Результат работы программы:

Значение префиксного выражения: 4

Значение постфиксного выражения: 3

Операция sizeof

sizeof выражение

sizeof (тип)

```
#include <iostream.h>
int main(){
float x = 1;
cout << "sizeof (float) :" << sizeof (float);
cout << "\nsizeof x :" << sizeof x;
}
```

sizeof (float) : 4

sizeof x : 4

Поразрядные (побитовые) операции

```
#include <iostream.h>

int main() {
    cout << "\n 6&5 = " << (6&5);
    cout << "\n 6|5 = " << (6|5);
    cout << "\n 6^5 = " << (6^5);
}
```

Результат работы программы:

```
6&5 = 4      ( 110 & 101 = 100 )
6|5 = 7      ( 110 | 101 = 111 )
6^5 = 3      ( 110 ^ 101 = 011 )
```


Операции деления и остатка от деления

```
#include <stdio.h>
int main() {
    int x = 11, y = 4;
    float z = 4;
    printf(" %d  %f\n", x/y, x/z);
    printf("Остаток: %d\n", x%y);
}
```

2 2.750000

Остаток: 3

Операции отрицания (-, !).

Операции отношения (<, <=, >, >=, ==, !=)

Логические операции (&& и ||).

Тернарная операция:

```
i = (i < n) ? i + 1:  
1
```

Простое и сложное
присваивание:

```
a=c+5;a += b;d-=a;
```

Примеры выражений:

```
(a + 0.12) / 6  
x && y || !z  
(t * sin(x) - 1.05e4) / ((2 * k + 2) * (2 * k + 3))
```

Приоритеты:

$a = b = c$ означает $a = (b = c)$

$a + b + c$ означает $(a + b) + c$

$(\sin(x + 2) + \cos(y + 1))$

Преобразования типов

- изменяющие внутреннее представление величин (с потерей точности или без потери точности);
- изменяющие только интерпретацию внутреннего представления.

Явные преобразования типа:

- **static_cast<тип>(выражение)**
reinterpret_cast<тип>(выражение)
dynamic_cast<тип>(выражение)
const_cast<тип>(выражение)
- приведение в стиле C: **(имя_типа)выражение** или **тип (выражение)**

Пример:

```
int a = 2;
```

```
float b = 6.8;
```

```
printf( "%lf %d", double ( a ), ( int ) b );
```

Правила преобразования типов

Операнды `char`, `unsigned char` или `short` преобразуются к `int` по правилам:

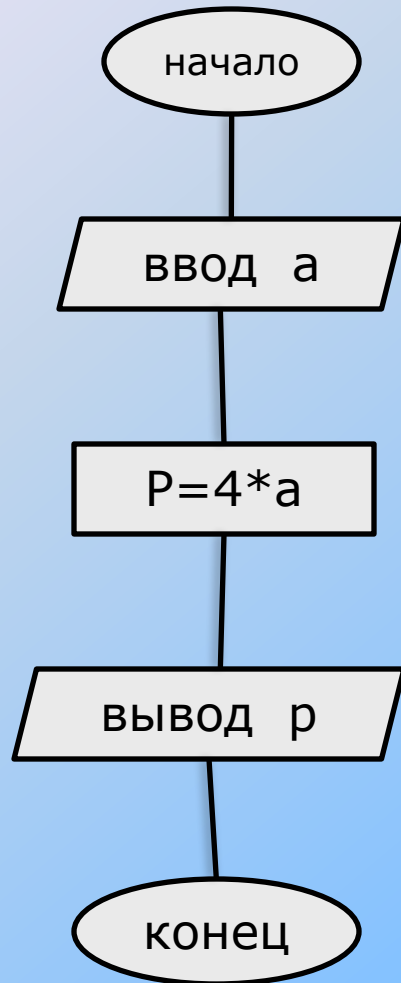
- *`char` расширяется нулем или знаком в зависимости от умолчания для `char`;*
- *`unsigned char` расширяется нулем; `signed char` расширяется знаком;*
- *`short`, `unsigned short` и `enum` при преобразовании не изменяются.*

Когда операнды становятся `int`, `float`, `double` или `long double`, то:

- *Если один из операндов имеет тип `long double`, то другой преобразуется к типу `long double`.*
- *Если один из операндов `double`, другой преобразуется к `double`.*
- *Если один из операндов `float`, другой преобразуется к `float`.*
- *Иначе, если один из операндов `unsigned long`, другой преобразуется к `unsigned long`.*
- *Иначе, если один из операндов `long`, то другой преобразуется к `long`.*
- *Иначе, если один из операндов `unsigned`, другой преобразуется к `unsigned`.*
- *Иначе оба операнда должны иметь тип `int`.*

Тип результата тот же, что и тип участвующих в выражении операндов.

Простейшая программа в среде Visual Studio



$p=4*a$

```
#include "stdafx.h"
#include "conio.h"
int _tmain(int argc, _TCHAR* argv[])
{
    float a,p;
    printf("input a: ");
    scanf("%f",&a);
    p=4*a;
    printf("\n p=%8.2f ",p);
    _getch();
    return 0;
}
```