Основы C++

Краткое содержание

C++ использует статическую типизацию, а корректная работа с операциями, преобразованием типов и вводом-выводом требует заранее учитывать типы данных и синтаксис операторов.

Типы данных

Статическая типизация

C++ — язык со **статической типизацией**: переменная должна быть объявлена с указанием типа **до первого использования**. Тип позволяет компилятору заранее выделить необходимый объём памяти.

Объявлять переменные можно не только в начале программы, но и непосредственно перед местом использования:

```
int count, figure, zero;
double x, y;
```

Значение при объявлении задавать необязательно.

Тип	Назначение	Особенности
`void`	Пустой тип	Обычно используется для функций, которые не возвращают результат
`bool`	Логический тип	Значения `true` или `false`
`char`	Символ или маленькое целое число	Обычно занимает 1 байт
`int`	Целые числа	Основной целочисленный тип
`long`	Большие целые числа	Используется при необходимости расширенного диапазона
`float`	Вещественные числа	Одинарная точность
`double`	Вещественные числа	Двойная точность, обычно 8 байт

Логический тип

`bool` принимает значения `true` и `false`. При преобразовании к `int` они обычно представляются как `1` и `0`, но при проверке выражений действует другое правило:

> ***Ложь*** *— только нулевое значение.* ***Истина*** *— любое значение, отличное от нуля.*

Например, `-3`, `0.5` и `7` считаются истинными.

`void` означает отсутствие возвращаемого значения. Функция типа `void` может выполнять вычисления, изменять данные или выводить информацию, но её результат нельзя использовать в дальнейших выражениях.

Целочисленные типы

`char` занимает 1 байт и может хранить числовое значение или символ:

```
char letter = 'A';
char number = 65;
```

Символ хранится как его числовой код в таблице ASCII. Для беззнакового варианта используется `unsigned char`; диапазон смещается в область неотрицательных значений.

`int` — основной тип для целых чисел. Его размер зависит от платформы, поэтому на разных системах диапазон может отличаться. Модификаторы `short`, `long` и `unsigned` изменяют размер или диапазон типа. На практике в большинстве учебных задач достаточно `int`.

Вещественные типы

Для дробных чисел используются `float` и `double`. `double` хранит значения точнее, но занимает больше памяти. В учебных задачах чаще применяются `float` или `double`.

Вещественные константы записываются через точку:

```
double a = 3.14;
```

Экспоненциальная запись:

```
1E-3 = 10⁻³
1E+3 = 10³
```

Строкового встроенного типа в C++ нет. Для строк используется специальный класс, а не фундаментальный тип данных.

Операторы и выражения

Присваивание и арифметика

Оператор `=` вычисляет выражение справа и записывает результат в переменную слева:

```
a = 1;
b = x + y;
```

`=` — присваивание, а `==` — сравнение. В блок-схеме присваивание обычно изображается прямоугольником, а условие — ромбом.

Основные арифметические операторы:

Оператор	**Операция**
`+`	Сложение

Сделано на сайте Зачёт | zachet.app

`-`	Вычитание
`*`	Умножение
`/`	Деление
`%`	Остаток от целочисленного деления

Например:

```
11 % 5  // 1
```

Остаток равен `1`, потому что из `11` можно вычесть `10`, кратное `5`. Это не дробная часть результата деления.

Инкремент и декремент

Инкремент увеличивает значение на единицу, **декремент** уменьшает его на единицу:

```
i++;
++i;
i--;
--i;
```

Позиция оператора важна при использовании внутри выражения:

```
a = i++;  // A
b = ++i;  // A
```

Сравнение и логика

Операторы сравнения:

```
<
>
<=
>=
==
!=
```

Знак `!` означает отрицание. Логические операции записываются так:

```
&&

||

!
```

Математическое двойное неравенство нужно разбивать на два сравнения. Условие $0 < b \leq 8$ записывается так:

```
b > 0 && b <= 8
```

Сокращённая запись

Операции вида `x = x + 2` можно записывать короче:

```
x += 2;
x -= 2;
x *= 2;
x /= 2;
```

Такая запись эквивалентна полной форме и не меняет смысл вычисления.

Приоритет

Вычисления выполняются слева направо с учётом приоритета операторов. Круглые скобки изменяют порядок вычислений:

```
result = (a + b) * c;
```

Стандартные математические функции подключаются из библиотеки `<cmath>`:

```
sqrt(x);  // :2>4@5=B0=K9
abs(x);   // <>4C;L
sin(x);   // A8=CA
cos(x);   // :>A8=CA
exp(x);   // M:A?>=5=B0
```

Другие функции приведены в таблицах стандартной библиотеки.

Приведение типов и точность

Смешивание целых и вещественных значений

Если в выражении участвует хотя бы один операнд вещественного типа, результат обычно становится вещественным:

```
int k = 10;
double y = 2.5;

k + 3       // F@K7C;LB0B
k + 3.0     // 2@5AB52;LB0K9
k * y       // 2@5AB52;LB0K9
```

Целочисленное деление

При делении двух целых чисел результат также целый. Дробная часть отбрасывается:

```
10 / 3   // 3
```

Чтобы получить вещественный результат, хотя бы один операнд должен быть вещественным:

```
10 / 3.0   // 3.333333...
```

Это одна из наиболее частых причин неожиданных результатов в программах.

Ошибки округления

Вещественные числа хранятся приближённо. Поэтому выражение вида

```
y / 3 * 3
```

может оказаться не строго равным `y`, даже если математически результат должен совпадать. После операций деления сравнение вещественных чисел на точное равенство нежелательно.

Вместо этого сравнивают модуль разности с небольшой погрешностью:

$$|a - b| < 10^{-6}$$

В C++ условие может выглядеть так:

```
abs(a - b) < 1E-6
```

Целочисленные выражения ведут себя иначе. Например, при `k = 10` выражение `k / 3` равно `3`, поэтому проверка результата должна учитывать целочисленное деление, а не математическое значение $10/3$.

Потоковый ввод и вывод

Подключение библиотеки

Для современного потокового ввода-вывода подключают библиотеку:

```
#include <iostream>
```

Основные объекты:

- `std::cout` — вывод в консоль;
- `std::cin` — ввод с клавиатуры.

Оператор `<<` передаёт данные в поток вывода, а `>>` извлекает данные из потока ввода:

```
std::cout << "25+48=";
std::cin >> x;
```

`std` — пространство имён стандартной библиотеки. Можно использовать сокращение:

```
using namespace std;
```

Потоковый вывод

Текст выводится в кавычках, числа и переменные — без кавычек:

```
std::cout << "57+30=" << result;
```

Несколько операций вывода подряд не создают пробелы или переносы автоматически:

```
std::cout << 1 << "A5=BO1@O
```

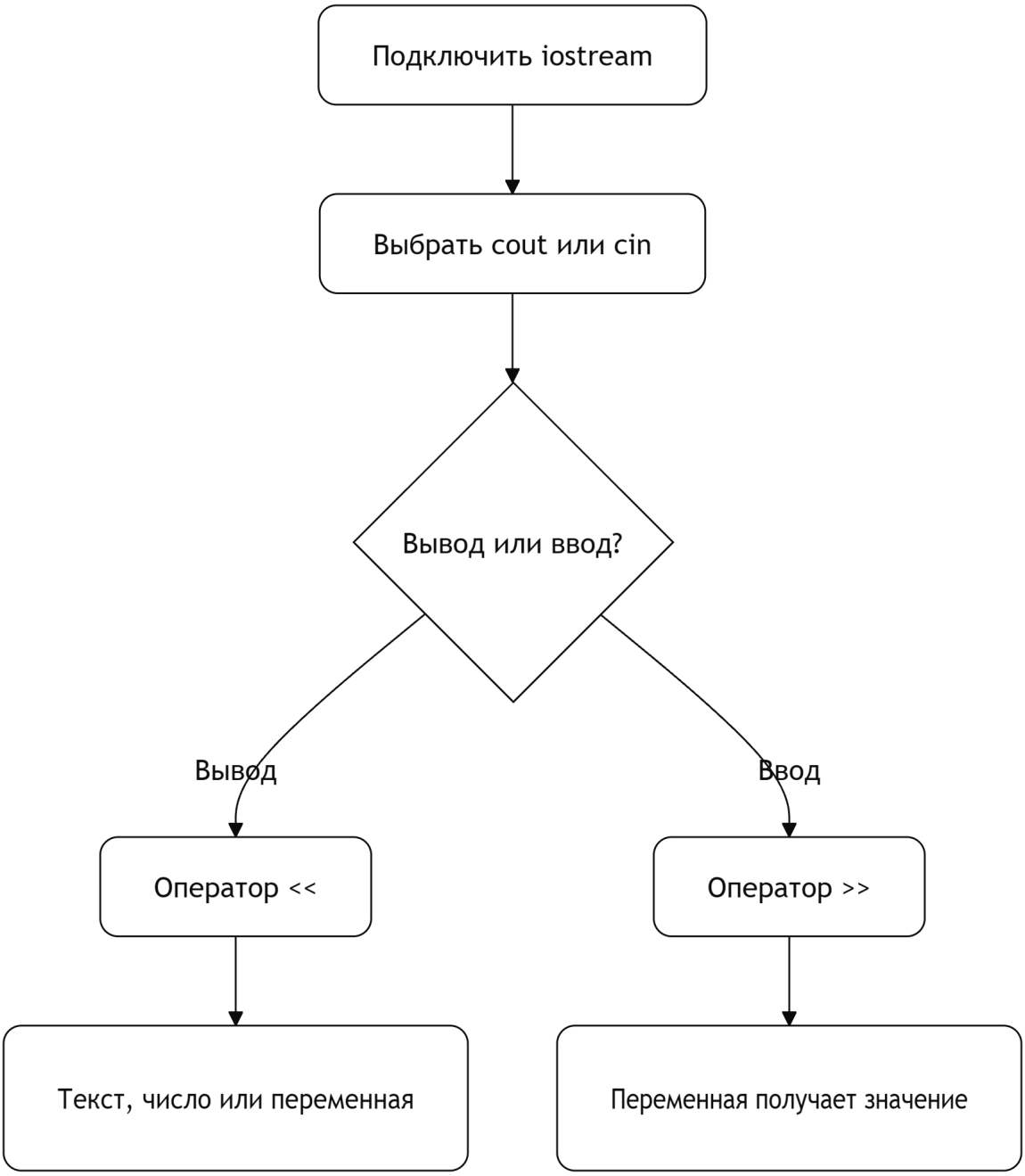
Результат будет `1AБ=BO1@O`. Пробел нужно добавить явно:

```
std::cout << 1 << " A5=BO1@O
```

Перенос строки можно выполнить двумя способами:

```
std::cout << "5@@@@std::endl;
std::cout << "A@@@O
```

`std::endl` переводит курсор на новую строку и очищает буфер вывода. `\n` только переводит строку и обычно используется чаще.



Потоковый ввод

Если значение задаётся пользователем, его не следует заранее присваивать в программе:

```
int x;
double y;
```

```
std::cin >> x >> y;
```

Так программа получает данные с клавиатуры и может работать с разными входными значениями.

Функции `printf` и `scanf`

C++ поддерживает не только потоки, но и функции ввода-вывода из языка C. Для них используется заголовочный файл:

```
#include <stdio.h>
```

Основные функции:

- `printf` — вывод данных;
- `scanf` — ввод данных.

Спецификаторы формата

Управляющая строка задаёт количество и типы аргументов.

Спецификатор	**Назначение**
`%s`	Строка символов
`%d`	Целое число типа `int`
`%f`	Вещественное значение

Пример ввода:

```
int x;
double y;

scanf("%d", &x);
scanf("%lf", &y);
```

Перед именем переменной в `scanf` обычно ставится `&`, поскольку функции нужен адрес памяти, куда будет записано значение.

Пример вывода:

```
printf("x = %d\n", x);
printf("y = %f\n", y);
```

Пояснительный текст и разделители нужно добавлять самостоятельно, иначе значения могут выводиться слитно:

```
printf("x = %d, y = %f\n", x, y);
```

Для `printf` спецификатор `%f` используется для вывода значения типа `double`, а для `scanf` значение типа `double` обычно считывается через `%lf`.

Ширина и точность вывода

В `printf` к спецификатору формата можно добавлять числовые параметры:

```
%HB@8=0ABL
```

- **Ширина поля** — общее количество позиций, отведённых под значение, включая цифры, точку и знаки после точки.
- **Точность** — количество знаков после десятичной точки.

Например:

```
printf("%10.4f", value);
```

означает: выделить минимум 10 позиций и вывести 4 знака после точки. Если само число короче поля, слева добавляются пробелы. Это позволяет выравнивать числа в столбцы.

Если ширина поля меньше фактической длины числа, число всё равно выводится полностью — оно не обрезается.

При точности `0` дробная часть не выводится, а число округляется до ближайшего целого:

```
printf("%.0f", 65.783);  // 66
```

Ширина и точность особенно полезны при создании аккуратных таблиц результатов.