Программирование

Лекция 2

Типы данных

- Целочисленные:
  1. `char` (символьный тип данных) Обычно 1 байт = 8 бит: $-2^7 \ldots 2^7-1$
  2. `short int`
  3. `int` Обычно 4 байта: $-2^{31} \ldots 2^{31}-1$ ($0 \ldots 2^{32}$ для unsigned)
  4. `long int`

  Могут быть беззнаковыми (`unsigned`).
  - $-2^{n-1} \ldots (2^{n-1}-1)$ (n — число бит)
  - $0 \ldots (2^n-1)$ для `unsigned`
- Числа с плавающей точкой:
  1. `float`, 4 байта, 7 значащих цифр.
  2. `double`, 8 байт, 15 значащих цифр.
- Логический тип данных `bool`.
- Пустой тип `void`.

Литералы

Значения встроенных типов в C++ задаются с помощью литералов.

- Целочисленные:
  1. `'a'` — код буквы 'a', тип `char`,
  2. `42` — все целые числа по умолчанию типа `int`,
  3. `1234567890L` — суффикс `'L'` соответствует типу `long`,
  4. `1703U` — суффикс `'U'` соответствует типу `unsigned int`,
  5. `2128506UL` — соответствует типу `unsigned long`.
- Числа с плавающей точкой:
  1. `3.14` — все числа с точкой по умолчанию типа `double`,
  2. `2.71F` — суффикс `'F'` соответствует типу `float`,
  3. `3.0E8` — соответствует $3.0 \cdot 10^8$.
- `true` и `false` — значения типа `bool`.
- Строки задаются в двойных кавычках: `"Text string"`.

Соответствует числу 97 в коде

Переменные

- При определении переменной указывается её тип. При определении можно сразу задать начальное значение (инициализация).

```
int      i = 10;
short    j = 20;
bool     b = false;

unsigned long l = 123123;

double x = 13.5, y = 3.1415;
float z;
```

- Нужно всегда инициализировать переменные.
- Нельзя определить переменную пустого типа `void`.

Операции

- Оператор присваивания: `=`.
- Арифметические:
  1. бинарные: `+ - * / %`,
  2. унарные: `++ --`.
- Логические:
  1. бинарные: `&& ||`,
  2. унарные: `!`.
- Сравнения: `== != > < >= <=`.
- Приведения типов: `(type)`.
- Сокращённые версии бинарных операторов: `+= -= *= /= %=`.

```
int i = 10;
i = (20 * 3) % 7;

int k = i++;
int l = --i;

bool b = !(k == l);

b = (a == 0) ||
    (1 / a < 1);

double d = 3.1415;
float f = (int)d;

// d = d * (i + k)
d *= i + k;
```

=4

k=4,i=5

l=4,i=4

=3

Операции инкремента и декремента

- int a = 10; // a = 10
- int b = ++a; // префиксный инкремент возвращает новое значение => b = 11 и a = 11
- int c = a++; // постфиксный инкремент возвращает старое значение => c = 11 и a = 12

Преобразование встроенных типов в операторах

- Выражениям так же как и значениям в C++ приписывается некоторые типы. Например, если a и b — это переменные типа int, то выражения (a + b), (a - b), (a * b) и (a / b) тоже будут иметь тип int.
- Важно всегда понимать, какой тип у выражения, которое вы написали в программе. Давайте проиллюстрируем это на следующем примере:
- int a = 20;
  int b = 50;
  double d = a / b; // d = 0, оба аргумента целочисленные, а значит деление целочисленное
- Как исправить этот код, чтобы получить вещественное значение в переменной d?
- Для этого хотя бы один из аргументов оператора деления должен иметь типа double. Этого можно добиться при помощи уже известного нам оператора приведения типов:
- double d = (double)a / b; // d = 0.4
- Почему это сработало? Дело в том, что операторы для встроенных типов C++ **всегда** работают с **одинаковыми типами аргументов**. Если аргументы имеют разные типы, то происходит преобразование типов (promotion).

Инструкции

- Выполнение состоит из последовательности *инструкций*.
- Инструкции выполняются одна за другой.
- Порядок вычислений внутри инструкций не определён.

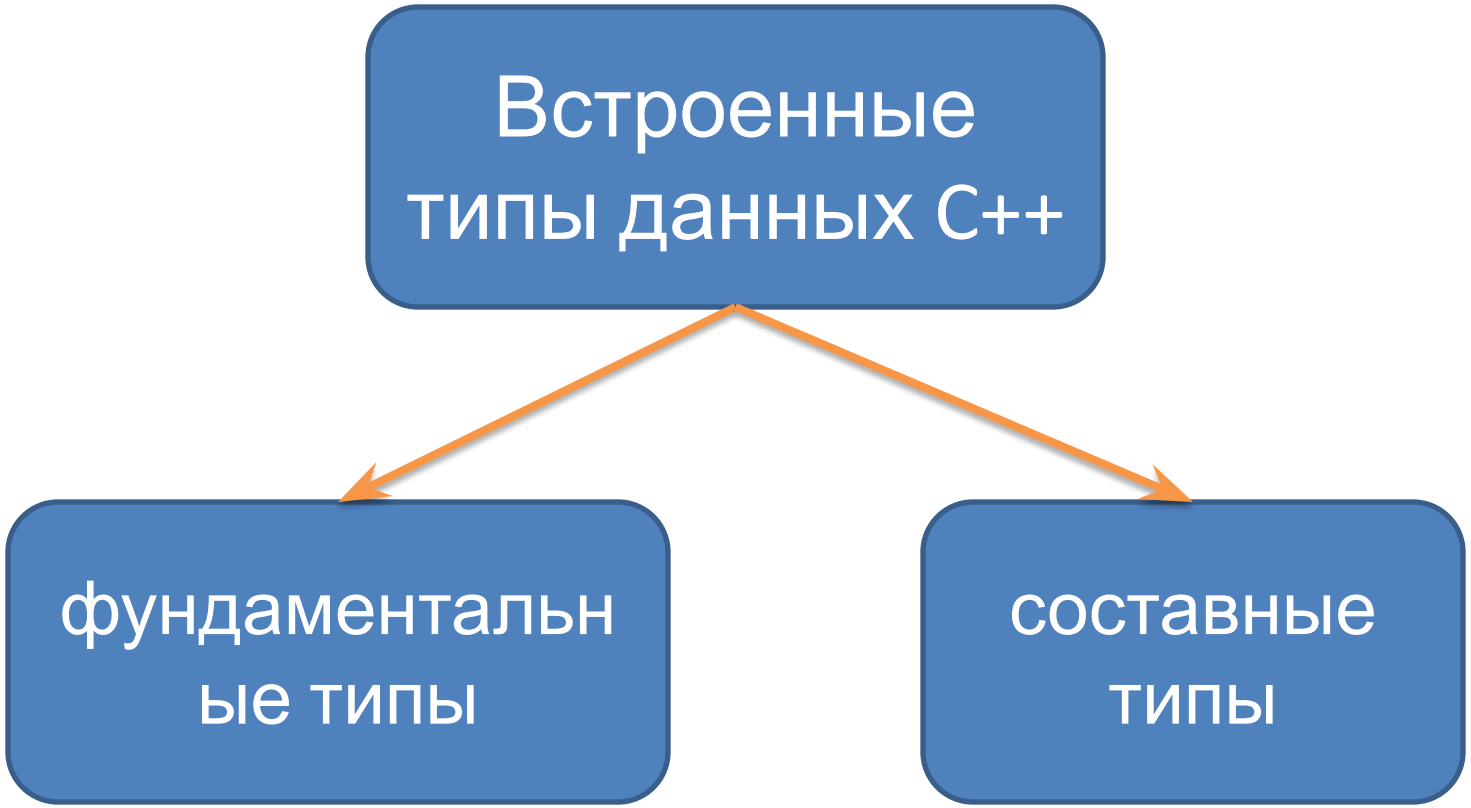
```
/* unspecified behavior */
int i = 10;
i = (i += 5) + (i * 4);
```

Значение выражения зависит от порядка вычислений

- Блоки имеют вложенную область видимости:

```
int k = 10;
{
    int k = 5 * i;   // не видна за пределами блока
    i = (k += 5) + 5;
}
k = k + 1;                  // k = 11
```

Типы данных


Встроенные типы данных C++
фундаментальные типы
составные типы

Переменные

- int cost_of_trip; // или costOfTrip

Правила именования:

- В именах разрешено использовать только алфавитных символов, цифр и символа подчеркивания (_).
- Первым символом имени не должна быть цифра.
- Символы в верхнем и нижнем регистре рассматриваются как разные.
- В качестве имени нельзя использовать ключевое слово C++.
- Имена, которые начинаются с двух символов подчеркивания или с одного подчеркивания и следующей за ним буквы в верхнем регистре, зарезервированы для использования реализациями C++, т.е. с ними имеют дело компиляторы и ресурсы. Имена, начинающиеся с одного символа подчеркивания, зарезервированы для применения в качестве глобальных идентификаторов в реализациях.
- На длину имени не накладывается никаких ограничений, и все символы в имени являются значащими. Однако некоторые платформы могут вводить свои ограничения на длину.

Переменные

```
int poodle;    // допустимое
int Poodle;    // допустимое и отличающееся от poodle
int POODLE;    // допустимое и отличающееся от двух предыдущих
Int terrier;   // недопустимое – должно использоваться int, а не Int
int my_stars3  // допустимое
int _Mystars3; // допустимое, но зарезервированное – начинается с
подчеркивания
int 4ever;     // недопустимое, потому что начинается с цифры
int double;    // недопустимое – double является ключевым словом C++
int begin;     // допустимое – begin является ключевым словом Pascal
int __fools;   // допустимое, но зарезервированное – начинается с двух
подчеркиваний
int the_very_best_variable_i_can_be_version_112;  // допустимое
int honky-tonk;        // недопустимое – дефисы не разрешены
```

Целочисленные типы

- Целыми являются числа без дробной части, например 2, 98, -5286 и 0.
- short, int, long и long long

- целочисленный тип `short` имеет ширину не менее 16 битов;
- целочисленный тип `int` как минимум такой же, как `short`;
- целочисленный тип `long` имеет ширину не менее 32 битов и как минимум такой же, как `int`;
- целочисленный тип `long long` имеет ширину не менее 64 битов и как минимум такой же, как `long`.

- sizeof - возвращает размер типа или переменной в байтах

Целочисленны

```
// limits.cpp -- некоторые ограничения целых чисел
#include <iostream>
#include <climits> // используйте заголовочный файл
int main()
{
    using namespace std;
    int n_int = INT_MAX;        // инициализация n_int
    short n_short = SHRT_MAX;   // символы, определенн
    long n_long = LONG_MAX;
    long long n_llong = LLONG_MAX;

    // Операция sizeof выдает размер типа или переменной
    cout << "int is " << sizeof (int) << " bytes." << endl;
    cout << "short is " << sizeof n_short << " bytes." << endl;
    cout << "long is " << sizeof n_long << " bytes." << endl << endl;
    cout << "long long is " << sizeof n_llong << " bytes." << endl << endl;
    cout << endl;

    cout << "Maximum values:" << endl;
    cout << "int: " << n_int << endl;
    cout << "short: " << n_short << endl;
    cout << "long: " << n_long << endl;
    cout << "long long: " << n_llong << endl << endl;

    cout << "Minimum int value = " << INT_MIN << endl;
    cout << "Bits per byte = " << CHAR_BIT << endl;
    return 0;
}
```

```
int is 4 bytes.
short is 2 bytes.
long is 4 bytes.
long long is 8 bytes.

Maximum values:
int: 2147483647
short: 32767
long: 2147483647
long long: 9223372036854775807

Minimum int value = -2147483648
Bits per byte = 8
```

Инициализация переменной

```
int n_int = INT_MAX;

int uncles = 5;                          // инициализация uncles значением 5
int aunts = uncles;                      // инициализация aunts значением 5
int chairs = aunts + uncles + 4;         // инициализация chairs значением 14


int owls = 101;  // традиционная инициализация в C
int wrens(432); // альтернативный синтаксис C++


int hamburgers = {24};                  // устанавливает hamburgers в 24

int emus{7};                            // устанавливает emus в 7
int rheas = {12};                       // устанавливает rheas в 12

int rocs = {};                          // устанавливает rocs в 0
int psychics{};                         // устанавливает psychics в 0
```

Типы без знаков

- short: -32 768 до 32 767
- беззнаковый вариант этого типа: 0 до 65 535.

```
unsigned short change;          // тип short без знака
unsigned int rovert;            // тип int без знака
unsigned quarterback;           // тоже тип int без знака
unsigned long gone;             // тип long без знака
unsigned long long lang_lang;   // тип long long без знака
```

Типы без знаков

```
Sam has 32767 dollars and Sue has 32767 dollars deposited.
Add $1 to each account.
Now Sam has -32768 dollars and Sue has 32768 dollars deposited.
Poor Sam!
Sam has 0 dollars and Sue has 0 dollars deposited.
Take $1 from each account.
Now Sam has -1 dollars and Sue has 65535 dollars deposited.
Lucky Sue!
```

```
// exceed.cpp –
#include <iostre
#define ZERO 0
#include <climit
int main()
{
    using namesp
    short sam = s
    unsigned sho
    cout << "Sam has " << sam << " dollars and Sue has " << sue;
    cout << " dollars deposited." << endl
         << "Add $1 to each account." << endl << "Now ";
    sam = sam + 1;
    sue = sue + 1;
    cout << "Sam has " << sam << " dollars and Sue has " << sue;
    cout << " dollars deposited.\nPoor Sam!" << endl;
    sam = ZERO;
    sue = ZERO;
    cout << "Sam has " << sam << " dollars and Sue has " << sue;
    cout << " dollars deposited." << endl;
    cout << "Take $1 from each account." << endl << "Now ";
    sam = sam - 1;
    sue = sue - 1;
    cout << "Sam has " << sam << " dollars and Sue has " << sue;
    cout << " dollars deposited." << endl << "Lucky Sue!" << endl;
    return 0;
}
```

Типичное поведение при переполнении

Целочисленные литералы

```
// hexoct1.cpp -- показывает шестнадцатеричные и восьмеричные литералы
#include <iostream>
int main()
{
    using namespace std;
    int chest = 42;         // десятичный целочисленный литерал
    int waist = 0x42;       // шестнадцатеричный целочисленный литерал
    int inseam = 042;       // восьмеричный целочисленный литерал

    cout << "Monsieur cuts a striking figure!\n";
    cout << "chest = " << chest << " (42 in decimal)\n";
    cout << "waist = " << waist << " (0x42 in hex)\n";
    cout << "inseam = " << inseam << " (042 in octal)\n";
    return 0;
}
```

```
Monsieur cuts a striking figure!
chest = 42 (42 in decimal)
waist = 66 (0x42 in hex)
inseam = 34 (042 in octal)
```

Целочисленные литералы

```
int main()
{
    using namespace std;
    int chest = 42;
    int waist = 42;
    int inseam = 42;

    cout << "Monsieur cuts a striking figure!" << endl;
    cout << "chest = " << chest << " (decimal for 42)" << endl;
    cout << hex;     // манипулятор для изменения основания системы счисления
    cout << "waist = " << waist << " (hexadecimal for 42)" << endl;
    cout << oct;     // манипулятор для изменения основания системы счисления
    cout << "inseam = " << inseam << " (octal for 42)" << endl;
    return 0;
}
```

```
Monsieur cuts a striking figure!
chest = 42 (decimal for 42)
waist = 2a (hexadecimal for 42)
inseam = 52 (octal for 42)
```

Тип char: символы и короткие целые числа

```
// chartype.cpp -- тип char
#include <iostream>
int main( )
{
    using namespace std;
    char ch;        // объявление переменной char
    cout << "Enter a character: " << endl;
    cin >> ch;
    cout << "Hola! ";
    cout << "Thank you for the " << ch << " character." << endl;
    return 0;
}
```

```
Enter a character:
M
Hola! Thank you for the M character.
```

Тип char: символы и короткие целые числа

```
// morechar.cpp -- сравнение типов char и int
#include <iostream>
int main()
{
    using namespace std;
    char ch = 'M';          // присваивает ch код ASCII символа M
    int i = ch;             // сохраняет этот же код в int
    cout << "The ASCII code for " << ch << " is " << i << endl;

    cout << "Add one to the character code:" << endl;
    ch = ch + 1;            // изменяет код символа в ch
    i = ch;                 // сохраняет код нового символа в i
    cout << "The ASCII code for " << ch << " is " << i << endl;

    // Использование функции-члена cout.put() для отображения символа
    cout << "Displaying char ch using cout.put(ch): ";
    cout.put(ch);

    // Использование cout.put() для отображения символьной константы
    cout.put('!');

    cout << endl << "Done" << endl;
    return 0;
}
```

```
The ASCII code for M is 77
Add one to the character code:
The ASCII code for N is 78
Displaying char ch using cout.put(ch): N!
Done
```

Литералы char

- 'A' соответствует 65, коду ASCII для символа A;
- 'a' соответствует 97, коду ASCII для символа a;
- '5' соответствует 53, коду ASCII для цифры 5;
- ' ' соответствует 32, коду ASCII для символа пробела;
- '!' соответствует 33, коду ASCII для символа восклицательного знака.

Коды управляющих последовательностей в C++

Название символа	Символ ASCII	Код C++
Новая строка	NL (LF)	\n
Горизонтальная табуляция	HT	\t
Вертикальная табуляция	VT	\v
Забой	BS	\b
Возврат каретки	CR	\r
Предупреждение	BEL	\a
Обратная косая черта	\	\\
Знак вопроса	?	\?
Одинарная кавычка	'	\'
Двойная кавычка	"	\"

```
char alarm = '\a';
cout << alarm << "Don't do that again!\a\n";
cout << "Ben \"Buggsie\" Hacker\nwas here!\n";
```

```
Ben "Buggsie" Hacker
was here!
```

Литералы char

```
cout << endl;            // использование манипулятора endl
cout << '\n';            // использование символьной константы
cout << "\n";            // использование строки
```

Следующие два оператора дают одинаковые результаты:

```
cout << endl << "What next?" << endl << "Enter a number:" << endl;
cout << "\n\nWhat next?\nEnter a number:\n";
```

```
// bondini.cpp -- использование управляющих последовательностей
#include <iostream>
int main()
{
    using namespace std;
    cout << "\aOperation \"HyperHype\" is now activated!\n";
    cout << "Enter your agent code:________\b\b\b\b\b\b\b\b";
    long code;
    cin >> code;
    cout << "\aYou entered " << code << "...\n";
    cout << "\aCode verified! Proceed with Plan Z3!\n";
    return 0;
}
```

```
Operation "HyperHype" is now activated!
Enter your agent code:________
Operation "HyperHype" is now activated!
Enter your agent code:42007007
You entered 42007007...
Code verified! Proceed with Plan Z3!
```

ТИП bool

```
bool is_ready = true;
```

True преобразовывается в 1, а false в 0:

```
int ans = true;          // ans присваивается 1
int promise = false;     // promise присваивается 0
```

Любое ненулевое значение преобразуется в true, а нулевое значение — в false:

```
bool start = -100;       // start присваивается true
bool stop = 0;           // stop присваивается false
```

Квалификатор const

Общая форма для создания константы:

```
const тип имя = значение;
```

Данный код не годится:

```
const int toes;          // значение toes в этот момент не определено
toes = 10;               // слишком поздно!
```

Совет

Если вы перешли на C++, имея до этого дело с языком C, и для определения символьных констант намерены пользоваться `#define`, лучше отдайте предпочтение `const`.

Числа с плавающей точкой

2.5, 3.14159 и 122442.32 — числа с дробными частями

34.1245 и 34124.5

0.341245 (базовое значение) и 100 (масштабный коэффициент)

0.341245 (такое же базовое значение) и 100 000 (больший масштабный коэффициент)

```
12.34           // число с плавающей точкой
939001.32       // число с плавающей точкой
0.00023         // число с плавающей точкой
8.0             // тоже число с плавающей точкой
```

Экспоненциальная запись:

```
2.52e+8         // можно использовать E или e; знак + необязателен
8.33E-4         // порядок может быть отрицательным
7E5             // то же, что и 7.0E+05
-18.32e13       // перед записью может стоять знак + или -
1.69e12         // внешний долг Бразилии в реалах
5.98E24         // масса Земли в килограммах
9.11e-31        // масса электрона в килограммах
```

Экспоненциальная запись

На заметку!

Форма `d.dddE+n` означает перемещение десятичной точки на `n` позиций вправо, а форма `d.dddE-n` — перемещение десятичной точки на `n` позиций влево.

Типы чисел с плавающей точкой

- float
- double
- long double

Высота горы Шаста в Калифорнии 14 179 футов
14 000 футов
14.179 тысяч футов

```
// floatnum.cpp -- типы с плавающей точкой
#include <iostream>
int main()
{
    using namespace std;
    cout.setf(ios_base::fixed, ios_base::floatfield);  /
    float tub = 10.0 / 3.0;                            /
    double mint = 10.0 / 3.0;                          /
    const float million = 1.0e6;
    cout << "tub = " << tub;
    cout << ", a million tubs = " << million * tub;
    cout << ",\nand ten million tubs = ";
    cout << 10 * million * tub << endl;
    cout << "mint = " << mint << " and a million mints = "
    cout << million * mint << endl;
    return 0;
}
```

```
tub = 3.333333, a million tubs = 3333333.250000,
and ten million tubs = 33333332.000000
mint = 3.333333 and a million mints = 3333333.333333
```

Арифметические операции в C++

```
int wheels = 4 + 2;
```

- Операция + выполняет сложение операндов. Например, 4 + 20 дает 24.
- Операция - вычитает второй операнд из первого. Например, 12-3 дает 9.
- Операция * умножает операнды. Например, 28*4 дает 112.
- Операция / выполняет деление первого операнда на второй.
- Например, 1000 / 5 дает 200. Если оба операнда являются целыми числами, то результат будет равен целой доли частного. Например, 17/3 дает 5, с отброшенной дробной частью.
- Операция % находит остаток от деления первого операнда на второй. Например, 19 % б равно 1. Оба операнда при этом должны быть целочисленными; использование операции % над числами в формате с плавающей точкой приведет к ошибке времени компиляции.

Примеры некоторых арифметических операций в C++

```
#include <iostream>
int main()
{
    using namespace std;
    float hats, heads;

    cout.setf(ios_base::fixed, ios_base::floatfield); // формат с фиксированной точкой
    cout << "Enter a number: ";
    cin >> hats;
    cout << "Enter another number: ";
    cin >> heads;
    cout << "hats = " << hats << "; heads = " << heads << endl;
    cout << "hats + heads = " << hats + heads << endl;
    cout << "hats - heads = " << hats - heads << endl;
    cout << "hats * heads = " << hats * heads << endl;
    cout << "hats / heads = " << hats / heads << endl;
    return 0;
}
```

```
Enter a number: 50.25
Enter another number: 11.17
hats = 50.250000; heads = 11.170000
hats + heads = 61.419998
hats - heads = 39.080002
hats * heads = 561.292480
hats / heads = 4.498657
```

Различные результаты, получаемые после деления

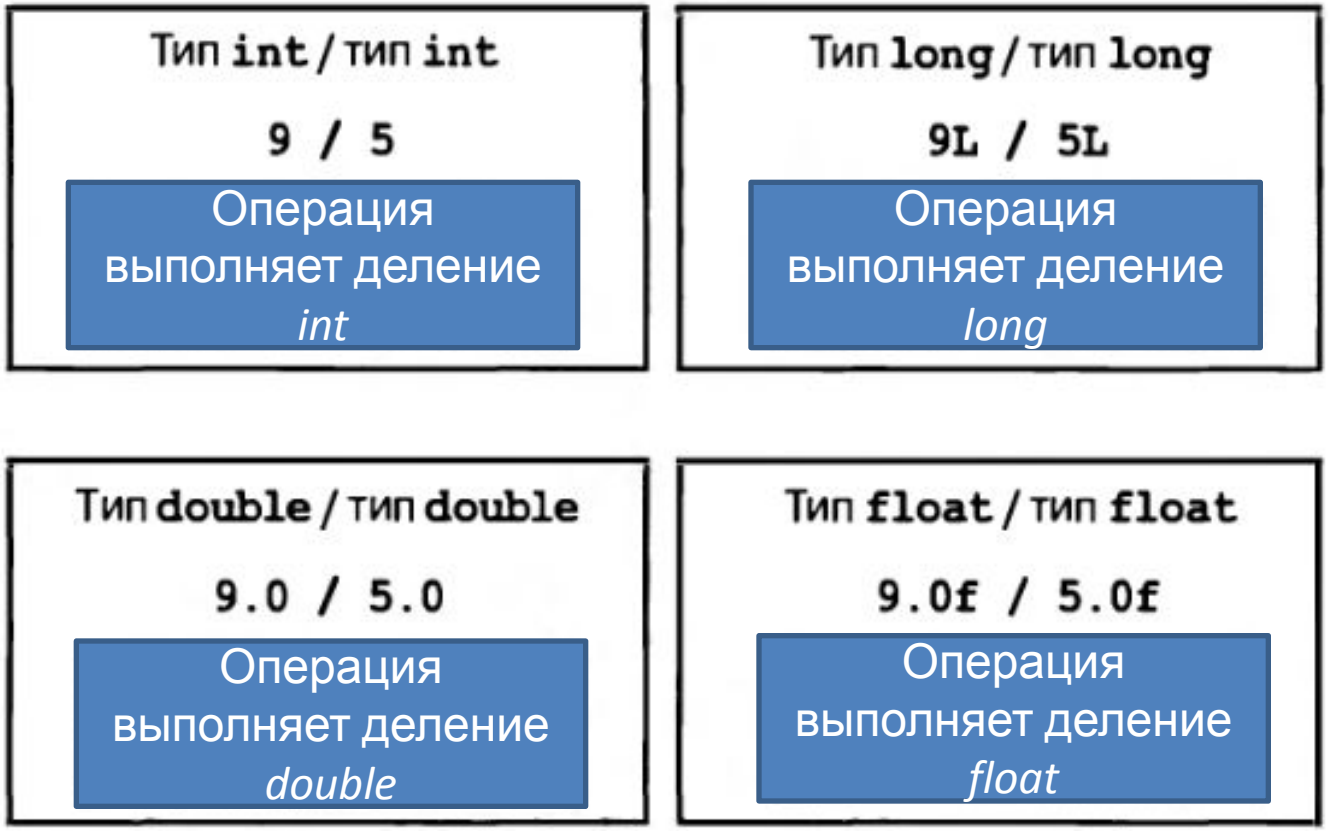
```
// divide.cpp -- деление целых чисел и чисел с плавающей точкой
#include <iostream>
int main()
{
    using namespace std;
    cout.setf(ios_base::fixed, ios_base::floatfield);
    cout << "Integer division: 9/5 = " << 9 / 5 << endl;
    cout << "Floating-point division: 9.0/5.0 = ";
    cout << 9.0 / 5.0 << endl;
    cout << "Mixed division: 9.0/5 = " << 9.0 / 5 << endl;
    cout << "double constants: 1e7/9.0 = ";
    cout << 1.e7 / 9.0 << endl;
    cout << "float constants: 1e7f/9.0f = ";
    cout << 1.e7f / 9.0f << endl;
    return 0;
}
```

```
Integer division: 9/5 = 1
Floating-point division: 9.0/5.0 = 1.800000
Mixed division: 9.0/5 = 1.800000
double constants: 1e7/9.0 = 1111111.111111
float constants: 1e7f/9.0f = 1111111.125000
```

Беглый взгляд на перегрузку операций

Использование одного и того же символа для обозначения нескольких операций называется **перегрузкой операций**.

Различные операции деления



Преобразования типов

- C++ преобразует значения во время присваивания значения одного арифметического типа переменной, относящейся к другому арифметическому типу.
- C++ преобразует значения при комбинировании разных типов в выражениях.
- C++ преобразует значения при передаче

```
so_long = thirty; // присваивание значения типа short переменной типа long
```

```
#include <iostream>
int main()
{
    using namespace std;
    cout.setf(ios_base::fixed, ios_base::floatfield);
    float tree = 3;             // int преобразован в float
    int guess = 3.9832;         // float преобразован в int
    int debt = 7.2E12;          // результат не определен в C++
    cout << "tree = " << tree << endl;
    cout << "guess = " << guess << endl;
    cout << "debt = " << debt << endl;
    return 0;
}
```

```
tree = 3.000000
guess = 3
debt = 1634811904
```

Резюме

- Целочисленными типами являются следующие: bool, char, signed char, unsigned char, short, unsigned short, int, unsigned int, long, unsigned long.
- Типов с плавающей точкой всего три: float, double и long double.

Вопросы для самоконтроля

- Почему в языке C++ имеется более одного целочисленного типа?
- Объявите переменные согласно перечисленным ниже описаниям.
- а. Целочисленная переменная short, имеющая значение 80. .
- б. Целочисленная переменная unsigned int, имеющая значение 42.110.
- в. Целочисленная переменная, имеющая значение 3 000 000 000.
- Вычислите следующие выражения:
- а. 8 * 9 + 2
- б. 6 * 3 / 4
- в. 3 / 4 * 6
- г. 6.0 * 3 / 4
- д. 15 % 4

Условные операторы

- Оператор `if`:

```
int d = b * b - 4 * a * c;
if ( d > 0 ) {
    roots = 2;
} else if ( d == 0 ){
    roots = 1;
} else {
    roots = 0;
}
```

- Тернарный условный оператор:

```
int roots = 0;
if (d >= 0)
    roots = (d > 0 ) ? 2 : 1;
```

Циклы

- Цикл `while`:

```
int squares = 0;
int k = 0;
while ( k < 10 ) {
    squares += k * k;
    k = k + 1;
}
```

- Цикл `for`:

```
for ( int k = 0; k < 10; k = k + 1 ) {
    squares += k * k;
}
```

- Для выхода из цикла используется оператор `break`.

Цикл do-while

- В C++ существует вариация цикла while, которая называется do-while. В отличие от обычного while в do-while условие проверяется не до, а *после* итерации. Т.е. такой цикл всегда имеет хотя бы одну итерацию.
- Давайте сравним обычный **while**:
- int i = 10;
- int sum = 0;
- while (i < 10)
- {
  sum += i;
- }
- // sum = 0
- И **do-while**:
- int i = 10;
- int sum = 0;
- do
- {
- sum += i;
- } while(i < 10);
- // sum = 10
- Как видите, в случае с do-while мы добавили 10 к переменной sum, а в обычном while — нет.

Управление циклами

```
int a = 323;
int b = 2;
while ( b <= a )
{
    if ( a % b == 0 )
     break; // выйти из цикла
    b = b + 1;
}
```

- После выполнения этого цикла мы найдём минимальное целочисленное b > 1 такое, что a делится на b, т.е. найдём наименьший простой делитель числа a. В данном случае b будет равен 17, т.к. 323 = 17 × 19.

Управление циклами

Ещё один оператор, который можно использовать с циклами — это оператор **continue.** Оператор **continue** прерывает текущую итерацию цикла и переходит к следующей. Например, можно посчитать сумму всех чисел от 1 до 100, которые не делятся на 17 или 19, следующим образом.

```
int sum = 0;
for ( int i = 1; i <= 100; ++i )
{
    if ( (i % 17 == 0) || (i % 19 == 0) )
      continue; // перейти к следующей итерации
    sum += i;
}
```

Ввод-вывод

- Будем использовать библиотеку <iostream>.

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

- Ввод:

```
int a = 0;
int b = 0;
cin >> a >> b;
```

- Вывод:

```
cout << "a + b = " << (a + b) << endl;
```

Функции

- В сигнатуре функции указывается тип возвращаемого значений и типы параметров.
- Ключевое слово `return` возвращает значение.

```
double square(double x) {
    return x * x;
}
```

- Переменные, определённые внутри функций, — *локальные*.
- Функция может возвращать `void`.
- Параметры передаются по значению (копируются).

```
void strange(double x, double y) {
    x = y;
}
```

Переменные не модифицируются

Задача

- Напишите функцию power, реализующую возведение целого числа в неотрицательную целую степень. Функция power должна принимать на вход два целых числа и возвращать целое число (смотрите шаблон кода). При выполнении задания учтите, что функция обязательно должна называться power, функция ничего не должна читать со входа или выводить.
- **Требования к реализации:** в этом задании вам нужно реализовать *только* функцию power. Вы можете определять вспомогательные функции, если они вам нужны. Реализовывать функции main не нужно.

- **Ограничения:** библиотеку cmath (и math.h) использовать запрещено.

```
// определите только функцию power, где
//    x - число, которое нужно возвести в степень
//    p - степень, в которую нужно возвести x
//

int power(int x, unsigned p) {
   int answer;
   /* считаем answer */
   return answer;
}
```

Решение

```
// определите только функцию power, где
//    x - число, которое нужно возвести в степень
//    p - степень, в которую нужно возвести x


int power (int x, unsigned p) {
   int answer = 1;
   /* считаем answer */
   for (int i=0; i<p; i++) {
      answer = answer * x;
   }

   return answer;
}
```