Любой язык содержит следующие базовые элементы: алфавит, лексемы, выражения, операторы. Лексемы образуются из символов, выражения — из лексем и символов, операторы — из символов, выражений и лексем.

Итак,

- *Алфавит языка* — это основные неделимые знаки, с помощью которых пишутся все тексты на языке (например, в русском языке — буквы).
- *Лексема* — минимальная единица языка, имеющая самостоятельный смысл (в русском языке — слово).
- *Выражение* — правило вычисления некоторого значения (в русском языке — словосочетание).
- *Оператор* — законченное описание некоторого действия (в русском языке — предложение).

1. Переменные, константы, типы данных

1.1. Тип данных

Тип данных — одно из основных понятий любого языка программирования. Позволяет компилятору определять характеристики переменных и констант, использующихся в программе.

То есть, тип данных определяет:

1. характеристику переменной или константы (целое число, символ и т. д.);
2. диапазон значений;
3. размер памяти, занимаемой этой переменной или константой (зависит от разрядности системы), определяется с помощью операции `sizeof()`;
4. операции, допустимые для данного типа.

В любом языке, в том числе C++, существуют встроенные типы и типы, определенные пользователем:

Встроенные

- *Целочисленный* — `int`. Существует четыре спецификатора: `signed` (знаковый), `unsigned` (беззнаковый), `short`, `long`. Спецификаторы `short` и `long` могут использоваться для описания короткого или длинного целого типов данных. При этом название типа `int` может быть опущено. По умолчанию все целочисленные типы являются знаковыми, поэтому спецификатор `signed` можно опускать. Итак, тип `int` определяет:

  1. Характеристика — целое число $(1, 2, -2$ и т. д.$)$;
  2. Диапазон значений зависит от размера памяти;
  3. Размер памяти зависит от спецификатора и разрядности системы:

Тип	Размер	Диапазон значений
`short`	2	$-32768..32767$
`unsigned short`	2	$0..65535$
`int`	2 или 4	$-32768..32767$ или $-2147483648..2147483647$
`unsigned int`	2 или 4	$0..65535$ или $0..4294967295$
`long`	4	$-2147483648..2147483647$
`unsigned long`	4	$0..4294967295$

  4. Определены операции сравнения, арифметические: сложение $(+)$, вычитание $(-)$, умножение $(*)$, определение частного от деления $(/)$, определение остатка от деления $(\%)$. Последние две операции: $a = x/y, b = x\%y$, означает, что $x = a * y + b$.

- *Вещественный* — существует два типа: `float`, `double` и один спецификатор `long`. Различаются количеством значащих цифр после запятой. Диапазон значений очень большой и роли не играет. Итак, вещественный тип определяет:

  1. Характеристика — вещественное число $(1.0, -0.25, 2$ и т. д.$)$;
  2. Количество значащих цифр зависит от размера памяти;
  3. Размер памяти зависит от спецификатора:

Тип	Размер (байт)	Количество значащих цифр
`float`	4	6–8
`double`	8	10–12
`long double`	10	12–15

  4. Определены операции сравнения, арифметические: сложение $(+)$, вычитание $(-)$, умножение $(*)$, деление $(/)$.

- *Символьный* — существует два типа: `char` для символов, определяемых согласно таблице ASCII-кодов, и `wchar_t`, для символов, определяемых в другой кодировке. Итак, символьный тип определяет:
  1. Характеристика — символ (`'a'`, `'1'`, `'+'` и т. д.);
  2. Диапазон: для типа `char` — 256 символов; для типа `wchar_t` — определяется реализацией, чаще всего достаточен для кодировки набора символов Unicode;
  3. Размер памяти зависит от реализации: для типа `char` — 2 байта; для типа `wchar_t` — 4–8 байт;
  4. Определены операции сравнения.
- *Логический* — `bool` используется в логических выражениях и как результат логических операций и операций сравнения. Итак, логический тип определяет:
  1. Характеристика — результат логических операций и операций сравнения (истина или ложь);
  2. Диапазон — два значения `true` (истина) и `false` (ложь);
  3. Размер памяти — 1 байт;
  4. Определены операции сравнения и логические операции.

Модифицированные К ним относятся массивы, строки, перечисления, структуры, классы и т. д.

1.2. Переменные

Место, в котором хранятся данные в компьютере, называется *объектом*. Для доступа к объекту необходимо знать его имя. Именованный объект называется *переменной*, имеющий конкретный тип. Данные, записанные в переменные, называют *значениями*.

Перед использованием любая переменная должна быть *определена*. При описании переменной определяется ее имя и тип. Пример описания:

```
int x, y; //описаны две переменные x и y типа int;
float z; //описана переменная z типа float.
```

При описании переменной желательно задавать начальные значения переменной (*инициализировать* переменные). Инициализация может быть произведена либо с помощью знака равенства `=` либо с помощью скобок `(...)`. Пример описания:

```
int x = 5, y = -2; //определены две переменные x и y типа int;
float z(2.54); //описана переменная z типа float.
```

Имя переменной может содержать латинские буквы, цифры и символ подчеркивания, причем начинаться может только с буквы (можно начинать с символа подчеркивания, но такая запись зарезервирована для встроенных констант, поэтому лучше ее не использовать). Прописные и строчные буквы различаются, поэтому `x` и `X` — это разные переменные. Имя переменной не может совпадать с зарезервированными словами.

Желательно, чтобы имя переменной определяло смысл переменной, например, `SqrCircle` — переменная, содержащая значение площади круга.

1.3. Константы

Константы — переменные, не изменяющие своего значения в течение работы программы. Константы бывают *именованные* и *неименованные*.

Именнованные константы определяются точно также, как переменные, только перед описанием типа ставят ключевое слово `const`:

```
const int x = 5; //константа типа int.
```

Именованные константы должны быть обязательно инициализированы. То есть, описание вида `const int x;` приведет к ошибке.

Неименованные константы разделяются на целочисленные, вещественные, символьные и строковые.

Неименованные *целочисленные константы* могут быть записаны в одной из трех форм:

- десятичная, например, `15`, `654`;
- восьмеричная — слева к записи числа добавляется `0`, например, `061`, `0251`;
- шестнадцатеричная — слева к записи числа добавляется `0x` или `0X`, например, `0x15`, `0X6A`.

Неименованные *вещественные константы* могут быть записаны в одном из двух видов:

- число с фиксированной точкой. Например, 0.000015, -14000.08. Дробная часть отделяется от целой точкой.

- число с плавающей точкой (в экспоненциальной форме). Например, `1.5E-5`, `-1.400008E+4`, `-8.7e+7`.

По умолчанию вещественные константы имеют тип `double`. Можно явно указать тип констант с помощью суффиксов `F`, `f` для типа `float` и `L`, `l` для типа `long double`. Например,
в выражениях `x = 5.3 * 10F, y = 8.4 - 123.5L` константы `5.3` и `8.4` будут иметь тип `double`, константа `10` — тип `float` и константа `123.5` — тип `long double`.

Неименнованные *символьные константы* представляют собой один или два символа (\ + некий символ), заключенных в апострофы. Например, `'a'`, `'-'`, `'W'`, `'\n'`, `'\t'`..

Символ обратной косой черты используется для представления:

- кодов, не имеющих графического изображения (например, `'\a'` — звуковой сигнал, `'\n'` — перевод курсора в начало следующей строки);
- символов апострофа (`'`) , обратной косой черты (`\`), знака вопроса (`?`) и кавычки (`"`);
- любого символа с помощью его шестнадцатеричного или восьмеричного кода, например, `\073`, `\0xF5`. Числовое значение должно находиться в диапазоне от 0 до 255.

Последовательности символов, начинающиеся с обратной косой черты, называют управляющими, или «escape»-последовательностями. В таблице 1 приведены их допустимые значения.

Строковые константы заключаются в кавычки. Внутри строковых констант могут использоваться управляющие последовательности. Примеры:

```
"Hello, my friend! \n"
```

2. Операторы ввода-вывода

Для консольного ввода-вывода в языке C++ существует две разновидности операторов:

- унаследованные от языка C операторы `scanf` и `printf`, для использования которых необходимо явно указывать тип вводимых или выводимых данных;

Таблица 1. Управляющие последовательности символов

Управляющая последовательность	**Значение**
\a	Звуковой сигнал
\b	Удаление предыдущего символа
\f	Перевод страницы
\n	Новая строка
\t	Горизонтальная табуляция
\v	Вертикальная табуляция
\r	Возврат каретки
\’	Апостроф (одиночная кавычка)
\”	Двойная кавычка
\\	Обратная косая черта
\?	Символ вопросительного знака
\0ddd	Восьмеричный код символа
\0xddd	Шестнадцатеричный код символа

- определенные в языке C++ конструкции `cout <<` и `cin >>`, которые самостоятельно определяют тип вводимых данных.

Что использовать, каждый решает сам. Вторые операторы позволяют не задумываться над типом вводимых данных, да и подходят к парадигме ООП. В то время, как первые быстрее выполняются.

2.1. Функции `scanf` и `printf`

Операторы `scanf` и `printf` позволяют вводить и выводить данные определенного формата.

Общий вид соответствующих функций:

```
int printf(const char *⟨управляющая строка⟩ ,...);
```

```
int scanf(const char *⟨управляющая строка⟩ ,...);
```

Управляющая строка состоит из символов:

1. Спецификаторы формата:

Код	Формат
`%c`	Символ
`%d`	Десятичное целое число со знаком
`%i`	Десятичное целое число со знаком
`%e`	Экспоненциальное представление числа (в виде мантиссы и порядка) (`e` на нижнем регистре)
`%E`	Экспоненциальное представление числа (в виде мантиссы и порядка) (`E` на верхнем регистре)
`%f`	Десятичное число с плавающей точкой
`%g`	Использует более короткий из форматов `%e` или `%f`
`%G`	Использует более короткий из форматов `%E` или `%F`
`%o`	Восьмеричное число без знака
`%s`	Символьная строка
`%u`	Десятичное целое число без знака
`%x`	Шестнадцатиричное без знака (строчные буквы)
`%X`	Шестнадцатиричное без знака (прописные буквы)
`%p`	Выводит указатель
`%n`	Соответствующий аргумент должен быть указателем на целое число.
`%%`	Выводит знак процента

2. Разделители (пробел, символ перехода на новую строку, символ табуляции, символ вертикальной табуляции).

3. Символы, не являющиеся разделителями.

Например, ввод сначала целого числа, потом десятичного числа с плавающей точкой, потом строки и вывод соответствующих переменных:

```
int x;
float y;
string str;
scanf("%d%f%s", &x, &y, str);
printf("%d %f %s", x, y, str);
```

Все данные, кроме строк, считываются с использованием ссылок `&` (подробнее будет рассмотрено ниже).

Так как пробел является разделителем, поэтому невозможно считать предложение, где пробел является таким же символом, как и все остальные.

2.2. Конструкции `cout <<` и `cin >>`

В данном случае используется понятие поток. *Поток* — нечто, связывающее источник с приемником.

Существует несколько типов потоков. Одним из них является стандартный поток — соединяющий клавиатуру и программу или программу и экран.

Для входного потока используется `cin >>`, для выходного — `cout <<`.

В отличие от `scanf` и `printf`, `cin` и `cout` являются объектами и способны сами определять тип вводимых и выводимых данных.

Например, ввод сначала целого числа, потом десятичного числа с плавающей точкой, потом строки и вывод соответствующих переменных:

```
int x;
float y;
string str;
cin >> x >> y >> str;
cout << "OUT:"<< x << ""<< y <<""<< str;
```

3. Базовые операторы

В любой программе необходимо выполнять вычисления. Для вычисления значений используются *вычисления*, которые состоят из операндов, знаков операций и скобок. Операнды задают данные для вычислений, операции задают действия. Скобки расставляют приоритеты. Каждый операнд является выражением или его частным случаем, например, константой или переменной.

3.1. Основные операции

Рассмотрим операции, определенные в языке C++ в порядке уменьшения их приоритета: Более приоритетными являются унарные операции (для которых требуется один операнд).

Инкремент и декремент Увеличение и уменьшение значения переменной на единицу.

Эти операции имеют вид: `++` и `--`

Имеют две формы записи — *префиксную* и *постфиксную*. В префиксной форме сначала изменяется операнд, а затем его значение становится результирующим значением выражения, а в постфиксной форме значением выражения становится исходное значение операнда, после чего он изменяется:

Например, постфиксная запись

```
int x = 5;
cout << x++;
```

означает, что сначала будет выведено на экран 5, а потом **x** будет увеличено на единицу. То есть, эквивалентно записи:

```
int x = 5;
cout << x;
x = x + 1;
```

Префиксная запись `int x = 5;`
`cout << ++x;`

означает, что сначала будет **x** будет увеличено на единицу, потом выведено на экран 6. То есть, эквивалентно записи:

```
int x = 5;
x = x + 1;
cout << x;
```

Операция определения размера sizeof предназначена для определения размера объекта или типа в байтах. Имеет две формы:

- `sizeof(выражение)` — количество байт, занимаемых результатом выражения;
- `sizeof (тип)` — количество байт, занимаемых переменной соответствующего типа.

Операции отрицания

- *Поразрядное отрицание* (~) — инвертирует каждый разряд в двоичном представлении целочисленного операнда.

  Например,

  ```
  short x = 5;
  cout << ~x;
  ```

Результат -6.

Представим число 5 в двоичном представлении типа `short`

`0000000000000101`

Инвертируем каждый разряд:

`1111111111111010`

Единица в старшем разряде означает отрицательное число. Это число представлено в дополнительном коде и равно -6.

- *Логическое отрицание* (`!`) дает в результате 0, если операнд истина, и 1, если операнд — ложь.
- *Арифметическое отрицание (унарный минус)* (`-`) меняет знак операнда на противоположный.

Операции работы с памятью *взятие адреса* (`&`), *разадресация* (`*`), *выделение памяти* (`new`), *освобождение памяти* (`delete`) будут рассмотрены позже.

Бинарные и тернарная операции (состоящие из двух или трех операндов). Даны в порядке уменьшения приоритета:

Арифметические операции: *умножение* (`*`), *деление* (`/`), *остаток от деления* (`%`) (определена только для целочисленных операндов), *сложение* (`+`), *вычитание* (`-`).

В случае целочисленных операндов операция деления определяет частное от деления, т. е., если представить целочисленное выражение x в виде $a * y + b$, то $a = x/y$, а $b = x\%y$.

Например, $5/2 = 2$, $5\%2 = 1$

Операции сдвига: *сдвиг влево* (`<<`), *сдвиг вправо* (`>>`) применяются к целочисленным операндам. Сдвигают двоичное представление первого операнда влево или вправо на количество разрядов, заданное вторым операндом. При сдвиге влево освободившиеся биты обнуляются, при сдвиге справа освободившиеся биты заполняются нулями, если первый операнд беззнакового типа и знаковым разрядом в противном случае. Операции сдвига не учитывают переполнение или потерю значимости.

Например:

```
short a = 16, b = 2;
cout << (a << b);
```

Результат : 64.

Представим 16 в двоичном представлении:

```
0000000000010000
```

Сдвигаем данные на 2 разряда влево, освободившиеся биты заполняем нулями:

```
0000000010000000
```

В десятичной форме это число 64.

Например:

```
short a = 16, b = 2;
cout << (a >> b);
```

Результат : 4.

Представим 16 в двоичном представлении:

```
0000000000010000
```

Сдвигаем данные на 2 разряда вправо, освободившиеся биты заполняем нулями:

```
0000000000000100
```

В десятичной форме это число 4. Таким образом, сдвиг влево (`a << b`) аналогичен выражению $a \times 2^b$, сдвиг вправо (`a >> b`) аналогичен выражению $\frac{a}{2^b}$.

Операции сравнения: *меньше* (<), *меньше или равно* (<=), *больше* (>), *больше или равно* (>=), *равно* (==), *не равно* (!=) сравнивают первый операнд со вторым. Результатом является `true` или `false` (любое значение, отличное от 0 интерпретируется как `true`). Операции сравнения на равенство и неравенство имеют меньший приоритет, чем остальные операции сравнения.

Поразрядные операции: применяются только к целочисленным операндам, работают с двоичными представлениями. Сравнение происходит побитово.

При *поразрядной конъюнкции* или *поразрядном И* (`&`) бит равен 1 тогда и только тогда, когда соответствующие биты операндов равны 1.

При *поразрядной дизъюнкции* или *поразрядном ИЛИ* (`|`) бит равен 1 тогда, когда хотя бы один из соответствующих битов равен 1.

При *поразрядном исключающем ИЛИ* (`^`) бит результата равен 1 тогда, когда соответствующий бит только одного из операндов равен 1.

Результатом будет соответствующее десятичное представление получившегося числа:

Например,

```
short x = 10, y = 5;
cout << x & y; \\ результат "--- 0
cout << x | y; \\ результат "--- 15
cout << x ^ y; \\ результат "--- 15
```

Представим 10 и 5 в двоичном коде:

```
00000000001010
&
00000000000101
=
00000000000000
```

Результат в десятичном коде — 0.

Представим 10 и 5 в двоичном коде:

```
00000000001010
|
00000000000101
=
00000000001111
```

Результат в десятичном коде — 15.

Представим 10 и 5 в двоичном коде:

```
00000000001010
&
00000000000101
=
00000000001111
```

Результат в десятичном коде — 15.

Логические операции: (`&&` и `||`). Операнды логических операций И (`&&`) и ИЛИ (`||`) могут быть любых типов. Преобразования типов не производится. Происходит срав-

нение каждого операнда с нулем (значение, равное нулю трактуется как `false`, ненулевые значение — как `true`). Результатом логической операции является `true` или `false`. Результатом логического И является `true`, тогда и только тогда, когда оба операнда имеют значение `true`. Логическое ИЛИ имеет значение `true` тогда, когда один из операндов имеет значение `true`.

Операции сравнения выполняются слева направо. Если значения первого из операндов достаточно для определения результата логической операции, то второй операнд не вычисляется.

Например, в следующем выражении:

```
x == 0 || y < 0
```

если x равно 0 и значение первого операнда — `true`, результат второго операнда не вычисляется.

Тернарная условная операция (`?:`) Эта операция выбора является тернарной, то есть, требует обязательно наличия трех операндов и имеет вид:

⟨*операнд1*⟩`?` ⟨*операнд2*⟩`:`⟨*операнд3*⟩.

Первый операнд — логическое выражение, имеющее результат `true` или `false`, второй операнд — выражение, выполняющееся, если результат первого операнда `true`, третий операнд — выражение, выполняющееся если результат первого операнда `false`. Например, в выражении:

```
X < Y ? Min = X : Min = Y;
```

находится минимум из двух чисел. Если `X < Y`, то минимум равен `X`, иначе — `Y`.

В случае, если необходимо поменять приоритет выполнения операций, используются круглые скобки.

3.2. Оператор присваивания

Оператор присваивания имеет следующий вид:

⟨*переменная*⟩=⟨*значение выражения*⟩.

Например, `a = x + 5;`

Последовательность выполнения: сначала вычисляется значение выражения, потом полученный результат записывается в область памяти, занятую переменной. Поэтому очень часто встречается записи вида:

```
a = a + 5;
```

Сначала вычисляется выражение $a + 5$, потом полученное значение присваивается переменной `a`, при этом предыдущее значение переменное уничтожается (об этом надо помнить).

Для удобства в языке C++ существуют составные операторы присваивания:

⟨*переменная*⟩+=⟨*значение выражения*⟩.

Например, `a += 5;` равносильно записи `a = a + 5;` Таким образом, можно сократить запись с арифметическими и поразрядными операциями.

3.3. Математические функции

В языке C стандартные математические функции содержались в заголовочном файле `"math.h"`. В C++ также можно использовать этот заголовочный файл. Кроме того в C++ можно подключать заголовочный файл `"cmath"`, в котором содержатся более разнообразные математические функции. В таблице 2 приведены некоторые математические функции из заголовочного файла `"cmath"`.

Примечание: все тригонометрические функции работают с радианами. Почти все функции в качестве аргументов используют вещественные числа.

Математические константы не определены в стандарте C++. Чтобы использовать их, необходимо сначала определить `_USE_MATH_DEFINES`, а затем включить `cmath` или `"math.h"`. Пример:

```
#define _USE_MATH_DEFINES
#include <cmath>
```

В месте использования константы указывается ее имя. В таблице 3 приведены некоторые математические константы.

Таблица 2. Математические функции

Функция	**Описание**
`abs(x)`	Абсолютная величина `x`. В заголовочном файле `math.h` необходимо использовать функция `fabs (x)` для вещественного `x`
`acos(x)`	Арккосинус `x`
`asin(x)`	Арксинус `x`
`atan(x)`	Арктангенс `x`
`ceil(x)`	Округление до ближайшего целого больше `x`
`cos(x)`	Косинус `x`
`exp(x)`	e^x
`floor(x)`	Округление до ближайшего целого меньше `x`
`log(x)`	Натуральный логарифм `x`
`pow(x,y)`	x^y, где `x` — вещественное, `y` — любое числовое
`sin(x)`	Синус `x`
`sqrt(x)`	Корень квадратный из `x`
`tan(x)`	Тангенс `x`

Таблица 3. Математические константы

Имя константы	**Выражение**	**Значение**
`M_E`	e	2.7182818284590452353 6
`M_LOG2E`	$\log_2 e$	1.4426950408889634073 6
`M_LOG10E`	$\log_{10} e$	0.43429448190325182765 1
`M_LN2`	$\ln(2)$	0.69314718055994530941 7
`M_LN10`	$\ln(10)$	2.30258509299404568402
`M_PI`	π	3.14159265358979323846
`M_PI_2`	$\pi/2$	1.57079632679489661923
`M_PI_4`	$\pi/4$	0.78539816339744830961 6
`M_1_PI`	$1/\pi$	0.31830988618379067153 8
`M_2_PI`	$2/\pi$	0.63661977236758134307 6
`M_2_SQRTPI`	$2/\sqrt{\pi}$	1.12837916709551257390
`M_SQRT2`	$\sqrt{2}$	1.41421356237309504880
`M_SQRT1_2`	$1/\sqrt{2}$	0.70710678118654752440 1

3.4. Преобразование типов

Довольно часто в выражениях присутствуют данные разных типов. Для корректного вычисления все данных должны быть приведены к одному типу. Преобразование бывает

двух видов: *явное*, когда программист сам приводит данные одного типа к нужному ему типу, и *неявное*, когда преобразование типов происходит автоматически.

Для явного преобразования нужно записать необходимый тип и в скобках переменную:

```
int x = 5;
cout << sqrt(float(x));
```

При явном преобразовании типа создается копия переменной нужного типа, существующая только во время выполнения оператора или функции, для которой и создавалась преобразованная переменная.

Неявное преобразование типов происходит в нескольких случаях:

Преобразование в операторе присваивания — тип выражения в правой части неявно преобразуется к типу переменной левой части. Если слева переменная меньшего типа (по количеству занимаемых байт в памяти), чем тип выражения справа, происходит потеря точности:

```
int x;
float a = 5.4, b = 7.5;
x = a + b;
cout << x;
```

В результате, сначала вычисляем значение выражения `a + b`. Результат — 12.9. Потом полученный результат надо присвоить переменной типа `int`. Результат — 12 (просто откидывается дробная часть).

Преобразование в выражениях — если в выражении содержатся переменные и константы разных типов, то все переменные приводятся к старшему типу.

Все переменные типа `char` и `short` всегда приводятся к типу `int`.

Так, например, достаточно простой способ получить из символа цифры (`char i = '5'`) целое число 5 :

```
int K = i - '0';
```

В таблице ASCII кодов цифры идут подряд, поэтому переменная `i` и символьная константа `'0'` преобразуются к типу `int` (порядковый номер символа в таблице ASCII кодов) и разность — значение переменной `i`.

Остальные типы приводятся последовательно от старшего к младшему:

1. Если один операнд имеет тип `long double`, второй приводится к типу `long double`;
2. Иначе если один операнд имеет тип `double`, второй приводится к типу `double` ;
3. Иначе если один операнд имеет тип `float`, второй приводится к типу `float`;
4. Иначе если один операнд имеет тип `unsigned long`, второй приводится к типу `unsigned long`;
5. Иначе если один операнд имеет тип `long`, второй приводится к типу `long`;
6. Иначе если один операнд имеет тип `unsigned int`, второй приводится к типу `unsigned int`;
7. Иначе все переменные имеют тип `int`.