Числовые типы:

Целочисленные типы:

- short. Переменная типа short занимает 2 Байта памяти, и принимает значения в диапазоне

  unsigned short: 0…65 535; 0000 0000 0000 0000 ..... 1111 1111 1111 1111

  $0…2^{16}–1;$

  signed short: –32 768 … 32 767;

  $–2^{15} … +2^{15} –1;$

- long – занимает 4 Байта памяти, и принимает значения в диапазоне

  unsigned long: 0 … 4 294 967 295;

  $0 … 2^{32}-1;$

  signed long: – 2 147 483 648 … 2 147 483 647;

  $– 2^{31} … 2^{31}–1;$

- int (Integer – Целое число). Платформенно зависимый тип данных, его величина зависит от процессора (CPU) операционной системы (ОС) и среды разработки (IDE – Integrated Development Environment). В Visual Studio для Microsoft Windows тип данных int заниамет 4 Байта, следовательно, его диапазоны принимаемых значений полностью совпадают с long.

- long long. Занимает 8 Байт памяти, и принимает значения в диапазоне

  unsigned long long int: $0 … 2^{64} – 1;$

  signed long long int: $–2^{63} … 2^{63} – 1;$

Вещественные типы:

Вещественные типы предназначены для хранения дробных чисел, (чисел с плавающей запятой).

Вещественные типы есть только знаковые, они не могут быть unsigned. В языке C++ есть всего два вещественных типа: **float** и **double**.

float – вещественный тип одинарной точности, занимает 4 Байта памяти.

double – вещественный тип двойной точности, занимает 8 Байт памяти.

float и double могут хранить ОЧЕНЬ БОЛЬШИЕ и ОЧЕНЬ МАЛЕНЬКИЕ числа, но эти числа могут быть не совсем точными.

Разделителем целой и дробной части у float и double является точка, а не запятая.

Объем занимаемой памяти переменной, константой или типом данных всегда можно определить оператором `sizeof()` следующим образом:

```
cout << sizeof(int) << endl;
```

или

```
cout << sizeof(1024) << endl; //это константа типа int, она занимает 4 Байта
```

Минимальное и максимальное значение для любого типа можно узнать при помощи макроопределений Visual Studio. Например, INT_MIN возвращает минимальное значение, которое можно записать в int, а INT_MAX – максимальное значение. У любой беззнаковой переменной минимальное значение всегда 0, а максимальное, например для int-а можно узнать при помощи UINT_MAX. U означает unsigned.

Д.З.: при помощи оператора sizeof и макроопределений вывести на экран объем занимаемой памяти для все числовых типов данных. Макроопределения можно найти в файлах "limits.h" и "float.h", эти фалы можно открыть любым текстовым редактором.

Имя переменной

Имя переменной нужно для того, чтобы к ней можно было обращаться по этому имени. К переменной обращаются для того, чтобы сохранить в ней какое-то значение, а потом использовать это значение. Когда мы сохраняем значение, мы обращаемся к переменной "на запись", а когда смотрим какое в ней значение, то обращаемся на чтение. В процессе компиляции имена переменных преобразуются в адреса памяти.

Для именования переменных используются идентификаторы (identifiers) составленные по определенным правилам. **Идентификатор(identifier) – это имя**.

Правила именования переменных

- Имя переменной (identifier) может состоять из символов латинского алфавита, строчных и ЗАГЛАВНЫХ, символов цифр 0123456789 и символа подчеркивания _; ABC....Zabc...z0123...9_
- Имя переменной (identifier) НЕ может начинаться символом цифры (~~`1stPlace`~~, `Place1`);
- Имена переменных *регистрозависимы*, то есть строчные и ЗАГЛАВНЫЕ символы различаются компилятором. Например `double Price;` и `double price;` это две разные переменные;
- Для именования переменных НЕЛЬЗЯ использовать ключевые слова языка C++ (`void`, `namespace`, `for`, `if`, `else`, `while` и т.д.);

Имя переменной должно быть осмысленным, то есть, по имени переменной должно становиться понятно, что в ней хранится!!! Например, переменная `double Weight;` содержит вес чего-либо.

Константы

Константа – это именованная область памяти, содержимое которой НЕ может изменяться в процессе выполнения программы. Для того, чтобы из переменной сделать константу, перед ее объявлением нужно написать ключевое слово `const`.

```
int speed = 0;              //Скорость (переменное значение)
const int MAX_SPEED = 250;  //Максимальная скорость (постоянное значение)
```

Константы принято называть заглавными буквами, для того чтобы после объявления было понятно, что это константа.

Кроме именованных констант существуют так же *символьные*, *строковые* и *числовые* константы.

Символьная константа – это один единственный символ, заключенный в одинарные кавычки (''), например `'+'` или `'A'`. Символьные константы – это константы типа `char`. Это легко проверить следующим образом:

```
cout << '+' << endl;
cout << sizeof(char) << endl;
cout << sizeof('+') << endl;
cout << typeid('+').name() << endl;
```

Строковая константа – это сколько угодно, каких угодно символов, заключенных в двойные кавычки "", например – `"Hello World"` или `"+"`. Строковые константы заниамют на 1 байт больше, чем содержат символов, это связано с тем, что компилятор неявно добавляет ASCII-символ с кодом 0 в конец строки. Это легко проверить следующим образом:

```
cout << "Строковые константы:\n";
cout << "Hello World" << endl;
cout << sizeof("Hello World") << endl;
cout << "+" << endl;
cout << sizeof("+") << endl;
```

Числовая константа – это просто число в исходном коде программы. Оно может быть целым, или дробным, например:

```
cout << 1024 << endl;       //Это числовая константа
cout << sizeof(1024) << endl;
cout << typeid(1024).name() << endl;
```

НО, у каждого значения (переменной, константы) в языке C++ есть тип. 1024 – это числовая константа типа `int`. Есть числовые константы и других типов:

```
cout << 3.14 << endl;       //Числовая константа типа double
cout << 5.   << endl;       //Числовая константа типа double
cout << 5.f << endl;        //Числовая константа типа float
cout << 123ll << endl;      //Числовая константа типа long long
cout << 123ull << endl;     //Числовая константа типа unsigned long long
```

Числовые, символьные и строковые константы еще называют *литералами*.

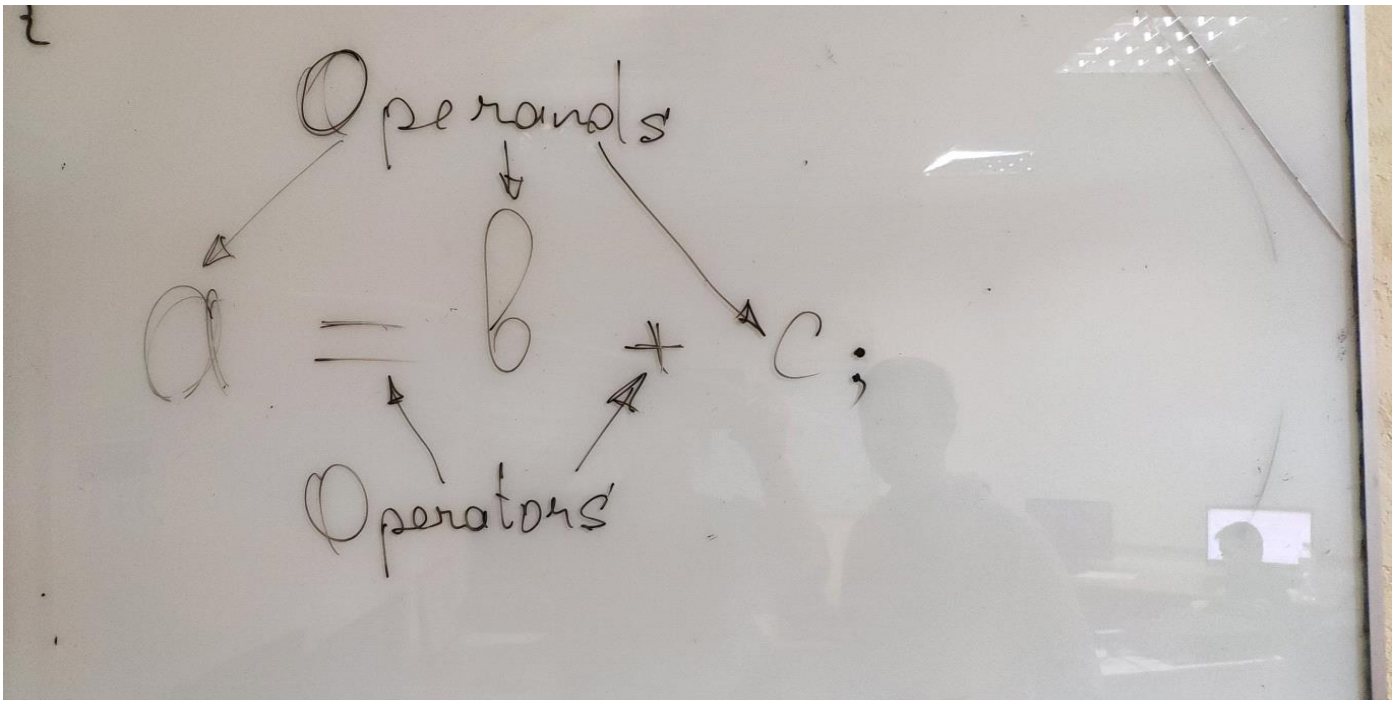
Операторы C++

Программа на языке C++ состоит из выражений, каждое из которых заканчивается символом `';'`.

Выражение (*Expression*) – это синтаксическая конструкция, состоящая из операндов и операторов.

Операнды – это объекты (элементы выражения), над которыми выполняется какое-то действие. В качестве операндов в выражениях обычно выступают переменные и константы.

Операторы – это объекты (элементы выражения), которые показывают, какое действие нужно выполнить над операндами. Операторы обозначаются одним или двумя специальными символами.



Операторы бывают: унарные, бинарные и тернарные. Унарные операторы выполняют действие над одним операндом, бинарные могут работать только с двумя операндами, а тернарные только с тремя операндами. Например `-3` здесь оператор `'-'` является унарным, он просто показывает что число 3 меньше нуля. В выражении 8 – 3 оператор минус – бинарный, он показывает из какого числа (8) вычесть другое число (3). 5*3 – выражение имеет смысл – одно число умножается на другое. *3 – выражение не имеет смысла, то есть, оператор * только бинарный – может работать только с двумя операндами (числами).

Все операторы языка C++ можно разделить на категории:

- *Арифметические операторы* (*Arithmetical operators*):

  Unary: + –;

  Binary: + – * / %;

  % — остаток от деления. Об этом операторе нужно знать две особенности:

  - Если делимое меньше делителя, то оно полностью выпадает в остаток

    ```
    int a = 25;
    ```

```
int b = 7;
cout << b % a << endl;
```

- Операция % — "Остаток от деления" НЕ применима к вещественным типам данных

```
int a = 25;
float b = 7;  //Ошибка на этапе компиляции
cout << b % a << endl;
```

- *Оператор присваивания* (*Assignment operator* =). Переменной слева, присваивает значение выражения справа. *Присвоить* значит *записать* (*сохранить*) в память. Переменную слева еще называют l-value, а выражение справа r-value.

l-value = r-value;

Например:

a = b + c;

здесь, в переменную ‘a’, которая слева от оператора = записывается (сохраняется) значение выражения ‘b+c’, которое находится справа от оператора “присвоить”.
В простейшем случае, выражение справа состоит из одной переменной или константы, например:

```
int a = 25;//Присвоить переменной 'a' значение 25. 25 - это числовая константа типа int.
int b = a;//Переменной 'b' присвоить значение переменной 'a'.
int c = (a + b) * 2; //Переменной 'c' присвоить значение выражения (a + b) * 2,
                     //или, в переменную 'c' записать (сохранить) значение выражения
                     //(a + b) * 2,что одно и то же,
                     //потому что ПРИСВОИТЬ - это ЗАПИСАТЬ!!!
```

- Increment/Decrement (++/--).

Increment – это унарный оператор, который увеличивает значение переменной на единицу. int i=2; i++; //после инкремента переменная i будет содержать значение 3

```
int i = 2;
cout << i << endl;
i++;   //Increment
cout << i << endl;   //После инкремента переменная i увеличилась на 1, то есть,
                     //теперь она содержит 3.
```

Decrement – это унарный оператор, который уменьшает значение переменной на 1.

```
int j = 5;
cout << j << endl;
j--;   //Decrement
cout << j << endl;
```

У инкремента и декремента есть две формы записи – *префиксная* и *постфиксная*. В префиксной форме записи оператор пишется перед операндом, а в постфиксной – после операнда:

```
int i = 0;
++i;   //Profix increment
i++;   //Postfix (Suffix) increment
--i;   //Prefix decrement
i--;   //Postfix decrement
```

Префиксная и постфиксная формы записи инкремента и декремента отличаются приоритетом по сравнению с другими операторами. У префиксной формы записи приоритет выше чем у других операторов, а у постфиксной – ниже, чем у других операторов.

a = b+c*d; //У оператора * самый высокий приоритет в этом выражении, он выполнится первым. У оператора = самый низкий приоритет в этом выражении, и он выполнится последним.
Можно сказать, что у префиксных инкремента и декремента САМЫЙ ВЫСОКИЙ ПРИОРИТЕТ, а у постфиксных САМЫЙ низкий ПРИОРИТЕТ, то есть они выполняются в последнюю очередь в любом выражении.

- *Составные присваивания* (*Compaund Assignments*). Используются, когда переменную нужно увеличить не на 1, а на другое значение, или в несколько раз. Сложные присваивания представляют собой комбинации, из арифметических операторов (+–*/%), и оператора присваивания (=). Например:

```
int i = 2;
i += 3;       //Увеличивает переменную 'i' на 3. //+= ПРИБАВИТЬ.
cout << i << endl;
i -= 2;       //Уменьшить переменную 'i' на 2.  //-= ОТНЯТЬ
cout << i << endl;
i *= 4;       //Увеличить переменную 'i' В 4 раза.
cout << i << endl;
i /= 3;       //Уменбшить переменную 'i' в 3 раза.
cout << i << endl;
```

- *Операторы сравнения* (*Comparison operators*). *Compare – Сравнить*.

Math	***C++***
=	==
≠	!=
>	>
<	<
≥	>=
≤	<=

Операторы сравнения предназначены для написания условий. *Условие* (*Condition*) – это сравнение. Все операторы сравнения возвращают `true` либо `false`, то есть значение типа `bool`. Если условие состоит из одной операции сравнения, то его называют *простым*. *Простые условия* можно объединять в *сложные*, при помощи *логических операторов*.

- *Логические операторы* (*Logical operators*).

! – NOT;
|| - OR;
&& – AND;
NOT (!) – это унарный оператор, который отрицает условие.
Например:

```
!true == false;//НЕ правда - это ложь;
cout << (!true == false) << endl;
```

OR – Результатом сложного условия будет true, если результат хотя бы одного простого условия – true. Логическое OR напоминает арифметическое сложение 1 и 0.
false || false || true = true;
0 + 0 + 1 = 1; //true
0 + 1 + 1 = 2; //true
0 + 0 + 0 = 0; //false

AND – Результатом сложного условия будет false, если результат хотя бы одного простого условия – false; Логическое AND напоминает арифметическое умножение 1и0.
1*1*1 = 1;//true
1*1*0 = 0;//false
0*1*1 = 0;//false

Управляющие структуры

Часто возникает необходимость сделать выбор того, какую часть программы нужно выполнить, или многократно выполнить определенную часть программы. Для этого в любом языке программирования есть управляющие структуры. Они делятся на *конструкции ветвления* и *циклы*.

Конструкции ветвления: `if...else...` и `switch`;

Циклы: `while...`, `do...while`, `for`;

Конструкцию ветвления if... else...

Выбирает один из двух вариантов кода, в зависимости от условия. У конструкции if следующий синтаксис:

```
if (Condition)
{
	...
	code1;
	...
}
else
{
	...
	code2;
	...
}
```

Condition – это *условие*. ***Условие* – это *сравнение***. Все операторы сравнения возвращают `true` либо `false`, то есть значение типа `bool`.

Если условие вернуло true, то выполняется code1, в противном случае, выполняется code2.

else и code2 являются не обязательными, то есть, if можно написать так:

```
if (Condition)
{
	...
	code;
	...
}
```

Если условие вернуло true, то code выполниться, если false, code будет проигнорирован.

Условие, состоящее из одной операции сравнения, называют *простым*. Несколько простых условий можно объединить в сложное, при помощи *логических операторов* (`&&` – `AND`, `||` – `OR`).

Конструкция множественного выбора switch

В отличие от if… else…, который позволяет выбрать **один** из **двух** вариантов кода, в зависимости от условия (Condition), switch позволяет выбрать **один** из **множества** вариантов кода, в зависимости от значения некоторой переменной. У конструкции `switch` следующий синтаксис:

```
switch (var)
{
case CONST_1: ...code1...; break;
case CONST_2: ...code2...; break;
.............................
.............................
case CONST_N: ...codeN...; break;
default: Default Code;

}
```

`var` – это переменная, по значению которой `switch` выбирает что нужно делать. Эту переменную (`var`), `switch` последовательно сравнивает с константами `CONST_1, CONST_2, ... CONST_N`, и если значения совпадают, то выполняется соответствующий код `code1, code2, ... codeN` до ключевого слова `break`. Ключевое слово `break` прерывает выполнение кода, и выходит за пределы конструкции `switch`. Если ключевое слово `break` отсутствует, то выполнится код, соответствующий следующему `case`, и так далее, пока не встретиться `break` или не закончится `switch`. Если значение переменной `var` не совпало ни с одной константой (`CONST_1, CONST_2, ..., CONST_N`), то выполнится код, после метки `default`, если она есть. Переменная `var` и константы `CONST_1, CONST_2, ..., CONST_N` могут быть только целочисленного (`short, long, int, long long`) либо символьного (`char`) типа.

Слово `case` означает "случай". `case` –ы также часто называют вхождениями, или метками (label).

Циклы

Цикл – это управляющая структура, которая позволяет многократно выполнить определенную часть кода (многократно повторить выполнение определенной части кода).

Цикл – это управляющая структура, которая позволяет зациклить выполнение определенной части кода.

Циклы бывают:

- с предусловием – while;
- с постусловием – do … while;
- цикл на заданное количество итераций – for.

Итерация – это однократное выполнение тела цикла.

Тело цикла – это код, который нужно зациклить.

`while (Condition)`
`{`
`group - of - statements; //Тело цикла.`
`}`
	`do`
	`{`
	`group - of - statements; //Тело цикла.`
	`} while (Condition);`

• Проверяется условие (Condition); • Если условие вернуло `true`, выполняется тело цикла; • Происходит возврат в начало, и снова проверяется условие, и т.д., пока условие не вернет `false`; • Если условие вернуло `false`, происходит выход за пределы цикла.	• Выполняется тело цикла; • Проверяется условие; • Если условие вернуло `true`, то происходит возврат в начало, и снова выполняется тело цикла, и т.д., до тех пор, пока условие не вернет `false`; • Если условие вернуло `false`, то происходит выход за пределы цикла.

Основным отличием между `while` и `do…while` является то, что `do…while` выполниться хотя бы один раз, не зависимо от условия.

`while` – сначала думает, потом делает, а `do…while` – сначала делает, а потом думает.