Ветвление

```
if (Условие) {
    // Блок кода, если истинно
} else {
    // Блок кода, если ложь
}
```

```
if (Условие) {
    // _блок кода при 1 условии_
} else if (2 условие){
    // _блок кода при выполнии 2 условия_
} else {
    // _блок кода_
}
```

Операторы условия

- `<` -меньше
- `>` - больше
- `<=` - меньше или равно
- `>=` - больше или равно
- `==` - равно
- `!=` - неравенство

Логические операции

- `!` - логическое НЕ
- `&&` - Логическое И
- `||` - Логическое ИЛИ

Управление памятью

`sizeof()` - возвращает размер типа данных в байтах

Указатели и ссылки

- *Указатель* - Переменная, которая хранит адрес какой-то ячейки памяти
  `*` или `[]` - передать указатель
- *Ссылка* - это альтернативный способ обращения к объекту в языке C++, чтобы не использовать. не всегда удобный указатель. Ссылка предоставляет альтернативное имя, для уже существующего объекта (псевдоним)

```
int x = 100;
int &a = x;
a++;
cout << x << endl;
```

Вариант с передачей ссылке в функции:

```
void inc(int& a) { // & - мы передаём ссылку
    a++;
}
int main(){
    int x = 4;
    inc(x);
    cout << x << endl;
}
```

Адресная арифметика

`cout << array;` - выведет адрес первого элемента
`cout << *array;` - разыменовывание
`cout << array+1;` - адрес следующей ячейки

Массивы

`int array[30]` - *статический массив* (`int marks[n]` - не получится)
`int array[num] = {}` - заполнить массив
`int* array = (int*) malloc(n * sizeof(int))` - динамический массив
`free(array)` - очистить память, где хранился динамический массив

Двумерные массивы

```
int arr[5][3] = {
    {4, 5, 3},
    {4, 5, 5},
    {4, 3, 3},
    {5, 5, 3},
    {1, 5, 2},
}
```

`matrix[строка][столбец]`\

Двумерный динамический массив

```
int n, m;
# Создание массива указателей
int **ptr = (int**) malloc(n * sizeof(int*));

for (int i = 0; i < n; i++) {
        ptr[i] = (int*) malloc (m * sizeof(int));
}
```

Циклы

- Цикл for

```
for (int i = 0; i < num; i++) {
    // Блок кода
}
```

- Цикл while

```
while (n > 0) {

}
```

Функции

Объявление функции - Объявление функции содержит информацию только о сигнатуре функции, без реализации этой функции

```
тип имя_функции(параметры) {
    код
    return возвращаемый параметр
}
```

Вызов функции:
`имя_функции(параметры)`

Функция - подпрограмма, которая может быть вызвана из других частей программы.
`void` - функция ничего не возвращает

Объявление функции

```
int* function_1(int*, int); // прототип функции
string function_2(string);
double function_3(float);
```

Символы и строки

`char` - это целочисленный тип данных для хранения одного символа (занимает в памяти 1 байт)
Объявление:
Значение записывается в одинарных кавычках
`char x = '!'`
Хранится значение таблицы. Значение записывается в ASCII

Преобразование типов

Явное приведение типов: `cout << (int)var << endl;`

```
char a = '2';
int x = (int)a - 48;
int x_2 = a - '0';
```

```
int a = 10, b = 10;
double x = a / (double)b;
// OR
double x = a / double(b);
```

`cin` - воспринимает до пробела
`cin.get(x)` - посимвольный ввод

Строки

`const` - объявление константой перменной

```
char ch = '!' // Символьный литерал
char str[] = "Hello, World"; // Строчный литерал
cout << str[0] << endl;
str[0] = 'W';
cout << str << endl;
```

С помощью `sizeof()` можно узнать длину строки `char str[]`, но +1 символ, чтобы знать где строка заканчивает. `strlen()` считает без нулевого символа.

```
#include <string>
...
string str;
string str2("Hello");
```

`string str(10, '!')` - Создание строки из одного и того же символа
`char *str = "Hello!"` - Раньше использовали вместо string
`\0` - нуль-символ, который находится в конце строки
Конкатенация:

```
    string str(10, '!');
    string str2(5, 'A');
    cout << str + str2 << endl;
```

`cout << str3.size() << endl;` - Размер строки
Увеличение и уменьшение строки

```
str3.resize(9);
cout << str3 << endl;
str3.resize(20, '#');
cout << str3 << endl;
```

Сравнение строк (Не зависит от длины строки, только по символам)

```
string str = "ABCDE";
string str2 = "BBCDE";

if (str > str2) {
    cout << "First" << endl;
} else {
    cout << "Second" << endl;
}
```

```
for (char ch: str) {
    cout << ch << ' ';
}
```

Индексирование

`front()` - первый символ
`back()` - последний символ
`push.back('a')` - добавить символ в конец строки
`append(str2)` - добавить строку в конец

Шаблонные функции

```
template <typename Type>
void arr1(Type var){
    cout << var << endl;
}
```

Работа с файлами

`#include <fstream>` - библиотека для работы с файлами

Открыть файл

Создаём объект файла "fout" для записи:\

```
int n = 1232434245;
fstream fout("output.txt", ios::out);
fout << "Hello world" << n << endl;
fout.close();
```

Создаём объект файла "file" для записи в конце:\

```
int n = 43;
fstream file("output.txt", ios::out | ios::app);
file << "Hello world" << n << endl;
file.close();
```

Создаём объект файла "fin" для чтения:
`fstream fin("example.txt", ios::in);`\

Второй способ открытия файла

Для чтения:
`ifstream file("example.txt");`
Для записи:
`ofstream file("example.txt");`

Повторно открыть файл

`file.open("example.txt", ios::app);`

`>>` - считывает до пробела
`getline()` - считывает строку
`fin.eof()` - курсор в конце (?)
`fin.get()` - получить символ\

`file.ignore(256, ':')` - игнорировать ":" или 256 символов
`file.find(';')`

Сдвиг курсора

`file.seekg(pos, start);`

- `pos` - насколько символов сдвинуть
- `start` - откуда начать пропускать

> - `std::ios::beg` - с начала файла
> - `std::ios::cur` - с текущей позиции
> - `std::ios::end` - с конца файла\

При достижения конца файла `seekg` перестаёт действовать. Необходимо выполнить `file.clear()`. После этого снова можем сдвинуть каретку в начало `file.seekg(0, std::ios::beg)`
`file.tellg()` - возвращает позицию каретки
Чтобы узнать размер файла:

```
file.seekg(0, std::ios::end);
int len = file.tellg();
```

Сложности алгоритмов

Большая О

Сложность алгоритмов: **временная**(время) и **пространственная**(память).
Время измеряется в количествах операциях.
Память - объём оперативной памяти, который потребуется алгоритму для обработки данных.
Основные типы временных сложностей:

- O(n)
- O(n^2)
- O(log n)
- O(n log n)
  Сайт для визуализации сортировок

Пузырьковая сортировка

Каждый элемент сравнивается со следующим. После каждого прохода мы ставим наибольший элемент в конец.
Для алгоритмов существуют три случая: худший, средний, лучший.
Сложность алгоритма всегда вычисляется для худшего варианта.
Оптимизация:

1. Каждый проход на один меньше
2. Если за проход не сделана ни одна замена, то элементы отсортированы, следовательно можно прекратить сортировку.

Сортировка выбором

Ищем наименьшее значение в массиве и ставим его на позицию с которой начали проход. Далее сдвигаемся и снова ищем наименьший элемент.

Сортировка вставками

Начинаем алгоритм со второго элемента и сравниваем его с предыдущем. Если надо, то меняем местами. Сравниваем следующую пару, если поменяли местами, то снова сравниваем предыдущую.

```
void insert_sort(vector<int>& array, int size, bool mode=true) {
        int i = 1;
        while (i < size) {
        if (((array[i-1] > array[i]) && mode) || ((array[i-1] < array[i])
&& !mode))
        {
                swap(array[i], array[i-1]);
                if (i != 1) i--;
        } else {
                i++;
        }
}
```

Vector - динамический массив

`#include <vector>` - подключение
`vector<int> array` - объявление пустого массива размером 0
`array.size()` - Получить размер массива
`array.push_back(var)` - добавить один элемент в конец массива
`array.pop_back()` - удалить последний элемент
`array.empty()` - проверка на пустоту
`array.resize()` - Переназначение размера\

```
void showVector(const vector<int>& array){
        for (int i = 0; i < array.size(); i++) {
```

```
                cout << array[i] << " ";
        }
        cout << endl;
}
```

Контейнер - структура, которая может хранить данные
Примеры: `map`, `list`, `vector`
var.method()
Когда прописываем явное значение - это **литерал**
Пример: `array.push_back(7.34)`

Рекурсия

Рекурсия - задание алгоритма вычисления функции с использованием вызова самой функции
Пример: факториал n!

- Итерационное определение: `произведение всех натуральных чисел от 1 до n`\
- Рекурсивное определение:

```
0! = 1
n! = n * (n-1)! # при n > 0
```

В рекурсивном определении всегда есть начальное значение. и сам вызов этого же определения, но с другими значениями - переход рекурсии
Пример 2: Числа Фибоначчи.

> Первые 2 числа равны единице, каждое следующее же равно сумме двух других предыдущих

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

int fib(int num) {
        if (num < 2) {
                return 1;
        } else {
                return fib(num - 1) + fib(num - 2);
        }
}

int main() {
        int a;
        cin >> a;
```

```
        cout << fib(a);
}
```

> Одни и те же значения функции вычисляются множества раз. Рекурсивный алгоритм нигде не сохраняет полученные значения и пересчитывает их каждый раз заново.\

Структура

```
struct Star {
        // Поля или свойства:
        string name;
        int size;
        int power;
};

int main() {
        Star x; // Объект или экземпляр
        Star y {"sun", 10, 3000};
        Star z;
        z.name = "sirius";
        z.size = 100;
        z.power = 10000;

        // Вывод свойств:
        cout << y.name << endl;
        cout << y.size << endl;
        cout << z.power << endl;

        return 0;
}
```

Значения по умолчанию:

```
struct Star {
        // Поля или свойства:
        string name;
        int size = 0;
        int power = 0;
};
```

Массив из структуры

```
Star stars[100];
stars[0].name = "sun";
```

Динамический массив

```
Star* stars = (Star*) malloc(n * sizeof(Star));
```