http://www.cplusplus.com/

#include <iostream>

#include <ctime>

#include <string>

#include<fstream>

#include "Header.h"

#include<bitset>

#include <typeinfo>

#define название значение - директива препроцессора(урок 66)

void main(int argc, char\* argv[]) - arg - количество ресурсов, argv - вывод их папок

using namespace std

setlocale(LC\_ALL, "ru") - русский

**#include<fstream>**

**Ofstream название переменной** – создание переменной класса для создания и заполнения файлов

**Переменная типа Ofstream.open(“название файла . формат”, ofstream::app)**  - создание файла , команда **ofstream::app** нужна, чтоб функция не стирала старые файлы, а дописывала новые.

**Переменная типа Ofstream.close()** – при окончании работы с файлом закрывает его

**Переменная типа Ofstream.open.is\_open()** – функция возвращает булевое значение понятия открыт файл или нет

**Переменная типа Ofstream << …**  - запись информации в файл

**Переменная типа Ofstream.write((char\*)& объект класса, размер объекта класса)** – запись объектов класса

**Ifstream название переменной** - создание переменной класса Ifstream для чтения файлов

**Переменная типа Ifstream.open(“название файла . формат”)** – открытие файла

**Переменная типа Ifstream.close()** – при окончании работы с файлом закрывает его

**Переменная типа Ifstream.is\_open() -** функция возвращает булевое значение понятия открыт файл или нет

**Переменная типа Ifstream.get( переменная для записи )** – побитовое копирование файлов в переменную + функция возвращает булевое значение есть ли там еще файлы

**Переменная типа Ifstream.eof()** – возвращает **true**,если в файле больше не осталось файлов

**Getline(Переменная типа Ifstream , переменная string)**  -запись строк из файла в переменную **string**

**>>**  - оператор записи файлов в переменную нужного типа

**Переменная типа Ifstream.write((char\*)& объект класса, размер объекта класса)** – вывод объекта класса, путем записи их в другой объект такого же класса

**fstream название переменной** – создание переменной класса для создания, заполнения и чтения файлов

**Переменная типа fstream.open(“название файла . формат”, fstream::app | fstream::in |**  **fstream::out )** – открытие файла переменной типа **fstream** + **fstream::app | fstream::in |**  **fstream::out**  - сохранение, запись и чтение файла

**Переменная типа fstream.close()** – при окончании работы с файлом закрывает его

**SetConsoleCP(1251)** – добавить возможность записи русских букв ( библиотека #include "Header.h" )

**SetConsoleCP(866)** – вернуть как было ( библиотека #include "Header.h" )

**Try {…} catch(const std::exception & ex) {…}** – выведение ошибки в случае не открытия файла

**fs.exceptions(ifstream::badbit | ifstream::failbit)**- функция для обработки ошибок

**ex.what()** – в **catch** функция отображающая ошибку

**ex.code()** – при работе с библиотекой **Ifstream** выдает код ошибки ( в **catch** нужно заменить на **ifstream::failure** )

**Throw…** - кинуть ошибку в catch

**//STD**

[**http://www.cplusplus.com/reference/**](http://www.cplusplus.com/reference/)

## Array

Обеспечивает мгновенный доступ по индексу.  
Фиксированный размер.  
Подходит для хранения известного кол-ва значений с возможным обновлением.

## Vector

Динамический массив.  
Мгновенный доступ по индексу.  
Быстро производит вставку и удаление элементов в конце вектора.  
Не подходит для вставки/удаления элементов из середины вектора.

## Stack

Быстрая вставка данных по принципу LIFO.  
Доступ возможен только к последнему элементу.  
Может использоваться для временного хранения данных с последовательным доступом.

## Queue

Быстрая вставка данных по принципу FIFO.  
Доступ возможен только к первому элементу.  
Может использоваться для временного хранения данных с последовательным доступом.

## Priority\_queue

Автоматическая сортировка элементов очереди - сверху всегда самый большой элемент.  
Доступ возможен только к самому большому элементу.  
Может использоваться для временного хранения отсортированных данных с последовательным доступом.

## Deque

Очередь с двусторонним доступом.  
Быстрая вставка и удаление данных из начала или конца контейнера.  
Имеет доступ по индексу.  
Не подходит для вставки/удаления элементов из середины очереди.

## List

Очередь с двусторонним доступом.  
Очень быстро производит вставку или удаление элементов для любого места в листе.  
Нет произвольного доступа. Возможны только итерации с начала или конца листа - доступ по индексу имеет линейную сложность O(N).

## Forward\_list

Очередь. Возможно итерировать только вперед.  
Доступ по индексу немного быстрей чем в List.

## Set

Бинарное дерево.  
Храняться только уникальные значения.  
Хранит эллементы с автоматической сортировкой по возрастанию.

## Multiset

Допускает одинаковые значения.

## Unordered\_set и Unordered\_multiset

Хеш-таблицы, порядок сортировки задается хеш-функцией.  
Мгновенный доступ по ключу.

## Map

Аналогичен контейнеру Set, хранит значение в виде std::pair.  
Ключем выступает pair.first, а значением pair.second.  
Поддерживает прямой доступ к значению по ключу - at() and [].  
Полезен, когда чисто цифровой ключ не подходит, а нужен к примеру, строковый или вообще другой объектный.

## Multimap

Аналогичен контейнеру Multiset, не поддерживает прямой доступ к значению по ключу.

## Unordered\_map and Unordered\_multimap

Аналогичны контейнерам Unordered\_set и Unordered\_multiset

**Типы операторов:**

**Begin**() - Return iterator to beginning (public member function )

**End() -** Return iterator to end (public member function ) ( указывает на элемент после последнего )

**Rbegin() -** Return reverse iterator to reverse beginning (public member function )

**Rend() -** Return reverse iterator to reverse end (public member function )

**cbegin() -** Return const\_iterator to beginning (public member function )

**cend() -** Return const\_iterator to end (public member function )

**crbegin() -** Return const\_reverse\_iterator to reverse beginning (public member function )

**crend() -** Return const\_reverse\_iterator to reverse end (public member function )

**advance( оператор, сдвиг )** – сдвиг операторов

**#include <typeinfo>**

**dynamic\_cast<в какой тип\*>(переменная)** – преобразование классов ( можно использовать в качестве if на проверку классов)

**#include <array>**

**Array< тип, количество > название** – создание контейнера

**Название массива[номер]** – работа с конкретным элементом

**Название.fill (…)** – заполнить весь массив одинаковыми элементами

**Array<…>::iterator название = значение** – создание начальных и конечных итераторов

**Auto** – сам подбирает нужный статический оператор

**Get<номер>(название переменной )** – возвращает значение переменной массива

**for\_each(начало цикла, конец цикла, функция для работы с операторами)** – цикл для массива

**#include <vector>**

**Vector<тип> название** – создание контейнера ( тот же **array** но в динамической памяти )

**Название.push\_back(…)** – заполнение последнего элемента массива

**Vector<…>::iterator название = значение** – создание начальных и конечных итераторов

**Название.empty()** – возвращает булевое значение пустой массив или нет

**Название.resize( новый размер, стандартное заполнение )** – пересоздание размера массива

**Название.reserve( на сколько )** – добавление ячеек в массив ( не удаляет имеющиеся как в **resize** )

**Название.shrink\_to\_fit()** – удаляет пустые ячейки

**Название.insert(Название.begin() + число сдвига , количество новых эл , их заполнение )** – создание элементов посередине массива

**Название.emplace(Название.begin() + число сдвига , заполнение )** -создание элементов посередине массива

**Название.max\_size()** – максимальное количество допустимых элементов в массиве

**Название.size()** – количество заполненных элементов массива

**Название.capacity()** – количество созданых ячеек

**#include<iterator>**

**for\_each(начало цикла, конец цикла, функция для работы с операторами)** – цикл для массива

**#include<stack>**

**Stack<тип> название** – создание контейнера

**Название.empty()** – возвращает булевое значение пустой стек или нет

**Название.pop()** – удаляет элемент стека

**Название.top()** – обращение к верхнему элементу стека

**Название.swap(Название)** – поменять местами значения стеков

**#include<queue>**

**Queue<тип> название** – создание контейнера

**Название.emplace(…)** – вставка элементов

**Название.push(…)** – вставка элементов

**Название.empty()** - возвращает булевое значение пустая очередь или нет

**Название.pop()** – удаляет последний элемент

**Название.front()** – возвращает первый элемент

**Название.back()** – возвращает последний элемент

**#include <concurrent\_priority\_queue.h>**

**Тоже самое что и queue за исключением того, что очередь сортируется от большего к меньшему и возврат первого элемента за функцией:**

**Название.top()** – возврат верхнего числа

**#include <deque>**

Совмещает **stack** и **queue**

**deque<тип> название** – создание контейнера ( тот же **array** но в динамической памяти )

**Название.push\_back(…)** – заполнение последнего элемента массива

**Название.push\_front(…)** – заполнение первого элемента массива

**deque<…>::iterator название = значение** – создание начальных и конечных итераторов

**Название.empty()** – возвращает булевое значение пустой массив или нет

**Название.resize( новый размер, стандартное заполнение )** – пересоздание размера массива

**Название.shrink\_to\_fit()** – удаляет пустые ячейки

**Название.insert(Название.begin() + число сдвига , количество новых эл , их заполнение )** – создание элементов посередине массива

**Название.emplace\_front(…)** – вставить элемент в начало

**Название.emplace\_back(…)** – вставить элемент в конец

**Название.max\_size()** – максимальное количество допустимых элементов в массиве

**Название.size()** – количество заполненных элементов массива

**#include <list>**

**list<тип> название** – создание контейнера

**auto название = Название.тип итератора()** – создание итератора

**Название.advance( итератор, на сколько сместить )** – смещение оператора для получения того или иного элемента

**Название.insert(оператор, количество, …)** – добавление элементов в лист

**Название.empty()** – возвращает булевое значение пустой массив или нет

**Название.push\_back(…)** – заполнение последнего элемента массива

**Название.push\_front(…)** – заполнение первого элемента массива

**Название.emplace\_front(…)** – вставить элемент в начало

**Название.emplace\_back(…)** – вставить элемент в конец

**Название.assign(сколько, …)** – добавление определенного количества элементов в конец

**Название.sort()** – сортирует элементы от меньшего к большему

**Название.reverse()** – реверсирует данные

**Название.unique()** – оставляет только неповторимые элементы

**#include <set>**

Массив с порядком

**set<тип> название** – создание контейнера с неповторимыми значениями

**multiset<тип> название** – создание контейнера с повторяемыми значениями

**Название.empty()** – возвращает булевое значение пустой массив или нет

**Название.insert(…)** – вставить элемент

**Название.emplace (…)** – вставить элемент

**Название.find(…)-** обращение к элементу под номером …

**Auto название = Название.find(…)** – обращение через указатель к элементу ( расшифровывать с \* )

**#include <unordered\_set>**

Создание того же **set** но без порядка

**Unordered\_set<тип> название** – создание контейнера с неповторимыми значениями

**Unordered \_multiset<тип> название** – создание массива с повторяемыми значениями

**Название.insert(…)** – вставить элемент без порядка

**Название.emplace (…)** – вставить элемент с порядком

**Название.bucket(…)** – возвращает номер по счету ковша с элементом …

**Название.bucket\_size(…)**- возвращает размер ковша элементов …

**Название.bucket\_count()** – возвращает количество ковшей

Остальное как и в **set**

**#include <map>**

Создает массив типа **pair** с сортировкой

**map<тип. тип> название** – создание контейнера

**multimap<тип. тип> название** – создание контейнер с повторяющимся значением

**название.insert( make\_pair(… , …))** - заполнение контейнера

**название.emplace( make\_pair(… , …) )** - заполнение контейнера

**auto** **название = название.find(…)** – создание указателя на элемент ( расшифровка с \* )

**название.at(…)** – возвращает ссылку на элемент по указанному номеру

**#include <unordered\_map>**

Все как и в **map** кроме:

**unordered\_map<тип. тип> название** – создание контейнера

**unordered\_multimap<тип. тип> название** – создание массива с повторяющимся значением

**Название.bucket(…)** – возвращает номер по счету ковша с элементом …

**Название.bucket\_size(…)**- возвращает размер ковша элементов …

**Название.bucket\_count()** – возвращает количество ковшей

**название.emplace\_hint(оператор, make\_pair(… , …) )** - заполнение массива

**#include <algorithm>**

**Copy( начало, конец, back\_inserter(функция) )** – копирование массивов

**Fill\_n(inserter(название контейнера , оператор) , количество, … )** – вставка элементов в контейнер

**Make\_move\_iterator( итератор другого контейнера )** – в новый контейнер того же типа можно вставить начальный и конечный итераторы такого же типа контейнеров и в результате получить копирование **v2(v1.begin(),v1.end())**

**#include <functional>**

**Function< возвращаемый тип ( принимаемый тип, принимаемый тип ) > название = [ ссылки на объекты ] ( принимаемые параметры ) { тело функции } ( сами принимаемые параметры ) ;** - создание лямбда функции

**[ ссылки на объекты ] ( принимаемые параметры ) { тело функции } ( сами принимаемые параметры ) ;** - создание лямбда функции без имени

**#include <regex>**

Разбор строки на компоненты и работа с ними

[**https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B3%D1%83%D0%BB%D1%8F%D1%80%D0%BD%D1%8B%D0%B5\_%D0%B2%D1%8B%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F**](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B3%D1%83%D0%BB%D1%8F%D1%80%D0%BD%D1%8B%D0%B5_%D0%B2%D1%8B%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F) **– стр. в вики**

**Cmatch название** – создает массив элементов в строке ( можно итерироваться по нему, нулевой элемент – сама строка )

**Regex название (…)** – условие выбора из строки

**Regex\_match ( название строки.c\_str() , переменная cmatch , переменная regex )** – возвращает булевое значение правильно построена ли строка

**Regex\_search ( название строки.c\_str() , переменная cmatch , переменная regex )** – находит выражение, заданное в условии, и возвращает булевое значение есть ли она

**Regex\_replace ( название строки.c\_str() , переменная cmatch , переменная regex или строка, которая будет заменять старое выражение )** – возвращает строку, может поменять ее составные, можно использовать с cout

regex regular("([\\w-]+)"

"(@)"

"([\\w-]+)"

"(\.)"

"([a-z]{2,5})"); - пример условия для email ( borshal85@gmail.com )

**#include <thread>**

Создание многопоточных программ

**Threat название( название функции, параметры функции )** – создание другого потока для функции

**thread::hardware\_concurrency()** – выводит количество ядер на пк

**название.joinable()** – возвращает булевое значение можно соединить потоки или нет ( соединять можно, если потоки работают одновременно и до этого к соединяемому потоку не была использована функция **detach()** )

**название.join()** – соединяет данный поток с тем, в котором была объявлена функция

**this\_thread::get\_id()** – возвращает ID процесса

**название**. **detach()** – отделяет поток от остальных навсегда

**#include <mutex>**

Дополнение к **#include <thread>** по управлению потоками

**mutex** – базовый mutex, которым может владеть один поток в единицу времени. При попытке повторного овладения мьютексом, потоком, уже владеющим им, произойдёт deadlock(или будет брошено исключение с кодом ошибки resource\_deadlock\_would\_occur).

**recursive\_mutex** – обладает теми же свойствами, что и std::mutex, но позволяет рекурсивное овладение мьютексом, то есть многократный вызов метода lock() в потоке, который владеет мьютексом. При этом, метод unlock() должен быть вызван не меньшее количество раз, чем был вызван lock(). В противном случае вы получите deadlock, т.к. этот поток никогда не освободит мьютекс и остальные потоки будут находиться в вечном ожидании.

**timed\_mutex** – обладая свойствами std::mutex, std::timed\_mutex, так же, обладает дополнительными методами позволяющими блокировку на время.

**recursive\_timed\_mutex** – рекуррентная версия std::timed\_mutex.

**Mutex название** – создание переменной типа **mutex**

**название**. **lock** – если мьютекс не принадлежит никакому потоку, тогда поток, вызвавший lock, становится его обладателем. Если же некий поток уже владеет мьютексом, то текущий поток(который пытается овладеть им) блокируется до тех пор, пока мьютекс не будет освобожден и у него не появится шанса овладеть им.

**название**. **try\_lock** - если мьютекс не принадлежит никакому потоку, тогда поток, вызвавший try\_lock, становится его обладателем и метод возвращает true. В противном случае возвращает false. try\_lock не блокирует текущий поток.

**название**. **unlock** – освобождает ранее захваченный мьютекс.

**название**.**try\_lock\_for** – расширенная версия try\_lock, которая позволяет задать продолжительность ожидания, прежде чем стоит прекратить попытку овладения мьютексом. Т.е. возвращает true в том случае, если удалось овладеть мьютексом в заданный промежуток времени. В противном случае возвращает false. Принимает std::chrono::duration, в качестве аргумента.

**название**.**try\_lock\_until** – та же, что предыдущая, но принимает std::chrono::time\_point в качестве аргумента.

# #include <condition\_variable>

**название**.**wait( элемент класса unique\_lock<mutex> … ( элемент класса mutex ) , булевая функция которая будет отвечать за работу функции wait )**  – ставит поток в ожидание сигнала. Ожидание не лимитировано временем. Может принимать в качестве аргумента предикат, от результата которого будет зависеть выход потока из ожидания. Т.е. если даже wait был завершен благодаря сигналу, происходит проверка предиката после чего поток снова становится в ожидание, если предикат ложен. На псевдокоде: (while(!predicate) wait;). А нужно это, в первую очередь, для того, чтобы избежать реагирования на фальшивое(spurious) пробуждение(см. врезку ниже).

**название**.**wait\_for** – Ожидание заданное время std::chrono

**название**.**wait\_until** - Ожидание до определенной даты

**название**.**notify\_one** – Посылает сигнал одному из ожидающих потоков; т.е. разблокирует один поток. Какой поток будет разбужен – не известно. Гарантировано лишь то, что один из них будет.

**название**.**notify\_all** – Посылает сигнал всем ожидающим потокам; т.е. разблокирует все потоки ожидающие на данном объекте condition\_variable

**#include <windows.h>**

**Sleep( время )** – останавливает компиляцию на указанное время в тысячных секунды

**#include <memory>**

**Shared\_ptr<тип> название( new тип( … ) )** – создание умного указателя

# Название класса<тип []> название ( new тип[ size ] ) – создание массива с помощью умных указателей

**int** — целочисленный тип данных.

**float** — тип данных с плавающей запятой.

**double** — тип данных с плавающей запятой двойной точности.

**char** — символьный тип данных.

**bool** — логический тип данных.

**string** - слова

добавление (\*) – ссылочный тип данных

добавление **static** – тип статик

**bitset< количество единиц и нулей > binary( переменная с которой считывается бит код )** – преобразование бинарного кода из переменной

**… -** обозначение в функции, что может быть добавлено еще неограниченное количество переменных

**Mutable** – делает константу меняемой

**break** - остановить цикл, случай

**const** - объявление неизменной переменой

**endl** - закончить строку

**continue** - продолжить цикл, пропустив задачи после команды continue

**return** значение - вернуть значение функции

**inline** - встраивает функцию в код( тупо вставляет функцию в код без поиска в памяти)

**delete** - удаляет в динамической памяти ненужный указатель

**nullptr** - удаляет адрес указателя

**тип** название.подназвание = значение - присваивание новым классам значений

**This. Или this->** - в классах указывает на сам объект

**Friend прототип функции** – создание дружественной функции, которая может принимать значения private и protected вне инициализации класса ( само разрешение инициализируется в теле класса)

**Public:** - общий класс

**Protected:** - класс для наследования

**Private:**  - закрытий класс

**Virtual** – создание виртуальных функций (их можно менять при наследовании

**Override** – проверка виртуальных функций на дееспособность

**Typedef** – создание сокращенного названия функции

**sizeof ()** - вес в байтах переменной (используют для нахождения количества элементов в массиве)

**rand()** - рандомное значение переменной

**srand()** - точка отсчета для рандома функции rand

**strlen(название массива)** - количество элементов в чаровом массиве

**strcat(массив 1 , массив 2)** - переписывает чаровый массив 2 в 1, объединение строк

**to\_string( что угодно )** - сделать тип string из чего угодно

**pair<тип 1,тип 2> название(параметр типа 1, параметр типа 2)** – стандартный класс с парами элементов ( обращение идет через first и second)

**typedef pair< тип 1,тип 2> новое имя** – создание скорченной записи функции с словом **typedef**

clock() – выводит время работы программы

**do {} while () {}** - выполняет действие, а потом зацикливается от while

**if () {} else {}**

**switch (условие) { case вариант ответа: ... default: … break;}**

**goto название линк; ... название линк:** - пропустить часть (...)

**arr[строки][колонки]{ {} , {} , {} };** - объявление двухмерного массива

**template <typename ... , typename ...>** - создание шаблонной переменной (она может принимать любые значения (строки 10 - 14) но у каждого имени только одно значение)

**\*pa = &a - \*название = &переменная** - указатель

**&a = b - &название = переменная** - ссылка

**\*переменная класса наследника = &переменная базового класса ( или a = &b )** – при наследовании можно присваивать указателям базового класса любую ссылку класса наследника

**int \*pa = new int** - тип переменной \*название = new то что хочешь создать - создание чего-то в динамической памяти

**int \*arr = new int[size]** - тип массива \*название = new тип[размер] - создание массива в динамической памяти

**int \*\*arr = new int\*[cols]** - создание динамического массива для указателей

**int \*&arr** - c этой фигней можно менять направление указателя (урок 59)

**char название массива[] = "что угодно"** \

- создание чарового массива

**char название массива[] = {'','','','\0'}**  /

(тип)название - изменение типа переменной

**char \*название[] = {"...","...","..."}** - двумерный чаровый массив

**тип переменой (\*название)(параметры)** - создание указателя на функцию в качестве параметра

**условие ? действие : условие ? действие : действие** - тот же If

# Struct название{ переменная : количество бит, переменная : количество бит …}; - создание структуры, в ней можно экономить оперативную память указывая на то, сколько юит должна хранить каждая переменная

# Union название { переменные } – объединяет все переменные в скобках, тобишь создает одну переменную и кучу ссылок на нее в разных типах, объявленные в скобках

# Enum название { имя = значение, имя = значение } - замена цифр названиями для удобства

# Namespace название {…} – создает пространство имен

# Bad\_alloc() – ошибка нехватки памяти

# Free(указатель на одну ячейку) – удаление ячейки в памяти

# class название { public: private: protected: ... } - создание своего класса

# название класса(переменные) {действие с переменными} - в классе паблик можно клешнями создать конструктор

# название класса(переменные) : название класса(переменные) {действие с переменными} – создание конструктора с наследованием

**Тип возвращаемого значения operator сам оператор () {}** - изменение значений операторов в классах

# Название.operator сам оператор = (название) – работа с операторами классов

# return \*this; - возвращение этой же переменной класса!!!!!!!!!!!

# Тип функции название класса :: название функции(параметры) {инициализация} – создание функции для класса вне класса ( в теле класса нужно объявить прототип функции)

# Friend тип функции название класса :: название функции(параметры функции) – создание дружественных функций для классов

# Int human :: number = 0; (тип переменной название класса :: название переменной = значение) – присваивание значений статистическим типам данных урок 92

# Название класса название массива[] { название класса ( ), название класса ( ) } – создание массивов класса

# Название класса имя класса : тип передачи класс который наследуешь - наследование класса

# Virtual тип функции название функции(параметры) = 0 - создание чисто виртуальной функции (классы которые имеют хотя б 1 чистую виртуальную функцию называются абстрактными)

# Virtual ~название деструктора() = 0 ( вне класса нужно создать пустой деструктор) - создание истого виртуального деструктора

# ::название класса::название функции(переменные) – вызов метода базового класса из класса наследника

# ((название класса)переменная класса наследника) . параметр или метод – указание конкретного параметра или метода в случае повторений названий при наследовании (урок 112)

# Class название : тип передачи virtual название наследуемого класса - виртуальное наследование (ромбовидное наследование), создано чтоб базовый класс создался 1 раз для всех тех, кто наследует его виртуально (сложно урок 114)

# template <typename … , typename …> и сам класс - создание шаблонных классов

# заголовок функции или конструктора : переменная( другая переменная ) {…} – копирование значения в объект касса другим способом

# название класса<пераметры>

# for(создать переменную : контейнер) {…} – создание цикла итерирующегося по контейнеру и записывающего его значения в переменную

# typeid( переменная ) . name() – возвращает тип переменной

# template<> class название<конкретный тип> {…}; - переопределение функций класса для отдельных типов

**array** - массив

**итерация** - повторение цикла

**рекурсия** - функция в функции, которые создают цикл

**стек** - ограничение рекурсии

**конкатенация** - объединение строк

**инкапсуляция** - принцип независимости данных (скрытый функционал, но оставлены внешние элементы для одержания результата)

**наследование** - один объект может наследовать функции другого объекта и дополнять их еще какими-то функциями

**полиморфия** - возможность объекта вести себя по разному в зависимости от ситуации и реагировать на действие специфическим образом характерным дя данного объекта(возможность получать один и тот же результат разными способами в зависимости от ситуации)

**свойство** - характеристика

**метод** - получаемый результат

Операторы есть **унарные, бинарные** и **тернарные**

**Конструктор**

**Деструктор**

**Терминирующий ноль ‘\0’**

**Агрегация –** возможность использовать класс встроенный в другой класс в других классах

**Композиция -** возможность использовать класс встроенный в другой класс только во внешнем классе

**Инстанцирование** - процесс создания объектов класса

**Ошибка сегментации** - ошибкаработы с памятью

**Предикаты** – булевые функции

**(++)** - инкремент

**(--)** - декремент

название массива - указатель на его первый элемент

256 типов элементов хранит в себе char

ascii - таблица элементов