Szablony		
template <class identifier> function_declaration;		template <class T> class Schow {T w; public: T get()};
template <*class* T> T klasa<T>::metoda ()	Def. Metody poza klasą	template <class T> T Show<T>::get(){ return w; }
template <> *class* Nazwa <*TypSpecjalizowany*>	Specjalizacja	template<> class Schow<int> {int w; public: int get(){return w
istream - wejście z pliku		
ifstream(const char*) ifstream file("nazwapliku.txt"); lub ifstream file;		
void open(const char*)	otwiera plik	file.open("nazwapliku.txt");
bool is_open()	sprawdza czy plik jest otwarty	if(!file.is_open()) return false;
void close()	zamyka plik	file.close();
>> | int get() | getline (istream&, string&)	czyta z pliku: dowolny typ, znak, całą linie	char c; file>>c; c=file.get(); getline(plik,str);
bool good() | bool eof()	czy dobry, koniec pliku	if(!file.good()) return false;
string - string str		
int length()	długość	
char& operator[] (size_t pos);	znak o danej pozycji	if(str[0]=='a')
const char* c_str() const;	konwertuje do char *	str.c_str()
size_t find (const char* s, size_t pos = 0) const;	Nie znalazł string::npos, pozyjca	if(str.find("dupa")!=string::npos) cout<<"znaleziono"
int compare (const char* s) const;	0 Takie same	str.compare("tekst")
list, map, vector <Typ zmiennej> cntir		
void push_back (const value_type& val);	dodaje na koniec kontenera	cntir.push_back(1);
iterator erase (iterator position);	usuwa z pozycji iteratora	vector<int>::iterator it= cntir.begin(); cntir.erase(it);
bool empty() const;	sprawdza czy konetenet pusty	cntir..empty()
size_type size() const;	zwraca ilość elementów w kont.	int ilosc=cntir.size();
iterator begin(); iterator end();	iterator na pierwszy i poostatni element	for(it=cntir.begin();it! =cntir.end();it++)
reference front(); reference back();	zwraca pierwszy/ostatni element	
list		
void sort(); template <class Compare> void sort (Compare comp);	sortuje listę, można podać funkcję sortującą	bool compare (Compare first, Compare second){}
void push_front(const value_type& val);	dodaj na początku	
vector		
reference operator[] (size_type n); reference at (size_type n);	dostęp do n+1 elementu	inta=cntir[0];
map <Typ1, Typ2> map<char,int> mapa; mapa["a"]=1;		
mapped_type& operator[] (const key_type& k);	zwraca wartość z określonej pozycji	cout<<mapa["a"];
iterator->first|second Pierwsza/druga wartość	dostęp do pierwszego drugiego el	map<char,int>::iterator it=mapa.begin(); it.first
cstring		
void * memcpy (void * destination, const void * source, size_t num);		
int strcmp (const char * str1, const char * str2);		
size_t strlen (const char * str);		

cctype		
int isalnum (int c);	czy liczba lub litera(wtedy 0)	
int isdigit (int c);		
int isalpha (int c);		
cmath		
sin,cos,tan,exp,pow(double,double)		
cstdlib		
atoi,strtod	char * do int, double	
abs	wartość bezwzględna	
cstdlib		
int rand (void);		srand(time(NULL)); rand()%max+min

Kolejność wywołań konstruktorów klasy bazowej, czy też obiektów składowych danej klasy, jest określona kolejnością:

Konstruktory klas bazowych w kolejności w jakiej znajdują się w sekcji dziedziczenia w deklaracji klasy pochodnej.

Konstruktory obiektów składowych klasy w kolejności, w jakiej obiekty te zostały zadeklarowane w ciele klasy.

Konstruktor klasy.