Facultatea de Inginerie Electrică

UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

EUROPEAN UNIVERSITY OF TECHNOLOGY

Facultatea de Inginerie Electrica

PCLP 2

Programarea calculatoarelor si limbaje de programare 2

PCLP2

An I semestrul II

"Coding is easy when you C it in action."

Cap. 9

Programare in mod proiect

9.1 Diferente C-C++

Limbajele C, C++

Limbajul C++ contine “in plus” fata de limbajul C: C

1) Cuvinte rezervate si comentarii (se pot introduce si cu // nu numai cu /* si */ si dupa alte instructiuni)
2) Conversii de tip (type casting)
3) Intrari/iesiri cu cin >> si cout <<
4) Declaratii variabile (oriunde in program)
5) Apel prin referinta
6) Functii care returneaza variabile
7) Functii inline
8) Parametri impliciti
9) Supraincarcarea functiilor
10) Alocarea dinamica a memoriei
11) Functii ca membri ai unei structuri
12) Operatorul de rezolutie
13) Tipul boolean
14) Clase

+

+

9.1 Diferente C-C++

1) Cuvinte cheie rezervate C++

nu pot fi utilizate ca nume de variabile, tipuri noi de date sau functii

Keywords common to the C and C++ programming languages

auto	break	case	char	const
continue	default	do	double	else
enum	extern	float	for	goto
if	int	long	register	return
short	signed	sizeof	static	struct
switch	typedef	union	unsigned	void
volatile	while			

C++ only keywords

asm	bool	catch	class	const_cast
delete	dynamic_cast	explicit	false	friend
inline	mutable	namespace	new	operator
private	protected	public	reinterpret_cast	
static_cast	template	this	throw	true
try	typeid	typename	using	virtual
wchar_t				

9.1 Diferente C-C++

2. Conversii de tip (type casting)

in C:

```
int n;
float x;
x=n; //genereaza warning
x = (float) n; //sintaxa C
```

in C++:

```
int n;
float x;
x=n; //genereaza warning
x = float(n); // sintaxa C++
```

Ce va afisa urmatorul program C++?

```
#include <iostream>
using namespace std;
int main()
{  cout<< char(65) <<"\n"; return 0;}
```

```
#include <iostream>
using namespace std;
int main()
{  cout<< (char)65 <<"\n"; return 0;}
```

Conversia de tip (char) determina compilatorul sa interpreteze 65 ca si character in loc de numar.
Se va afisa litera A din codul ASCII

9.1 Diferente C-C++

3) Definirea unei constante

in C:

```
#DEFINE dim 100
char sir[dim];
```

in C++:

```
const int dim=100;
char sir[dim];
```

EXEMPLE

in C++: calcul lungime (circumferinta) cerc

```
#include <iostream>
using namespace std;
const double pi = 3.14159;
const char newline = '\n';
int main ()
{ double r=5.0; // raza  cercului
  double lcerc;
  lcerc = 2 * pi * r;
  cout << lcerc;
  cout << newline;return 0;}
```

9.1 Diferente C-C++

3) cin , cout

```
// introducerea unui numar
int nr;
cout << "Numar>";
cin >> nr;
```

```
// introducerea unui sir
char s[100];
cout << "Sir> ";
cin >> s;
```

```
Sir> imi place C++
imi
```

Citeste primul sir pana la spatiu : imi
Pentru citire tot sirul utilizam:
cin.get(s,15) ;

```
// afisarea cu printf si cout in acelasi program
cout << “Totul”; printf( “despre”); cout << “ C/C++”;
```

9.1 Diferente C-C++

4. Declaratii variabile in C++

C++: oriunde in program

```
int x;
x = 10;
cout << x << endl; // afiseaza 10
{ double a, b = 2.0;
a = 3.14 * b;
cout << a << endl; // afiseaza 9.28
}
a *= 3;
cout << a << endl; // afiseaza 18.84
```

C++: in interiorul buclei for

```
for( int i =0; i<4; i++)
{  int a = i;
   a *= 10;
cout << a << endl;}
cout << i;
```

9.1 Diferente C-C++

4. Declaratii variabile in C++: alias

Variabile cu mai multe nume (alias-uri)

```
double x = 3.14;
double &y = x; // y face referire la x
y = 23.45; //modificam valoarea lui x prin y
cout << x << endl; // afiseaza 23.45
```

5. Transmiterea parametrilor

În C++ exista doua posibilitati de transmitere a parametrilor actuali catre o functie:

- prin valoare
- prin referinta

Transferul prin referinta este util si atunci când parametrul are dimensiune mare (ex: struct, class) si crearea în stiva a unei copii a valorii lui reduce timpul de executie .

```
void schimba(int &a, int &b)
{int  aux=a;
      a=b;
      b=aux;}
```

9.1 Diferente C-C++

6. Functii care pot returna variabile

În C++ functiile pot sa returneze variabile nu numai valori :

Ex. Calculul maximului dintre 2 valori reale

```
float &var_max(float &x, float &y)
{if (x > y) return x;
      else return y;}
```

Daca: `float a=2,b=3;`

Apel functie:

```
var_max(a,b)
```

Rezultat afisat : 3

7. Functii inline

In C++ exista functii inline : la fiecare apelare, codul funcţiei declarate inline este inserat în codul programului de către compilator

Ex.1 Calculul ipotenuzei

```
#include <iostream>
#include <cmath>
using namespace std;
inline double ipot(double a, double b)
{     return sqrt(a*a + b*b);}
int main()
{ cout <<ipot(3,4); return 0; }
//se apeleaza ca orice  functie
```

9.1 Diferente C-C++

8. Parametri impliciti

La apelarea unei functii cu parametri impliciti se poate omite specificarea parametrilor efectivi pentru acei parametri formali care au declarate valori implicite si se transfera automat valorile respective. Se pot specifica mai multe argumente cu valori implicite pentru o functie.

Ex. Calculul valorilor unei functii pentru parametri impliciti

```
int fct(int x=7, int y=9)
{       return x+y;}
int main()
{cout << fct()<<endl;
cout << fct(2) << endl;
cout << fct(2, 3) << endl;
return 0;}
```

```
16
11
5
```

9. Supraincarcarea functiilor si operatorilor

= posibilitatea de a atribui unui simbol mai multe semnificatii, care pot fi deduse în functie de context.

Ex. Definirea in acelasi program C++ a 2 functii diferite cu acelasi nume

```
#include <iostream>
using namespace std;
void fct(int a)
{  cout <<  a*a;}
void fct(char *a)
{    cout <<  a;}
int main()
{char mes[]="sir";
fct(2);
fct(mes);
return 0;}
```

Ce va afisa programul?

```
4sir
```

9.1 Diferente C-C++

10. Alocarea dinamica a memoriei: new/delete

Operatorii new si delete sunt similari functiilor din grupul malloc() si free(), dar constituie o metoda noua, superioara acestora si adaptata POO.

```
int * ip1, *ip2, *ip3;
ip1=new int;      // variabila întreaga neinitializata
ip2=new int(2); // variabila întreaga initializata cu 2
ip3=new int[100]; // tablou de 100 de întregi
```

11. Functii ca membri ai unei structuri

= posibilitatea de a atribui unui simbol mai multe semnificatii, care pot fi deduse în functie de context.

in C:

```
struct tip
      { int m1;
      int (*m2)(); }; //pointer la o functie ca membru
```

in C++:

```
struct tip
{
      int m1;
      int m2();    //functie ca membru
};
```

9.1 Diferente C-C++

12. Operatorul de rezolutie: ::

Limbajul C++ introduce operatorul de rezoluţie (::), care permite accesul la un obiect (sau variabilă) dintr-un bloc în care acesta nu este vizibil, datorită unei alte declaraţii.

```
…
char s[20]= "variabila globala";
void afiseaza(void)
{ char s[20] = “variabila locala”;
cout << s; //afiseaza variabila locala
cout << ::s; //afiseaza variabila globală
}
```

13. Tipul boolean: bool

In C: limbajul C nu oferă printre tipurile de bază tipul logic boolean. Acesta se poate simula prin folosirea unei enumerări : typedef enum {FALSE, TRUE} boolean;

In C++: exista predefinit tipul boolean

```
int main()
{
bool b = true;
b = false;
return 0;}
```

9.2 Biblioteci C++

Biblioteca Standard C++

Biblioteca Standard C++ (Standard C++ Library) cuprinde toate bibliotecile standard C precum si biblioteci dedicate C++.

Biblioteci functii I/O: <iostream>, <fstream>, etc: pentru procesarea fisierelor de intrare/iesire si operatii consola

Biblioteci functii siruri: <cstring> si <string> : functii procesare siruri de caractere/ text

Biblioteca functii matematice <cmath>: functii matematice

Biblioteca functii caractere <cctype>: pentru functii identificare/conversie caractere

9.2 Biblioteci C++

Biblioteca <cstring>

Functiile specifice sirurilor din <string.h> utilizate in C pot fi utilizate si in C++ incluzand biblioteca <cstring>

EXEMPLE

Ex. copiere siruri de caractere strcpy():

```
#include <iostream>
#include<cstring>
using namespace std;
int main() {
char a[10], b[10];
strcpy(a, "Info");
strcpy(b, "PCLP2");
cout<<a<<endl;
cout<<b<<endl; return 0;}
```

```
Info
PCLP2
```

9.2 Biblioteci C++

Biblioteca <cstring>

EXEMPLE

Ex. comparare siruri de caractere strcmp():

```
#include <iostream>
#include <cstring>
using namespace std;
int main() {
char a[10], b[10];
strcpy(a, "Info");
strcpy(b, "PCLP2");
cout<<a<<endl;
cout<<b<<endl;
if(!strcmp(a, b)) cout<<"siruri identice"<<endl;
  else cout <<"siruri diferite";return 0;}
```

```
Info
PCLP2
siruri diferite
```

9.2 Biblioteci C++

Siruri de caractere in C++: class string

string()	creaza un obiect string default (lungimea este 0)
string(const char *s)	creaza si initializeaza un string folosind un sir de caractere **s**
string(size_type n, char c)	creaza un string cu dimensiune **n** si fiecare element este initializat cu **c**
string(const string& s)	constructor de copiere
string(string s, int poz_init, int dim)	creaza un string din **s**, incepand cu elementul de pe pozitia **poz_init**, de lungime **dim** (sau pana la finalul lui s)

Ex. declarare sir caractere:

```
#include <iostream>
#include <string>
using namespace std;
int main() {
    string s1;     cin>>s1; //daca introduc de la tastatura “PCLP”
    cout << "s1: " << s1 << endl; //afiseaza sirul “PCLP”
    return 0;}
```

```
s1: PCLP
```

9.2 Biblioteci C++

Siruri de caractere in C++: class string

string()	creaza un obiect string default (lungimea este 0)
string(const char *s)	creaza si initializeaza un string folosind un sir de caractere **s**
string(size_type n, char c)	creaza un string cu dimensiune **n** si fiecare element este initializat cu **c**
string(const string& s)	constructor de copiere
string(string s, int poz_init, int dim)	creaza un string din **s**, incepand cu elementul de pe pozitia **poz_init**, de lungime **dim** (sau pana la finalul lui s)

Ex. declarare sir caractere:

```
#include <iostream>
#include <string>
using namespace std;
int main() {
    string s1; cin>>s1;    cout << "s1: " << s1 << endl; //introduc PCLP
    string s2("semestrul II");     cout << "s2: " << s2 << endl;
    return 0;}
```

```
s1: PCLP
s2: semestrul II
```

9.2 Biblioteci C++

Siruri de caractere in C++: class string

string()	creaza un obiect string default (lungimea este 0)
string(const char *s)	creaza si initializeaza un string folosind un sir de caractere **s**
string(size_type n, char c)	creaza un string cu dimensiune **n** si fiecare element este initializat cu **c**
string(const string& s)	constructor de copiere
string(string s, int poz_init, int dim)	creaza un string din **s**, incepand cu elementul de pe pozitia **poz_init**, de lungime **dim** (sau pana la finalul lui s)

Ex. declarare sir caractere initializat cu alt sir:

```
#include <iostream>
#include <string>
using namespace std;
int main() {
    string s1("PCLP“); cout << "s1: " << s1 << endl;
    string s2("semestrul II”); cout << "s2: " << s2 << endl;
    string s3(s2);     cout << "s3, copia celui de-al doilea string: " << s3 << endl;
    return 0;}
```

```
s1: PCLP
s2: semestrul II
s3, copia celui de-al doilea string: semestrul II
```

9.2 Biblioteci C++

Siruri de caractere in C++: class string

string()	creaza un obiect string default (lungimea este 0)
string(const char *s)	creaza si initializeaza un string folosind un sir de caractere **s**
string(size_type n, char c)	creaza un string cu dimensiune **n** si fiecare element este initializat cu **c**
string(const string& s)	constructor de copiere
string(string s, int poz_init, int dim)	creaza un string din **s**, incepand cu elementul de pe pozitia **poz_init**, de lungime **dim** (sau pana la finalul lui s)

Ex. declarare sir caractere cu dimensiune 5 caractere si initializat cu litera “o”:

```
#include <iostream>
#include <string>
using namespace std;
int main() {
string s4(5, 'o');
    cout << "s4: " << s4;    return 0;}
```

```
s4: ooooo
```

9.2 Biblioteci C++

Siruri de caractere in C++: class string

string()	creaza un obiect string default (lungimea este 0)
string(const char *s)	creaza si initializeaza un string folosind un sir de caractere **s**
string(size_type n, char c)	creaza un string cu dimensiune **n** si fiecare element este initializat cu **c**
string(const string& s)	constructor de copiere
string(string s, int poz_init, int dim)	creaza un string din **s**, incepand cu elementul de pe pozitia **poz_init**, de lungime **dim** (sau pana la finalul lui s)

Ex. declarare sir caractere initializat cu text copiat din string constant:

```
#include <iostream>
#include <string>
using namespace std;
int main() {
string  s5 = "al 5-lea"; //operator= este supradefinit pentru tipul de date string
    cout << "s5: " << s5;    return 0;}
```

```
s5: al 5-lea
```

9.2 Biblioteci C++

Siruri de caractere in C++: class string

string()	creaza un obiect string default (lungimea este 0)
string(const char *s)	creaza si initializeaza un string folosind un sir de caractere **s**
string(size_type n, char c)	creaza un string cu dimensiune **n** si fiecare element este initializat cu **c**
string(const string& s)	constructor de copiere
string(string s, int poz_init, int dim)	creaza un string din **s**, incepand cu elementul de pe pozitia **poz_init**, de lungime **dim** (sau pana la finalul lui s)

Ex. declarare sir caractere s6 din sirul s5 incepand cu elemental de pe poz 3 de dim 5:

```
#include <iostream>
#include <string>
using namespace std;
int main() {
string s5 = "al 5-lea";cout << "s5: " << s5<<endl;
string s6(s5, 3, 5);   cout << "s6: " << s6; return 0;}
```

```
s5: al 5-lea
s6: 5-lea
```

9.2 Biblioteci C++

Operatori supradefiniti in C++

operatorul +	string operator+ (const string& lhs, const string& rhs);
operatorul +=	string& operator+= (const string& str);
operatorul =	string& operator= (const string& str);
operatorii relationali (==, !=, <, >, ⇐, >=)	bool operator== (const string& lhs, const string& rhs);
operatorul »	istream& operator» (istream& is, string& str);
operatorul «	ostream& operator« (ostream& os, const string& str);

Ex. concatenare siruri : +

```
#include <iostream>
#include <string>
#include <iomanip>
using namespace std;
int main() {
string s1("text");  string s2("concatenat");     cout << s1 + " " + s2 << '\n';
return 0;}
```

```
text concatenat
```

9.2 Biblioteci C++

Operatori supradefiniti in C++

operatorul +	string operator+ (const string& lhs, const string& rhs);
operatorul +=	string& operator+= (const string& str);
operatorul =	string& operator= (const string& str);
operatorii relationali (==, !=, <, >, <=, >=)	bool operator== (const string& lhs, const string& rhs);
operatorul »	istream& operator» (istream& is, string& str);
operatorul «	ostream& operator« (ostream& os, const string& str);

Ex. concatenare siruri : +

```
#include <iostream>
#include <string>
#include <iomanip>
using namespace std;
int main() {
string s1("text");  string s2("concatenat");
s1+=s2;  cout << s1 ;
return 0;}
```

```
textconcatenat
```

9.2 Biblioteci C++

Operatori supradefiniti in C++

operatorul +	string operator+ (const string& lhs, const string& rhs);
operatorul +=	string& operator+= (const string& str);
operatorul =	string& operator= (const string& str);
operatorii relationali (==, !=, <, >, <=, >=)	bool operator== (const string& lhs, const string& rhs);
operatorul »	istream& operator» (istream& is, string& str);
operatorul «	ostream& operator« (ostream& os, const string& str);

Ex. copiere sir (prin atribuire): s1 va continue dupa atribuire s2

```
#include <iostream>
#include <string>
#include <iomanip>
using namespace std;
int main() {
    string s1 = "un string";     cout << s1 << endl;
    string s2 = "alt string";    s1 = s2;     cout << s1 << '\n';     return 0; }
```

```
un string
alt string
```

9.2 Biblioteci C++

Operatori supradefiniti in C++

operatorul +	string operator+ (const string& lhs, const string& rhs);
operatorul +=	string& operator+= (const string& str);
operatorul =	string& operator= (const string& str);
operatorii relationali (==, !=, <, >, <=, >=)	bool operator== (const string& lhs, const string& rhs);
operatorul »	istream& operator» (istream& is, string& str);
operatorul «	ostream& operator« (ostream& os, const string& str);

Ex. comparare siruri utilizand operatorii relationali. 2 conditii adevarate

```
int main() {
// operatorii relationali : ==, !=, <, >, <=, >=
    string s1 = "Programare";     string s2 = "PCLP";
    if(s1 == s2) cout << "cele doua cuvinte sunt egale\n";
    if(s1 < s2) cout << "s1 este lexicografic mai mic decat s2\n";
    if(s1 > s2) cout << "s2 este lexicografic mai mic decat s1\n";
    if(s1 != s2) cout << "cele doua cuvinte difera\n";
    if(s1 <= s2) cout << "s1 este lexicografic mai mic sau egal cu s2\n";    return 0;}
```

```
s2 este lexicografic mai mic decat s1
cele doua cuvinte difera
```

9.2 Biblioteci C++

Metodele clasei string

length → s.length()	(returneaza lungimea string-ului)
empty → s.empty()	returneaza daca sirul este gol sau nu
substr → s.substr(a,b)	returneaza un subsir de b caractere, incepand cu pozitia a; Daca b nu este precizat, se adauga la subsir toate caracterele incepand cu pozitia a
replace → s.replace(a,b,s2)	(inlocuieste portiunea de sir care incepe pe pozitia a si tine b caractere cu continutul sirului 2)
find → s.find(secv)	returneaza prima pozitie unde este gasita secventa secv, sau string::npos(constanta a clasei string), in cazul in care secventa nu a fost gasita)

Ex: Lungimea sirului

```
#include <iostream>
#include <string>
#include <iomanip>
using namespace std;
int main() {
    string s("PCLP2");    cout << s.length() << '\n';return 0;}
```

```
5
```

9.2 Biblioteci C++

Metodele clasei string

length → s.length()	(returneaza lungimea string-ului)
empty → s.empty()	returneaza daca sirul este gol sau nu
substr → s.substr(a,b)	returneaza un subsir de b caractere, incepand cu pozitia a; Daca b nu este precizat, se adauga la subsir toate caracterele incepand cu pozitia a
replace → s.replace(a,b,s2)	(inlocuieste portiunea de sir care incepe pe pozitia a si tine b caractere cu continutul sirului 2)
find → s.find(secv)	returneaza prima pozitie unde este gasita secventa secv, sau string::npos(constanta a clasei string), in cazul in care secventa nu a fost gasita)

Ex: Testare daca sirul este gol : empty()

```
#include <iostream>
#include <string>
#include <iomanip>
using namespace std;
int main() {
    string s("PCLP2");     cout << s.length() << '\n';
    if(s.empty()) cout << "s este gol\n";     if(!s.empty()) cout << "s nu este gol\n";     return 0;}
```

```
5
s nu este gol
```

9.2 Biblioteci C++

Metodele clasei string

length → s.length()	(returneaza lungimea string-ului)
empty → s.empty()	returneaza daca sirul este gol sau nu
substr → s.substr(a,b)	returneaza un subsir de b caractere, incepand cu pozitia a; Daca b nu este precizat se adauga la subsir toate caracterele incepand cu pozitia a
replace → s.replace(a,b,s2)	(inlocuieste portiunea de sir care incepe pe pozitia a si tine b caractere cu continutul sirului 2)
find → s.find(secv)	returneaza prima pozitie unde este gasita secventa secv, sau string::npos(constanta a clasei string), in cazul in care secventa nu a fost gasita)

Ex: Extrage 8 caractere din sir: subtr()

```
#include <iostream>
#include <string>
#include <iomanip>
using namespace std;
int main() {
    string s1("calculator");     string s2 = s1.substr(0, 8);
    cout << s2 << '\n';    return 0;}
```

```
calculat
```

9.2 Biblioteci C++

Metodele clasei string

length → s.length()	(returneaza lungimea string-ului)
empty → s.empty()	returneaza daca sirul este gol sau nu
substr → s.substr(a,b)	returneaza un subsir de b caractere, incepand cu pozitia a; Daca b nu este precizat, se adauga la subsir toate caracterele incepand cu pozitia a
replace → s.replace(a,b,s2)	(inlocuieste portiunea de sir care incepe pe pozitia a si tine b caractere cu continutul sirului 2)
find → s.find(secv)	returneaza prima pozitie unde este gasita secventa secv, sau string::npos(constanta a clasei string), in cazul in care secventa nu a fost gasita)

Ex: Inlocuieste caractere din sir replace()

```
#include <iostream>
#include <string>
#include <iomanip>
using namespace std;
int main() {
   string s1 = "doar";
   string s2 = s1.replace(1, 2, "inozau"); //literele o si a se inlocuiesc
   cout << s2 << '\n'; return 0;}
```

```
dinozaur
```

9.2 Biblioteci C++

Metodele clasei string

length → s.length()	(returneaza lungimea string-ului)
empty → s.empty()	returneaza daca sirul este gol sau nu
substr → s.substr(a,b)	returneaza un subsir de b caractere, incepand cu pozitia a; Daca b nu este precizat, se adauga la subsir toate caracterele incepand cu pozitia a
replace → s.replace(a,b,s2)	(inlocuieste portiunea de sir care incepe pe pozitia a si tine b caractere cu continutul sirului 2)
find → s.find(secv)	returneaza prima pozitie unde este gasita secventa secv, sau string::npos(constanta a clasei string), in cazul in care secventa nu a fost gasita)

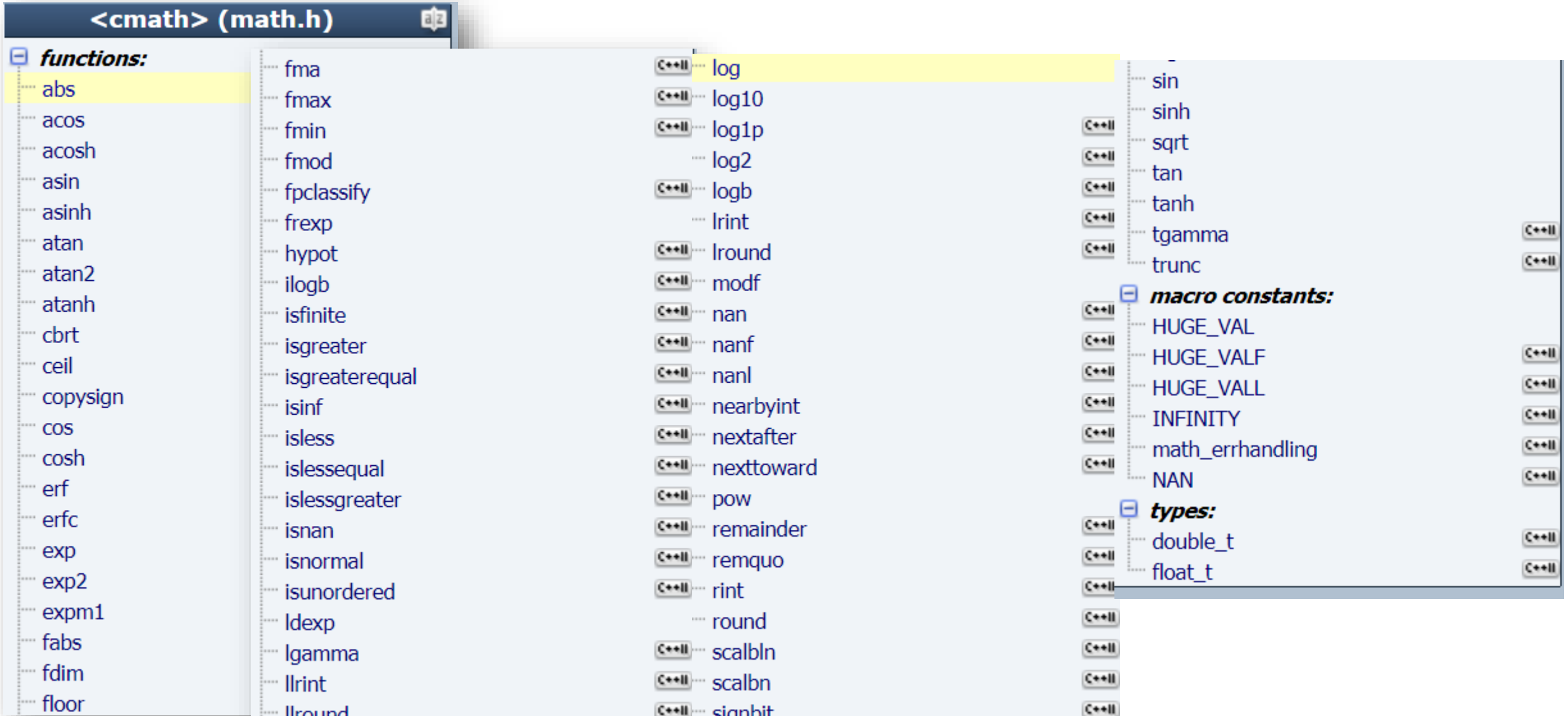
Ex: Identifica pozitia unde se gasesc prima data caracterele din sirul specificat : find()

```
#include <iostream>
#include <string>
#include <iomanip>
using namespace std;
int main() {
string s1("PCLP2");   int pos;     pos = s1.find("LP");     cout << pos << '\n'; return 0;}
```

2

9.2 Biblioteci C++

Biblioteca <cmath>



9.2 Biblioteci C++

Biblioteca <cmath>

EXEMPLE

Ex.functii trigonometrice

```
#include <iostream>
#include <cmath>
using namespace std;
const double pi=3.1415;
int main() {
cout << sin(pi/2)  << '\n'; //  1.000
cout << atan2(1, 1)  << '\n'; //  0.785 = PI / 4
cout << exp(3)       << '\n'; // 20.086
cout << log(exp(3))  << '\n'; //  3.000
cout << log2(1024)   << '\n'; // 10.000
cout << log10(0.01)  << '\n'; // -2.000
cout << pow(7, 2)    << '\n'; //49
cout << sqrt(25)     << '\n'; //  5.000
cout << floor(1.618) << '\n'; //  1.000
cout << ceil(1.618)  << '\n'; //  2.000
cout << abs(-13)     << '\n';  // 13   return 0;}
```

9.2 Biblioteci C++

Biblioteca <cmath>

EXEMPLE

Ex. identifica maximul si minimul dintre parametri

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{ cout <<"fmax (100.0, 1.0) = "<< fmax(100.0,1.0)<<endl;
  cout << "fmax (-100.0, 1.0) = "<< fmax(-100.0,1.0)<<endl;
  cout <<"fmin (100.0, 1.0) = "<< fmin(100.0,1.0)<<endl;
  cout << "fmin (-100.0, 1.0) = "<< fmin(-100.0,1.0)<<endl;
  return 0; }
```

Sintaxa:
double fmax (double x, double y);
float fmax (float x, float y);

```
fmax (100.0, 1.0) = 100
fmax (-100.0, 1.0) = 1
fmin (100.0, 1.0) = 1
fmin (-100.0, 1.0) = -100
```

Ex. Calculeaza ipotenuza cu functia hypot() (T Pitagora)

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{ double x=3, y=4; cout <<"cateta 1="<<x<<",cateta 2="<<y<<"=>ipotenuza="<<hypot (x, y);
 return 0;}
```

```
cateta 1=3,cateta 2=4=>ipotenuza=5
```

9.2 Biblioteci C++

Constante matematice

Symbol	Expression	Value
M_E	e	2.7182818284590452536
M_LOG2E	log2(e)	1.4426950408889634073
M_LOG10E	log10(e)	0.43429448190325182765
M_LN2	ln(2)	0.69314718055994530942
M_LN10	ln(10)	2.30258509299404568402
M_PI	pi	3.14159265358979323846
M_PI_2	pi/2	1.57079632679489661923
M_PI_4	pi/4	0.78539816339744830962
M_1_PI	1/pi	0.31830988618379067154
M_2_PI	2/pi	0.63661977236758134308
M_2_SQRTPI	2/sqrt(pi)	1.12837916709551257390
M_SQRT2	sqrt(2)	1.41421356237309504880
M_SQRT1_2	1/sqrt(2)	0.70710678118654752440

Constantele matematice nu sunt definite in Standard C/C++.

Pentru a le utiliza exista 2 variante:

- ❑ le definim ca si constante cu #define sau const, SAU
- ❑ introducem:

```
#define _USE_MATH_DEFINES // for C++
#include <cmath>

#define _USE_MATH_DEFINES // for C
#include <math.h>
```

9.2 Biblioteci C++

Constante matematice

EXEMPLE

Ex. Se afiseaza constante matematice

```
#include <iostream>
#if defined(_USE_MATH_DEFINES) && !defined(_MATH_DEFINES_DEFINED)
#define  _USE_MATH_DEFINES
#define M_E        2.71828182845904523536
#define M_LOG2E    1.44269504088896340736
#define M_PI       3.14159265358979323846
#define M_PI_2     1.57079632679489661923
#define M_PI_4     0.78539816339744830962
#define M_1_PI     0.31830988618379067154
#define M_2_PI     0.63661977236758134308
#endif  /* _USE_MATH_DEFINES */
#include <cmath>
using namespace std;
int main()
{   cout <<"pi="<<M_PI<<endl;          cout <<"pi/2="<<M_PI_2<<endl;
    cout<<"pi/4="<<M_PI_4<<endl;     cout<<"1/pi="<<M_1_PI<<endl;
    cout<<"2/pi="<<M_2_PI<<endl;     cout<<"e="<<M_E<<endl;
    cout<<"log2(e)="<<M_LOG2E<<endl;     return 0;}
```

```
pi=3.141593
pi/2=1.570796
pi/4=0.785398
1/pi=0.318310
2/pi=0.636620
e=2.718282
log2(e)=1.442695
```

9.2 Biblioteci C++

Conversii caractere : Functiile atoi () si atof()

EXEMPLE

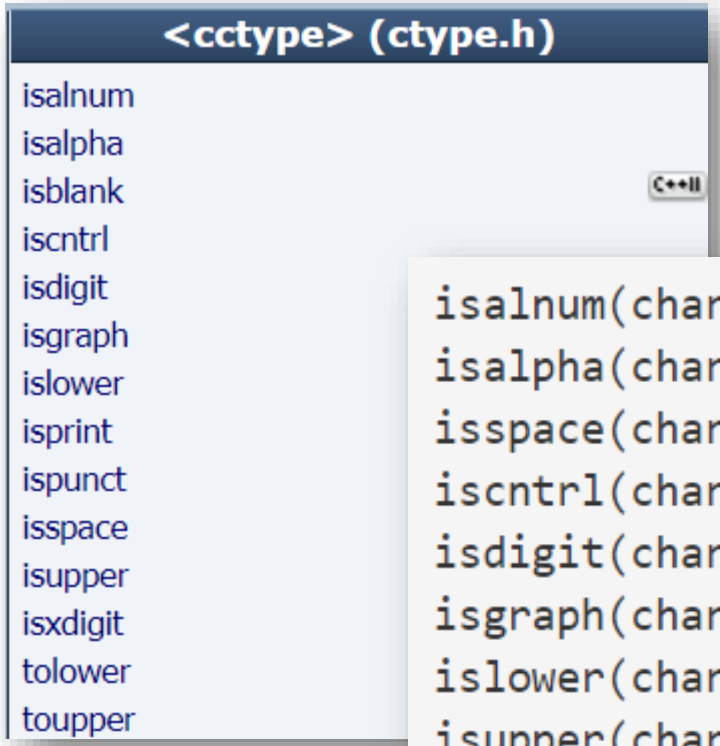
Ex. Functii conversie ASCII to integer sau double (nu float)

```
#include <iostream>
using namespace std;
int main() {
int x = atoi("123");
cout << x << '\n'; // 123
double y = atof("1.618");
cout << y << '\n'; return 0;}
```

```
123
1.618
```

9.2 Biblioteci C++

Biblioteca <cctype>




```
isalnum(char) // daca este litera sau cifre
isalpha(char) // daca este litera
isspace(char) // daca este spatiu sau caracter separator cum ar fi '\n'
iscntrl(char) // daca caracterul este caracter de control cum ar fi '\n'
isdigit(char) // daca este cifre
isgraph(char) // daca caracterul se poate printa
islower(char) // daca este litera mica
isupper(char) // daca este litera mare
isxdigit(char) // daca este cifra din reprezentarea unui numar in baza 16
ispunct(char) // daca este semn de punctuatie
```

9.2 Biblioteci C++

Biblioteca <cctype>

EXEMPLE

Ex. Se converteste un sir de caractere ASCII la intreg (an calendaristic)

```
#include <iostream>
#include <cctype>
using namespace std;
int main ()
{ char str[]="1776";// daca prima litera e o cifra presupun ca reprezinta un an
  int year;
  if (isdigit(str[0]))
  { year = atoi (str);   cout<<"An: "<<year;}
  return 0;}
```

```
An: 1776
```

9.2 Biblioteci C++

Biblioteca <cctype>

EXEMPLE

Ex. Testeaza daca caracterul este litera sau nu

```
#include <iostream>
#include <cctype>
using namespace std;
int main ()
{ int i=0;
  char str[]="C1D2";
  while (str[i])
  { if (isalpha(str[i])) cout<<"caracterul "<< str[i]<< " este litera"<<endl;
    else cout<<"caracterul " <<str[i]<< " nu este litera"<<endl;
    i++;  }
  return 0;}
```

```
caracterul C este litera
caracterul 1 nu este litera
caracterul D este litera
caracterul 2 nu este litera
```

9.2 Biblioteci C++

Biblioteca <cctype>

EXEMPLE

Ex. Testeaza daca caracterul este litera sau nu si numara cate litere si non-litere sunt in sir

```
#include <iostream>
#include <cctype>
using namespace std;
int main ()
{ int i=0, k=0,c=0;
  char str[]="C++ sem 2";
  while (str[i])
  { if (isalpha(str[i])) {cout<<"caracterul "<< str[i]<< " este litera"<<endl;k++;}
    else {cout<<"caracterul " <<str[i]<< " nu este litera"<<endl;c++;}
    i++;  }
    cout<<"litere:"<<k;cout<<" non litere:"<<c;
  return 0;}
```

```
caracterul C este litera
caracterul + nu este litera
caracterul + nu este litera
caracterul   nu este litera
caracterul s este litera
caracterul e este litera
caracterul m este litera
caracterul   nu este litera
caracterul 2 nu este litera
litere:4 non litere:5
```

9.2 Biblioteci C++

Biblioteca <cctype>

EXEMPLE

Ex. Converteste toate caracterele majuscule dintr-un sir in litere mici

```
#include <iostream>
#include <cctype>
using namespace std;
int main ()
{ int i=0;
  char str[]="LIMBAJUL C++\n";
  char c;
  while (str[i])
  {    c=str[i];
       if (isupper(c)) c=tolower(c);
       cout<<c;    i++;  }
  return 0;}
```

```
limbajul c++
```

9.3 STL in C++

Standard Template Library (STL) in C++

Biblioteca STL: colecție de algoritmi, structuri de date și alte componente care pot fi utilizate pentru a simplifica si optimiza codul C++.

Componente de baza:

Algoritmi: pentru sortare, selectie și căutare binară a datelor stocate în containere.

Containere: vector, listă, map, set și stivă (stack), utile pentru a stoca și manipula date.

Iteratoare: obiecte care oferă o modalitate de a parcurge elementele unui container. Ex: forward_iterator, bidirectional_iterator și random_access_iterator.

Functors-Obiecte funcție: sunt obiecte care pot fi folosite ca argumente de funcție pentru algoritmi. Ele oferă o modalitate de a transmite o funcție unui algoritm

Adaptoare: sunt componente care modifică comportamentul altor componente din STL. Ex. adaptorul reverse_iterator poate fi folosit pentru a inversa ordinea elementelor dintr-un container.

9.3 STL in C++

Standard Template Library (STL) in C++

Structura de date implementata	Nume STL	#include
Tablou dinamic	vector	<vector>
Lista dublu inlantuita	list	<list>
Stiva	stack	<stack>
Coada cu doua capete	queue	<queue>
Arbore binar/Multime	set	<set>
Multime (se pot repeta valorile)	multiset	<set>
Hash table	map	<map>
Hash table (se pot repeta valorile)	multimap	<map>
Max-heap	priority_queue	<queue>

9.3 STL in C++

1) Algoritmi: exemple

Sortare crescator:
sort(startaddress, endaddress)

Ex.: Sortare sir ordine crescatoare

```
#include <algorithm>
#include <iostream>
using namespace std;
int main()
{int arr[] = {3, 5, 1, 2, 4};
cout<<“sirul in ordine
      crescatoare: ”;
sort(begin(arr), end(arr));
for (int i : arr)
    {cout << i << " ";}
return 0;}
```

```
1 2 3 4 5
```

Sortare descrescator:
sort(first_iterator, last_iterator, greater<int>())

Ex.: Cautare element in sir sortat

```
#include <algorithm>
#include <iostream>
using namespace std;
int main()
{int arr[] = {3, 5, 1, 2, 4};
sort(begin(arr), end(arr), greater<int>());
for (int i : arr)
    {cout << i << " ";}
return 0;}
```

```
5 4 3 2 1
```

9.3 STL in C++

1) Algoritmi: exemple

Cautare binara:

binary_search(startaddress, endaddress, valuetofind)

Ex.: Cautare element in sir sortat

```
#include <algorithm>
#include <iostream>
using namespace std;
int main()
{ int x;
int arr[] = {3, 5, 1, 2, 4};
sort(begin(arr), end(arr));
for (int i : arr)     {cout << i << " ";}
cout<<"x=";cin>>x;
if (binary_search(arr, arr+5, x))
cout <<x<<" este in sir";
else cout <<x<<" nu este in sir";
return 0;}
```

```
1 2 3 4 5 x=3
3 este in sir
```

Inversare:

reverse(first_iterator, last_iterator)

Ex.: Inversare elemente sir

```
#include <algorithm>
#include <iostream>
using namespace std;
int main()
{
    int arr[] = {3, 5, 1, 2, 4};
    reverse(begin(arr), end(arr));
    for (int i : arr)     {cout << i << " ";}
    return 0;
}
```

```
4 2 1 5 3
```

9.3 STL in C++

2) Containere: tipuri

Containere secventiale:
- ❑ vector
- ❑ list
- ❑ deque
- ❑ array
- ❑ forward_list (in C++11)

Container Adaptors:
- ❑ queue
- ❑ priority_queue
- ❑ stack

Containere asociative: implementeaza date structurate sortate in care se pot face cautari rapide
- ❑ set
- ❑ multiset
- ❑ map
- ❑ multimap

9.3 STL in C++

2) Containere secventiale: vectori

În C++ siruri de numere pot fi declarate :

- prin tablouri
- prin vectori: tablouri dinamice cu elemente omogene care au proprietatea de a se redimensiona automat când se adaugă sau se șterge un element

SINTAXA

vector<type> nume;

unde: type **poate fi : int, float, double, long, …**

nume: numele variabilei

Obs:

1.Trebuie inclusa biblioteca : #include <vector>

2. in codeblocks: activați suportul C++11 în compilator

```
// 1. lista de initializare
vector<int> vector1 = {1, 2, 3, 4, 5};
// 2. Initializare uniforma
vector<int> vector2 {1, 2, 3, 4, 5};
//3.Initializare cu aceeasi valoare
vector<int> vector3(5, 12); // echivalent cu : vector<int> vector3 = {12, 12, 12, 12, 12};
//4.Initializare cu valoarea 0
vector<int> vector4(n); cin>>n;
// se crează vector4 cu n elemente, egale cu 0
//5. Creare vector si initializare cu x
vector<int> vector5(n , x); cin>>n>>x;
```

9.3 STL in C++

2) Containere secventiale: vectori

EXEMPLU

Ex. : initializarea si afisarea vectorilor

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
vector<int> vector1 ={1, 2, 3, 4, 5};
vector<int> vector2 {6, 7, 8, 9, 10};
vector<int> vector3(5, 12);
cout << "vector1 = ";
for (const int i : vector1) {    cout << i << "  ";  }
cout << "\nvector2 = ";
for (const int i : vector2) {    cout << i << "  ";  }
cout << "\nvector3 = ";
  for (int i : vector3) {    cout << i << "  ";  }
return 0;}
```

```
vector1 = 1  2  3  4  5
vector2 = 6  7  8  9  10
vector3 = 12  12  12  12  12
```

9.3 STL in C++

2) Containere secventiale : vectori atribuire

SINTAXA

Spre deosebire de tablourile standard C, vectorilor li se pot atribui valori în orice moment.

```
vector<int> A , B = {2 , 4, 6 , 8};
A = B;
A = {1 , 2 , 3 , 4 , 5};
```

2) Containere secventiale : vectori dimensiune: size(), capacity()

Numărul de elemente din vector poate fi determinat cu funcția size():

```
vector<int> A = {2 , 4 , 6 , 8};
cout << A.size(); // 4
```

Obs: Memoria alocată pentru un vector este mai mare decât memoria folosită. Acest lucru se întâmplă pentru a se evita realocarea memoriei (operație care presupune copierea tuturor elementelor) de fiecare dată când se adaugă elemente.

Funcția capacity() returnează numărul de elemnte pe care le poate avea vectorul la momentul curent, fără a se realoca memoria. capacity() >= size(), nr de elemente care pot fi adăugate fără realocare : capacity() - size().

9.3 STL in C++

2) Containere secventiale : vectori redimensionare resize()

SINTAXA

Funcția **resize()** permite redimensionarea unui vector

- **V.resize(n);**
- **V.resize(n , x);**

Vectorul V se redimensionează încât să aibă n elemente:

- dacă n < V.size(), în vector vor rămâne primele n elemente, celelalte vor fi șterse.
- dacă n > V.size(), în vector se vor adăuga n - V.size() elemente egale cu x , în cazul (2.) sau egale cu valoarea implicită pentru tipul elementelor din vector, în cazul (1.). Această valoare este 0 pentru tipurile numerice.

9.3 STL in C++

2) Containere secventiale : vectori adăugare element: push_back()

Ex.. : adaugare elemente la sfarsitul vectorului

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
  vector<int> num {1, 2, 3, 4, 5};
  cout << "Vector Initial:";
  for (const int i : num) {
    cout << i << "  ";
  }
  // adauga 6 si 7 la vector
  num.push_back(6);
  num.push_back(7);
  cout << "\nDupa adaugare: ";
  for (const int i : num) {
    cout << i << "  ";  }
  return 0;}
```

```
Vector Initial:1  2  3  4  5
Dupa adaugare: 1  2  3  4  5  6  7
```

Ex.. : initializare elemente in vector cu push_back

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
vector<int> A;
for(int i = 0 ; i < 5 ; i++)
    A.push_back(2 * i);
cout<<"varianta 1"<<endl;
for(int i = 0 ; i < A.size() ; i++)
    cout << A[i] << ' ';
cout <<endl<<"dimensiune vector: "<<A.size()<<endl;
cout<<"varianta 2"<<endl;
for (const int i : A)
    cout << i << ' ';
cout <<endl<<"dimensiune vector: "<<A.size();
return 0;}
```

```
varianta 1
0 2 4 6 8
dimensiune vector: 5
varianta 2
0 2 4 6 8
dimensiune vector: 5
```

9.3 STL in C++

2) Containere secventiale : vectori Accesare/modificare elemente: at(), front(), end()

funcția at(): returnează o referință la elementul de la poziția dată

Ex.. : inlocuire element din vector

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
vector<int> A = {2 , 4 , 6 , 8 , 10};
cout<<"Sir initial A:";
for(int x : A)
    cout << x << " ";
A.at(1) = 20; //A = {2 , 20 , 6 , 8 , 10}
cout<<"\nSir dupa inlocuire element A[1]=20 :";
for(int x : A)
    cout << x << " ";
return 0;}
```

```
Sir initial A:2 4 6 8 10
Sir dupa inlocuire element A[1]=20 :2 20 6 8 10
```

Ex.. : inlocuire primul, al 2lea si ultimul element din vector

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
vector<int> A = {2 , 4 , 6 , 8 , 10};
cout<<"Sir initial A:";
for(int x : A)
    cout << x << " ";
A.at(1) = 20; //A = {2 , 20 , 6 , 8 , 10}
A.front() = 7; //A = {7 , 20 , 6 , 8 , 10}
A.back() = 5; //A = {7 , 20 , 6 , 8 , 5}
cout<<"\nSir dupa inlocuire elemente:";
for(int x : A)
    cout << x << " ";}
```

```
Sir initial A:2 4 6 8 10
Sir dupa inlocuire elemente:7 20 6 8 5
```

9.3 STL in C++

2) Containere secventiale : vectori stergere elemente: pop_back(), clear()

Ex.. : stergere ultimul element din vector

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
  vector<int> nrprime={2, 3, 5, 7};
  // initial vector
  cout << "Sir initial: ";
  for (int i : nrprime) {
    cout << i << " ";  }
  // remove the last element
  nrprime.pop_back();
  // final vector
  cout << "\nSir dupa stergere ultimul element: ";
  for (int i : nrprime) {
    cout << i << " ";  }
  return 0;
}
```

```
Sir initial: 2 3 5 7
Sir dupa stergere ultimul element: 2 3 5
```

Ex.. : sterge toate elementele din vector

```
include <iostream>
#include <vector>
using namespace std;
int main() {
vector<int> A = {2 , 4 , 6 , 8 , 10};
cout<<"Sir initial A:";
for(int x : A)
    cout << x << " ";
cout<<"\nSir dupa golire :";
A.clear();
for(int x : A)
    cout << x << " ";
return 0;
}
```

```
Sir initial A:2 4 6 8 10
Sir dupa golire :
```

9.3 STL in C++

2) Containere secventiale : vectori algoritmi

Interschimbare : swap(x,y)

Ex.: schimba elementele a 2 siruri

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
int main() {
vector<int> a = {1, 2, 3};
vector<int> b = {4, 5, 6};
cout << "Before swap:" << endl;
cout << “\na: ";for(int num : a) { cout << num << " ";}
cout << "b: "; for(int num : b) { cout << num << " ";}
swap(a, b);
cout << “\nAfter swap:" << endl;cout << "a: ";
for(int num : a) {cout << num << " ";}
cout << “\nb: ";
for(int num : b) {cout << num << " ";}
return 0;}
```

```
Before swap:
a: 1 2 3
b: 4 5 6
After swap:
a: 4 5 6
b: 1 2 3
```

Stergere primul element cu valoarea val specificata: remove(first, last, val);

Ex.: Stergere primul element =2

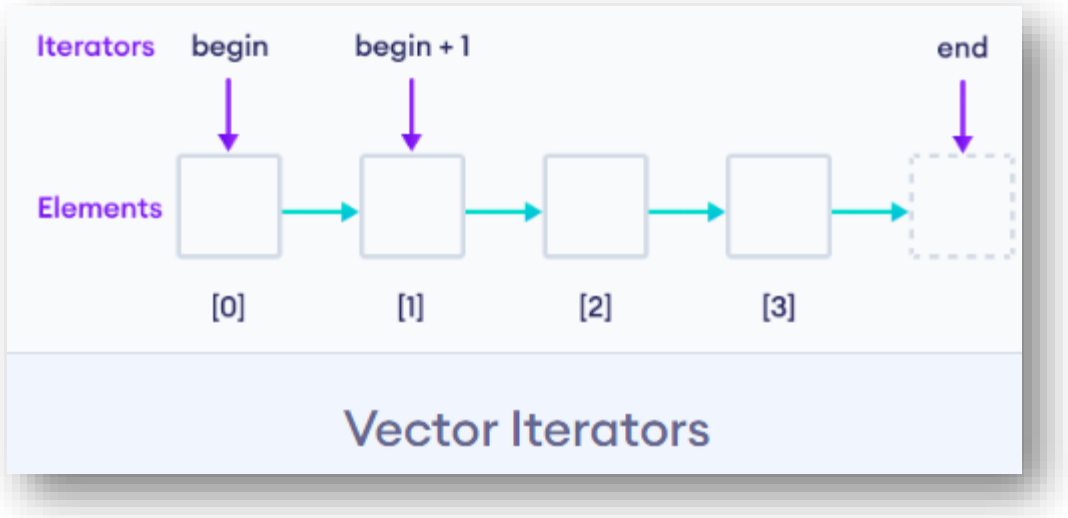
```
#include <algorithm>
#include <vector>
#include <iostream>
using namespace std;
int main() {
vector<int> vec = {4, 2, 3, 2, 5};
cout << "Before deletion: ";
for(int num : vec) {
    cout << num << " ";}
cout << endl;
remove(vec.begin(), vec.end(), 2);
cout << "After deletion: ";
for(int num : vec) {
    cout << num << " ";}
return 0;}
```

```
Before deletion: 4 2 3 2 5
After deletion: 4 3 5 2 5
```

9.3 STL in C++

3) Iteratori- accesare elemente vector

Iteratori: utilizati în principal în secvențe de numere, caractere etc. reducand complexitatea și timpul de execuție a programului.



9.3 STL in C++

3) Iteratori- accesare elemente vector

Iteratori: utilizati în principal în secvențe de numere, caractere etc. reducand complexitatea și timpul de execuție a programului.

Ex.. : afisare elemente vector utilizand iterator

```
#include<iostream>
// #include<iterator> // optional
#include<vector>
using namespace std;
int main()
{ vector<int> A= { 1, 2, 3, 4, 5 };
vector<int>::iterator ptr;
cout << "Elementele sirului: ";
for (ptr = A.begin(); ptr < A.end(); ptr++)
   cout << *ptr << " ";
return 0;}
```

```
Elementele sirului: 1 2 3 4 5
```

Ex.. : avansare pozitie iterator

```
#include<iostream>
#include<vector>
using namespace std;
int main()
{ vector<int> A= { 1, 2, 3, 4, 5 };
vector<int>::iterator ptr;
cout << "Pozitia initiala iterator: ";
ptr = A.begin();
cout << *ptr << " ";
advance(ptr, 3);
cout << "\nPozitia iteratorului dupa avansare in sir: ";
cout << *ptr << " ";
return 0;}
```

```
Pozitia initiala iterator: 1
Pozitia iteratorului dupa avansare in sir: 4
```

9.3 STL in C++

3) Iteratori- Stergere elemente vector: iterator & erase()

Ex.. : stergere primul element din vector

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
vector<int> A = {2 , 4 , 6 , 8 , 10};
vector<int>::iterator it;
it = A.begin();
cout<<"Sir initial A:";
for(int x : A)     cout << x << " ";
cout<<"\nSir dupa golire :";
A.erase(it);
for(int x : A)     cout << x << " ";
return 0;}
```

```
Sir initial A:2 4 6 8 10
Sir dupa golire :4 6 8 10
```

it= A.begin()+2; // stergere al 3-lea element

```
Sir initial A:2 4 6 8 10
Sir dupa stergere al 3-lea element :2 4 8 10
```

Ex.. : sterge secventa elemente din vector

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
vector<int> A = {2 , 4 , 6 , 8 , 10};
vector<int>::iterator it;
cout<<"Sir initial A:";
for(int x : A)
    cout << x << " ";
it= A.begin()+1;
A.erase(it , it+2);
cout<<"\nSir dupa stergere elemente :";
for(int &x : A)
    cout << x << " ";
return 0; }
```

```
Sir initial A:2 4 6 8 10
Sir dupa stergere elemente :2 8 10
```

9.3 STL in C++

3) Iteratori- Insertie elemente vector in alt vector: iterator & copy (),inserter()

Ex.. : insertie elemente dintr-un vector in altul

```
#include<iostream>
#include<vector>
using namespace std;
int main()
{ vector<int> A= { 1, 2, 3, 4, 5 };
vector<int>  B = {10, 20, 30};
cout<<"A=";
for (int x : A)         cout << x << " ";
cout<<"\nB=";
for (int x : B)         cout << x << " ";
vector<int>::iterator ptr = A.begin();
advance(ptr, 3); // set pozitie
// insereaza B dupa al 3lea element din A
copy(B.begin(), B.end(), inserter(A,ptr));
cout <<"\nVectorul A (dupa insertia elementelor din vectorul B): ";
cout<<"\nA=";
for (int x : A)         cout << x << " ";
return 0;}
```

```
A=1 2 3 4 5
B=10 20 30
Vectorul A (dupa insertia elementelor din vectorul B):
A=1 2 3 10 20 30 4 5
```

9.3 STL in C++

3) Iteratori- Insertie elemente vector in alt vector: iterator & copy (),inserter()

Ex.. : insertie elemente dintr-un vector in altul

```
#include<iostream>
#include<vector>
using namespace std;
int main()
{ vector<int> A= { 1, 2, 3, 4, 5 };
vector<int>  B = {10, 20, 30};
cout<<"A=";
for (int x : A)         cout << x << " ";
cout<<"\nB=";
for (int x : B)         cout << x << " ";
vector<int>::iterator ptr = A.begin();
advance(ptr, 3); // set pozitie
// insereaza B dupa al 3lea element din A
copy(B.begin(), B.end(), inserter(A,ptr));
cout <<"\nVectorul A (dupa insertia elementelor din vectorul B): ";
cout<<"\nA=";
for (int x : A)         cout << x << " ";
return 0;}
```

```
A=1 2 3 4 5
B=10 20 30
Vectorul A (dupa insertia elementelor din vectorul B):
A=1 2 3 10 20 30 4 5
```

9.3 STL in C++

3) Iteratori- Insertie elemente vector in alt vector: iterator & emplace()

Ex.. : insertie elemente dintr-un vector in altul

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{vector<int> A = { 10, 20, 30 };
cout<<"A=";
for (int x : A)
    cout << x << " ";
auto it = A.emplace(A.begin(), 15);
cout << "A dupa insertie element 15 la inceput de sir: ";
for (auto it = A.begin(); it != A.end(); ++it)
    cout << *it << " ";
return 0;
}
```

```
A=10 20 30 A dupa insertie element 15 la inceput de sir: 15 10 20 30
```

9.3 STL in C++

4) Containere secventiale liste

SINTAXA

Liste: secvente de date care pot fi allocate in zone de memorie necontinue.

list <data-type> name_of_list;

Obs: Trebuie inclusa biblioteca : #include <list>

Ex.. : creare si afisare lista

```
#include <iostream>
#include <list>
using namespace std;
int main()
{    // definire lista
    list<int> list{12,45,8,6};
    for (int i : list) //afisare lista
     { cout << i << ' ';    }
    return 0;}
```

```
12 45 8 6
```

front() – Returnează valoarea primului element din listă.
back() – Returnează valoarea ultimului element din listă.
push_front() – Adaugă un nou element la începutul listei.
push_back() – Adaugă un nou element la sfârșitul listei.
pop_front() – Sterge primul element al listei și reduce dimensiunea listei cu 1.
pop_back() – Sterge ultimul element al listei și reduce dimensiunea listei cu 1.
insert() – Inserează elemente noi în listă înaintea elementului într-o poziție specificată.
size() – Returnează numărul de elemente din listă.
begin() – returnează un iterator care indică primul element al listei.
end() – returnează un iterator care indică ultimul element.

9.3 STL in C++

4) Containere secventiale **liste**

Ex.. : operatii lista, elemente initializate in program

```
#include <iostream>
#include <list>
using namespace std;
void afis(list<int> g)
{   list<int>::iterator it;
    for (it = g.begin(); it != g.end(); ++it)
        cout << ' ' << *it;}
int main()
{   list<int> list{12,45,8,6};
    cout<<"lista initiala:"; afis(list);
    cout<<"\nPrimul element:"<<list.front();
    cout<<"\nUltimul element:"<<list.back();
    list.sort();          cout<<"\nLista sortata:"; afis(list);
    list.reverse();     cout<<"\nLista sortata revers:"; afis(list);
    list.pop_front(); cout<<"\nLista sortata fara primul element:";afis(list);
    list.pop_back(); cout<<"\nLista sortata fara ultimul element:";afis(list);
    return 0;}
```

```
lista initiala: 12 45 8 6
Primul element:12
Ultimul element:6
Lista sortata: 6 8 12 45
Lista sortata revers: 45 12 8 6
Lista sortata fara primul element: 12 8 6
Lista sortata fara ultimul element: 12 8
```

9.3 STL in C++

4) Containere secventiale liste cu iterator

Ex.. : creare lista si parcurgere lista in ambele directii

```
#include <iostream>
#include <list>
using namespace std;
int main() {
list<int> nums {1, 2, 3, 4, 5};
// initialize iterator to point to beginning of nums
list<int>::iterator itr = nums.begin();
cout << "Moving forward: " << endl;
// display the elements in forward order
while (itr != nums.end()) {
      cout << *itr  << ", ";
      itr++; }//  move iterator by 1 position forward
cout << endl << "Moving backward: " << endl;
// display the elements in backward order
while (itr != nums.begin()) {
      if (itr != nums.end()) {
      cout << *itr << ", ";   }
      itr--; }//  move iterator by 1 position backward
cout << *itr << endl;
return 0; }
```

```
Moving forward:
1, 2, 3, 4, 5,
Moving backward:
5, 4, 3, 2, 1
```

9.3 STL in C++

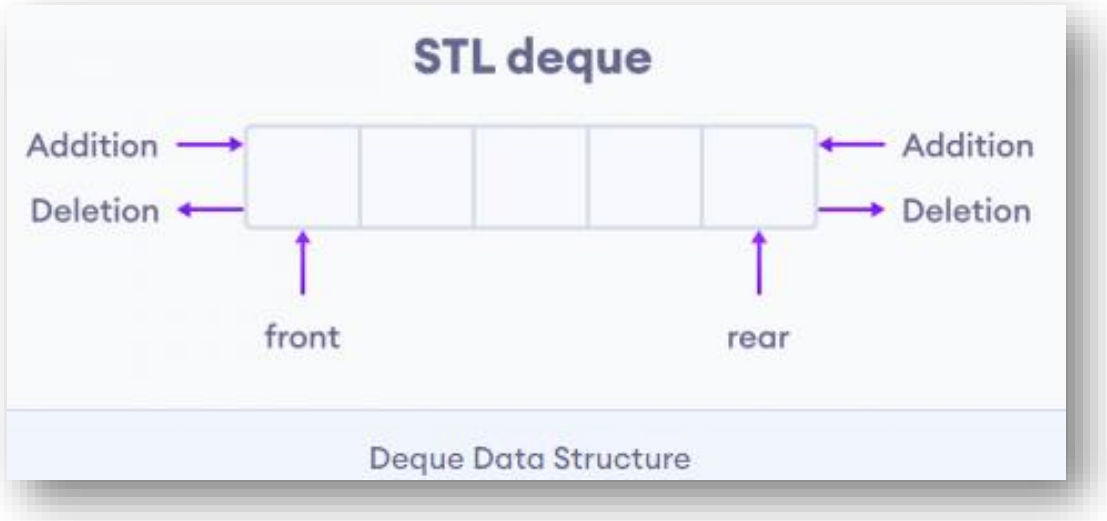
4) Containere secventiale deque

SINTAXA

Deque: Double-ended queues sunt containere în care se pot adauga elemente la ambele capete.Sunt asemănătoare vectorilor, dar sunt mai eficiente în cazul inserării și ștergerii elementelor.

deque <data-type> name_of_deque;

Obs: Trebuie inclusa biblioteca : #include <deque>



9.3 STL in C++

4) Containere secventiale deque

Ex.. : creare si afisare deque

```
#include <iostream>
#include <deque>
using namespace std;
void afis(deque<int> g)
{   for (int num : g)   cout << num << " ";    }
deque<int> readdeque() {
deque<int> g;
int num;
cout << "Valori intregi, non caracter pentru stop"<<endl;
while (cin >> num) {
  g.push_back(num);    }
return g;}
```

```
Valori intregi, non caracter pentru stop
1 2 3 8 4 6.
lista initiala:1 2 3 8 4 6
Primul element:1
Ultimul element:6
adaugam nr 10 la inceput:10 1 2 3 8 4 6
adaugam nr 5 la sfarsit:10 1 2 3 8 4 6 5
Stergem primul element :1 2 3 8 4 6 5
Stergem ultimul element:1 2 3 8 4 6
```

```
int main()
{   deque<int> d1 = readdeque();
cout<<"lista initiala:"; afis(d1);
cout<<"\nPrimul element:"<<d1.front();
cout<<"\nUltimul element:"<<d1.back()<<endl;
d1.push_front(10);cout<<"adaugam nr 10 la inceput:";
      afis(d1);
d1.push_back(5);cout<<"\nadaugam nr 5 la sfarsit:";
      afis(d1);
d1.pop_front();
cout<<"\nStergem primul element :";afis(d1);
d1.pop_back();
cout<<"\nStergem ultimul element:";afis(d1);
return 0;}
```

9.3 STL in C++

4) Containere secventiale array

SINTAXA

array: sunt containere asemănătoare vectorilor, cu elemente de acelasi tip si dimensiune fixa

array <data-type, size> name_of_array; Obs: Trebuie inclusa biblioteca : #include <array>

Ex.. : creare si afisare array

```
#include<iostream>
#include<array> // for array, at()
#include<tuple> // for get()
using namespace std;
int main()
{array<int,6> ar = {1, 2, 3, 4, 5, 6};
cout << "Elemente array utilizand container: ";
for (int i : ar)  cout << i << ' ';
cout << "\nElemente array utilizand at(): ";
for ( int i=0; i<6; i++)   cout << ar.at(i) << " ";
cout << "\nElemente array utilizand get: : ";
cout << get<0>(ar) << " " << get<1>(ar) << " ";
cout << get<2>(ar) << " " << get<3>(ar) << " ";
cout << get<4>(ar) << " " << get<5>(ar) << " ";
cout << "\nElemente array utilizand [] : ";
for ( int i=0; i<6; i++)   cout << ar[i] << " ";
cout << endl;return 0;}
```

```
Elemente array utilizand container: 1 2 3 4 5 6
Elemente array utilizand at(): 1 2 3 4 5 6
Elemente array utilizand get: : 1 2 3 4 5 6
Elemente array utilizand [] : 1 2 3 4 5 6
```

9.3 STL in C++

5) Containere adaptive

SINTAXA

- queue: container adaptors , FIFO (First In First Out)
- priority_queue: container proiectat astfel încât primul element al cozii să fie fie cel mai mare, fie cel mai mic dintre toate elementele din coadă, iar elementele să fie în ordine crescătoare sau descrescătoare (=>prioritate)
- stack: container adaptors , LIFO (Last In First Out)

9.3 STL in C++

5) Containere adaptive queue (FIFO)

SINTAXA

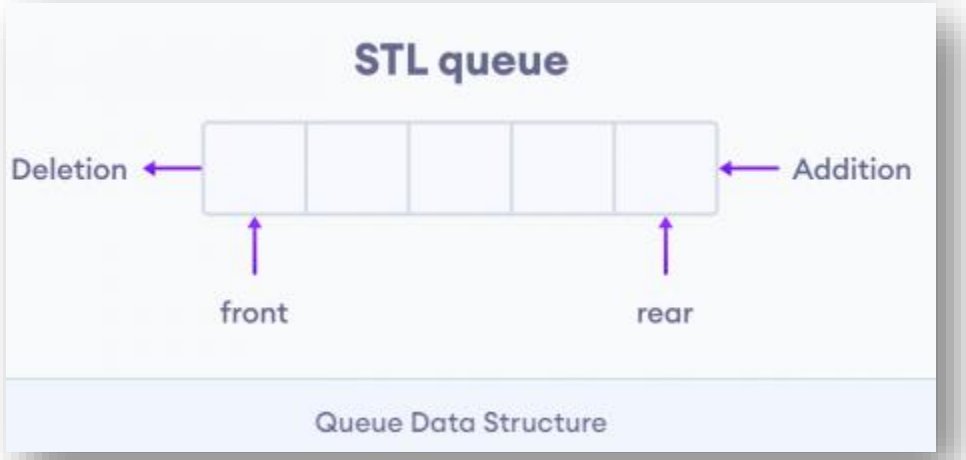
queue <data-type> name_of_queue; Obs: Trebuie inclusa biblioteca : #include <queue>

Ex.. : creare si afisare queue

```
#include <iostream>
#include <queue>
using namespace std;
void showq(queue<int> q1)
{  queue<int> g = q1;
   while (!g.empty()) { cout << '\t' << g.front();  g.pop();    }
    cout << '\n';}
int main()
{queue<int> c;
c.push(10); c.push(20); c.push(30);
cout << "queue  : "; showq(c);
cout << "\nqueue size() : " << c.size();
cout << "\nqueue.front() : " << c.front();
cout << "\nqueue.back() : " << c.back();
cout << "\nqueue.pop() : ";
cout<<"Sterg primul element\n";
c.pop(); cout<<"queue dupa stergere prim element:";
showq(c);return 0;}
```

```
queue  :        10      20      30

queue size() : 3
queue.front() : 10
queue.back() : 30
queue.pop() : Sterg primul element
queue dupa stergere prim element:       20      30
```



9.3 STL in C++

5) Containere adaptive priority_queue

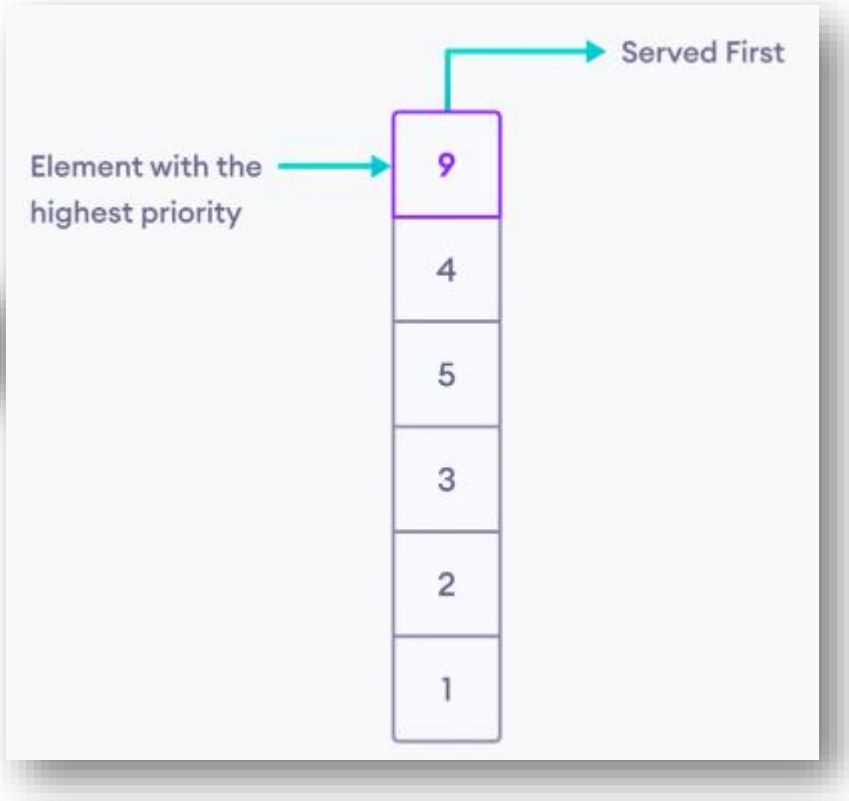
SINTAXA

priority_queue<data-type> name_of_queue;
Obs: top element este implicit cea mai mare valoare
Trebuie inclusa biblioteca : #include <queue>

Ex.. : creare si afisare priority_**queue**

```
#include <iostream>
#include <queue>
using namespace std;
int main()
{int arr[6] = { 10, 2, 4, 8, 6, 9 };
priority_queue<int> pq;
cout << "sir: ";
for (auto i : arr)   cout << i << ' ';
cout << endl;
for (int i = 0; i < 6; i++) {pq.push(arr[i]);}
cout << "Priority Queue: ";
while (!pq.empty()) {cout << pq.top() << ' '; pq.pop();}
return 0;}
```

```
sir: 10 2 4 8 6 9
Priority Queue: 10 9 8 6 4 2
```



9.3 STL in C++

5) Containere adaptive stack (LIFO)

SINTAXA

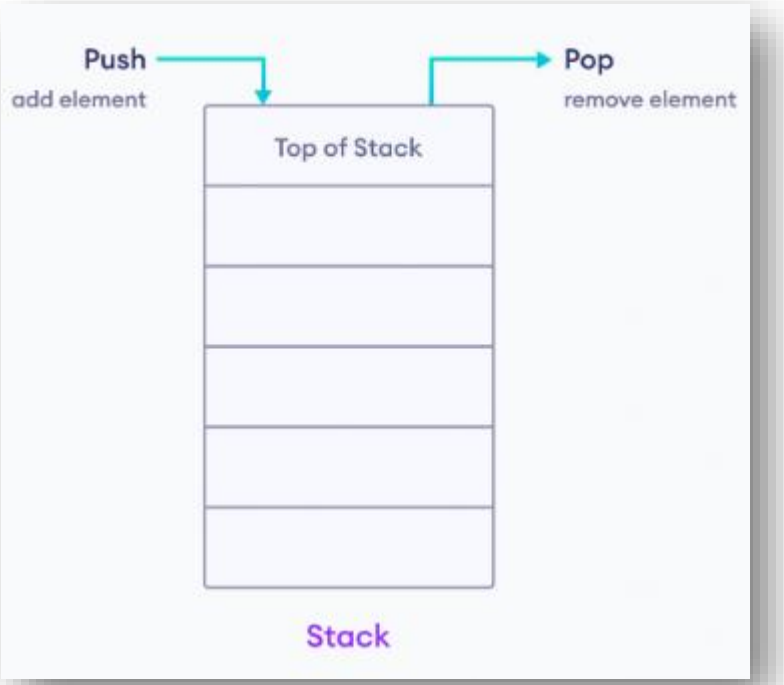
stack<data-type> name_of_queue;

Obs:Trebuie inclusa biblioteca : #include <stack>

Ex.. : creare si afisare stack

```
#include <iostream>
#include <stack>
using namespace std;
int main() {
stack<int> s1;
s1.push(21
s1.push(22);
s1.push(24);
s1.push(25);
cout<<"stack: ";
while (!s1.empty()) {  cout << s1.top() <<" "; s1.pop();}
return 0;    }
```

```
stack: 25 24 22 21
```



9.3 STL in C++

6) Containere asociative

SINTAXA

- Set: tip de container asociativ în care fiecare element trebuie să fie unic (valoarea elementului îl identifică). Valorile sunt stocate într-o anumită ordine sortată: fie crescător, fie descrescător.
- Multiset: similar cu set, cu exceptia ca mai multe elemente pot avea aceeasi valoare.
- Map: fiecare element are key value si mapped value, nu pot avea aceeasi key mai multe elemente mapped
- Multimap: similar cu map cu exceptia ca mai multe elemente pot avea aceeasi key.

9.3 STL in C++

6) Containere associative set

SINTAXA

set <data_type> set_name Obs:Trebuie inclusa biblioteca : #include <set>

Ex.. : creare si afisare set crescator

```
#include <iostream>
#include <set>
using namespace std;
int main()
{set<int> a; a.insert(10);a.insert(13);a.insert(11);
for (int i : a) {cout << i << ' ';}
return 0;}
```

```
10 11 13
```

Ex.. : creare si afisare set descrescator

```
#include <iostream>
#include <set>
using namespace std;
int main()
{set<int, greater<int> > a; a.insert(10); a.insert(13); a.insert(11);
for (int i : a) {cout << i << ' ';}
return 0;}
```

```
13 11 10
```

Ex.. : creare si afisare set: elementele cu aceeasi cheie sunt ignorate(ex.: al 2-lea 10)

```
#include <iostream>
#include <set>
using namespace std;
int main()
{set<int> a; a.insert(10); a.insert(13); a.insert(10);
for (int i : a) {cout << i << ' ';}
return 0;}
```

```
10 13
```

9.3 STL in C++

6) Containere associative **multiset**

SINTAXA

multiset <data_type> set_name Obs:Trebuie inclusa biblioteca : #include <set>

Ex.. : creare si afisare set crescator

```
#include <iostream>
#include <set>
using namespace std;
int main()
{multiset<int> a;
a.insert(10); a.insert(13); a.insert(12);
for (int i : a) {cout << i << ' ';}
cout << '\n';
a.insert(10);
cout<<"mai inseram valoarea 10 odata:";
for (int i : a) {cout << i << ' ';}
return 0;
}
```

```
10 12 13
mai inseram valoarea 10 odata:10 10 12 13
```

9.3 STL in C++

6) Containere associative map

SINTAXA

map <key_type, value_type> map_name Obs:Trebuie inclusa biblioteca : #include <map>

Ex.. : creare si afisare map, integer = keys , string = values,
pair= constructor pentru creare pereche key-value,
insert() = metoda de inserare elemente in map

```
#include <iostream>
#include <map>
#include <string>
using namespace std;
int main() {
  map<int, string> a;
  a.insert(pair<int, string>(1, "one"));
  a.insert(pair<int, string>(2, "two"));
  cout << a[1] << " " << a[2] << endl;
return 0;
}
```

```
one two
```

Ex.. : creare si afisare map, integer = keys , string = values, cu initializare {{ }}

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

int main() {
  map<int, string> sample_map { { 1, "one"}, { 2, "two" } };
  cout << sample_map[1] << " " << sample_map[2] << endl;
}
```

```
one two
```

9.3 STL in C++

6) Containere associative **multimap**

SINTAXA

multimap<key_type , value_type> map_name; Obs: #include <map>

Ex.. : creare si afisare map, integer = keys , string = values, pair= constructor pentru creare pereche key-value, insert() = metoda de inserare elemente in map

```
#include <iostream>
#include <map>
using namespace std;
int main ()
{multimap<int, string> m = {{25, "Popa"}, {27, "Rusu"}, {21, "Popescu"}};
cout << "Initial size = " << m.size() << endl;
cout << "\nElemente initiale multimap sortate crescator= " <<endl;
for(auto& p: m)  cout << "{" << p.first << ", " << p.second << "} "; cout<<endl;
m.insert({22, "Corpadea"});
m.insert({20, "Sima"});
m.insert({{23, "Dinu"}, {22, "Zebrean"}});
cout << "\nDupa inserare multimap=" <<endl;
for(auto& p: m)        cout << "{" << p.first << ", " << p.second << "} ";
cout << "\nDupa inserare, size = " << m.size() << endl;
return 0;}
```

```
Initial size = 3

Elemente initiale multimap sortate crescator=
{21, Popescu} {25, Popa} {27, Rusu}

Dupa inserare multimap=
{20, Sima} {21, Popescu} {22, Corpadea} {22, Zebrean} {23, Dinu} {25, Popa} {27, Rusu}
Dupa inserare, size = 7
```

9.3 STL in C++

6) Containere associative multimap

Ex.. : aceeasi problema rezolvata cu make_pair

```
#include <iostream>
#include <map>
using namespace std;
int main ()
{multimap<int, string> m = {{25, "Popa"}, {27, "Rusu"}, {21, "Popescu"}};
cout << "Initial size = " << m.size() << endl;
cout << "\nElemente initiale multimap sortate crescator= " <<endl;
for(auto& p: m)      cout << "{" << p.first << ", " << p.second << "} ";
cout<<endl;
m.insert(make_pair(22, "Corpadea"));
m.insert(make_pair(20, "Sima"));
m.insert(make_pair(23, "Dinu"));
m.insert(make_pair(22, "Zebrean"));
cout << "\nDupa inserare multimap=" <<endl;
for(auto& p: m)     cout << "{" << p.first << ", " << p.second << "} ";
cout << "\nDupa inserare, size = " << m.size() << endl;
return 0;}
```

```
Initial size = 3

Elemente initiale multimap sortate crescator=
{21, Popescu} {25, Popa} {27, Rusu}

Dupa inserare multimap=
{20, Sima} {21, Popescu} {22, Corpadea} {22, Zebrean} {23, Dinu} {25, Popa} {27, Rusu}
Dupa inserare, size = 7
```

9.3 STL in C++

7) Functors

DEFINITIE

Functor (function object) : class sau struct object care poate fi apelat ca o functie.

Ex.. :Suma a 2 numere cu o functie si un functor, operator() supraincarca operatorul apel functie (function call operator ())

Acelasi efect ca o functie dar alt mod de operare

```
#include <iostream>
using namespace std;
int suma (int a, int b) {       return a + b;}
class AddFunctor {
public:
    int operator()(int a, int b) { return a + b;    }
};
int main() {
AddFunctor add; // Create an instance of the AddFunctor
int a,b, rez1,rez2;
cout<<"a="; cin >>a;
cout<<"b=";cin>>b;
rez1 = suma(a, b);
rez2=add(a,b);
cout << "Rez1: " << rez1 <<"Rez2="<< rez2<< endl;
return 0;}
```

```
a=10
b=20
Rez1: 30 Rez2: 30
```

9.3 STL in C++

7) Functors

DEFINITIE

Functor (function object) : **C++ Lambda expression (anonymous function objects)**

Ex.. :Creare functie Lambda care afiseaza un text

```
#include <iostream>
using namespace std;
int main() {
// create a lambda function that prints "Hello World!"
auto greet = [ ]()
   {cout << "Hello World!";    };
// [ ] is called the lambda introducer which denotes the
//start of the lambda expression
// () parameter list
greet(); // call lambda function
return 0;}
```

```
Hello World!
```

Ex.. :Creare functie Lambda care insumeaza 2 nr. intregi

```
include <iostream>
using namespace std;
int main() {
auto add = [] (int a, int b) {
      cout << "Sum = " << a + b;
      };
add(100, 78); // call lambda function
return 0;}
```

```
Sum = 178
```

Echivalent cu :

```
void add(int a, int b) {
  cout << "Sum = " << a + b;
   }
```

7) Functors

DEFINITIE

Functor (function object) : **C++ Lambda expression (anonymous function objects)**

Ex.. : Creare functie Lambda care aduna 50 la o valoare initiala si=100

```
#include<iostream>
using namespace std;
int main() {
int si = 100;
  // capture si by value
auto add_to_sum = [si] (int num) {
  return si + num;
  };
int sf = add_to_sum(50);
cout << "100 + 50 = " << sf;
return 0;}
```

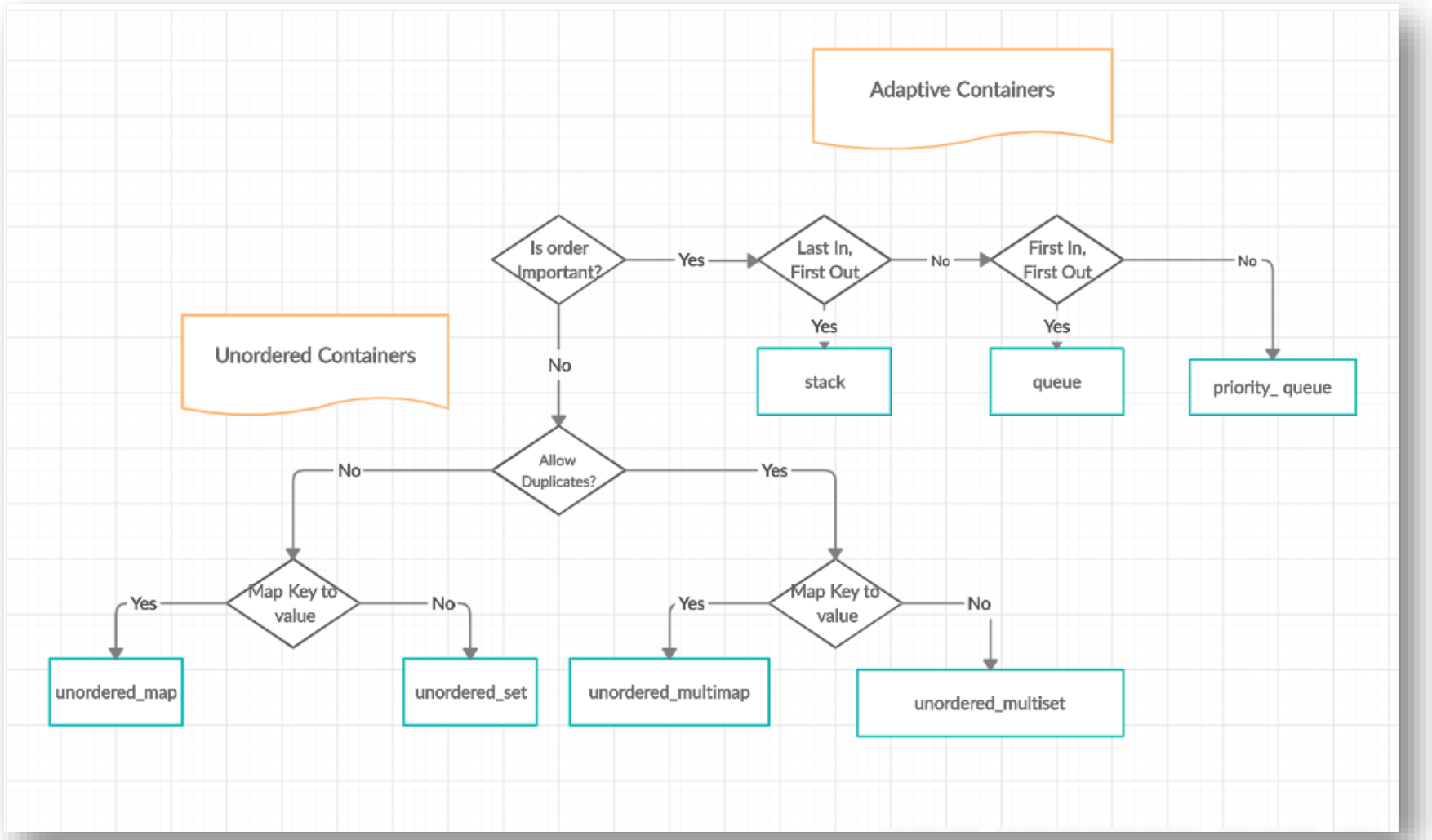
Ex.. : Creare functie Lambda care determina daca nr sunt pare

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int main() {
vector<int> a = {1, 2, 3, 4, 5, 8, 10, 12};
cout<<"afisare vector:";
for (int i : a) {
    cout << i << " ";  }
int pare = count_if(a.begin(), a.end(), [](int num) {return num % 2 == 0;  });
cout << "\nSunt: " << pare << " numere pare.";
return 0;}
```

Functia **count_if()** returneaza numarul de elemente dintr-un interval (a.begin(),a.end(), conditie) pentru care conditie =true.

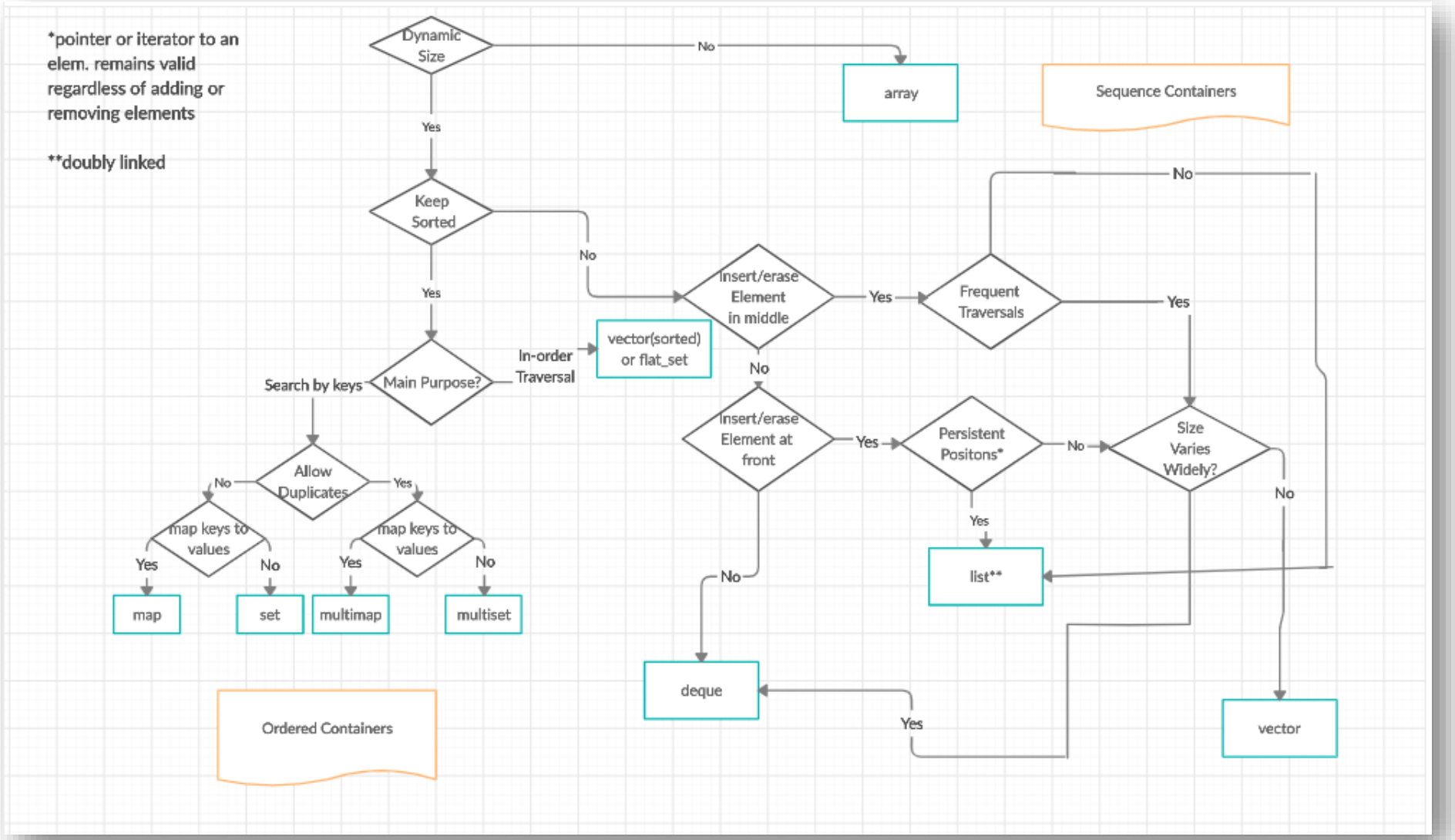
9.3 STL in C++

Containere in C++



9.3 STL in C++

Containere in C++



TEST kahoot

Pentru login, introduceti codul afisat pe ecran, in browser la adresa:

http://kahoot.it