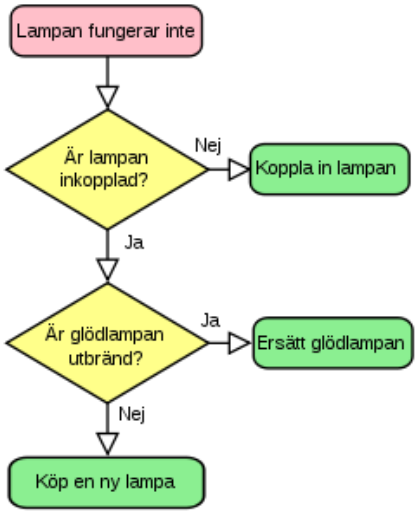
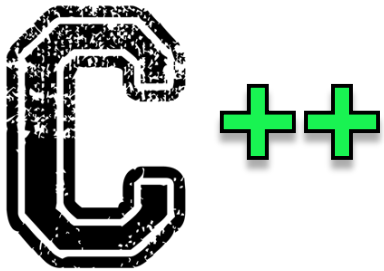



```cpp
#include <iostream>
#include <cmath>
using namespace std;

int main(){
      double x, y, a, b;
      cout << "Unesi x, y " ;
      cin >> x >> y;

      a = x * y;
      b = a / x;
      cout << "a = "<<a<<" b ="<< b <<endl;

      system("PAUSE");
      return 0;
}
```



#include<stdio.h>

STRUKTURE PODATAKA I ALGORITMI

Dejan Sredojević

Konsultacije: četvrtak, kabinet 12, 15:00 – 17:00

e-mail: dsredojevic.vps@gmail.com

VISOKA POSLOVNA ŠKOLA STRUKOVNIH STUDIJA NOVI SAD

Vrste struktura podataka

- Kolekcije i skupovi
- Linearne strukture
- Stabla
- Grafovi i Mreže

- Link: ***http://www.cplusplus.com/reference/stl/***

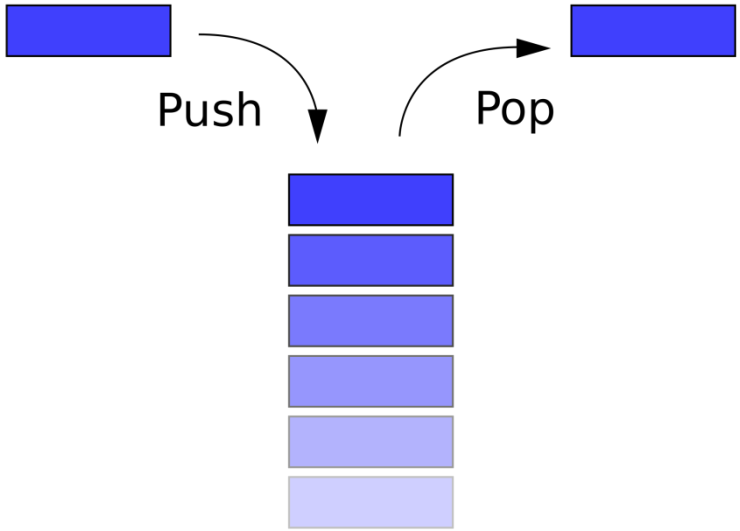
Linearne strukture

- Stek (Stack)
- Red (Queue)
- Dvostruki red (Deque)
- Lista (List)

Stek (Stack)

- Ubaci i izbaci element sa istog kraja

  LIFO – Last In First Out



Stek (Stack)

- Da bi mogao da se koristi stek kao struktura podataka u STL-u, onda se na početku programa mora uključiti direktiva

```
#include <stack>
```

- Stek sadrži elemente određenog tipa. Ako želimo da deklarišemo stek koji će sadržati stringove kao podatke, deklaraciju pišemo:

```
stack<string> s;
```

- Naravno, kad god se koristi string kao tip podataka, mora se uključiti direktiva

```
#include <string>
```

- Ako želimo da deklarišemo stek koji će sadržati cele brojeve kao podatke, deklaraciju pišemo:

```
stack<int> s;
```

- ili uopšte za proizvoljni tip podataka (primitivni tip podataka, klasa) T

```
stack<T> s;
```

Stek (Stack)

Dozvoljeni metodi

- Za proveru da li je stek prazan koristi se metodata empty, tj.

```
s.empty() //->bool
```

- Ova metoda vraća true ako je stek prazan ili false uslučaju da stek ima barem jedan element.
- Za proveru koliko ukupno elemenata se nalazi u steku se koristi metoda size

```
s.size() //->int
```

- Za unos novog elementa t tipa T na vrh steka se koristi metoda push, na primer

```
s.push(t) //->void
```

- Za čitanje i skidanje elementa sa vrh steka se koristi metoda pop, na primer

```
s.pop() //->T
```

- Ako želimo samo da pročitamo element na vrhu steka, ali ne i da ga odstranimo sa vrha, onda se koristi metod top

```
s.top() //->T
```

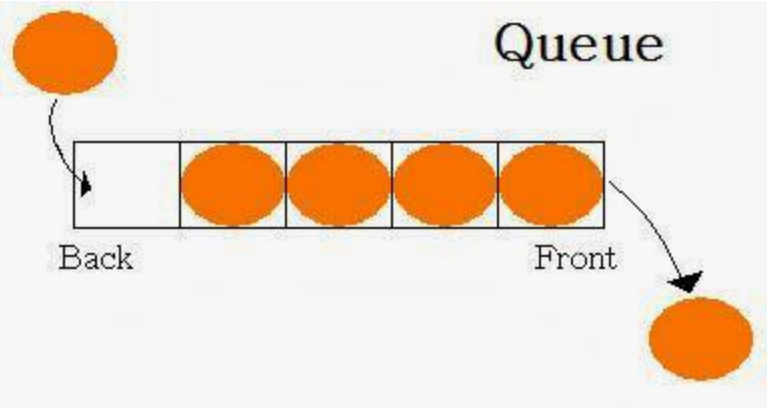
Primer - Stek

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

int main() {
     //deklarisemo stek strukturu sa elementima tipa string
     stack<string> s;

     //Unesimo tri stringa i sacuvajmo u steku
     s.push("STRING 1");
     s.push("STRING 2");
     s.push("Str 3");

     cout<<"Unesi element steka: "<<endl;
     string w;
     cin>>w;
     s.push(w);

     //Stampamo broj clanova steka koji smo formirali
     cout << "Broj elemenata steka = " << s.size() << endl;

     //Proverimo da li stek nije prazen
     while (!s.empty()) {
          cout << s.top() << endl; //Stampanje sadrzaja sa vrha steka
          s.pop(); //Iscitavanje vrha steka i skidanje elementa sa vrha steka
     }

     return 0;
}
```

Red (Queue)

- Ubaci element na jednom a izbaci sa drugog kraja

  FIFO – First In First Out



Red (Queue)

- Da bi se koristila struktura reda mora se uključiti direktiva queue iz STL-a:

```
#include <queue>
```

- Ako želimo da deklarišemo red za elemente tipa T, onda deklaracija glasi:

```
queue<T> q;
```

Red (Queue)

Dozvoljeni metodi

- Za proveru da li je red prazan, koristi se metoda empty

```
q.empty() //->bool
```

- Da bi se doznalo koliko ukupno elemenata je uneseno u red koristi se metod size

```
q.size() //-> int
```

- Da bi se postavio element t tipa T na kraj reda q, koristi se

```
q.push(t) //-> void
```

- Za čitanje i uklanjanje elementa sa početka reda q koristi se metod pop, kao u navedenom slučaju

```
q.pop() //->T
```

- Za čitanje, ali ne i uklanjanje poslednjeg elementa u redu koristi se metod back

```
q.back() //-> T
```

- Za čitanje, ali ne i uklanjanje prvog elementa (prednji deo reda) koristi se metoda front

```
q.front() //-> T
```

Primer - Red

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

int main() {
     queue<string> q;  //deklarisanje reda ciji elementi su tipa string
     //Unos tri stringa u red
     q.push("PROBA 1");
     q.push("PROBA 2");
     q.push("PROBA 3");

     string w;
     cin>>w;
     q.push(w);

     //Stampanje broja clanova reda
     cout << "Broj elemenata reda = " << q.size() << endl;

     //Dokle god red nije prazan
     while (!q.empty()) {
          cout << q.front() << endl;  //Citanje stringa sa vrha reda i stampanje,
FIFO
          q.pop();  //Citanje i uklanjanje stringa sa vrha reda
     }

     return 0;
}
```

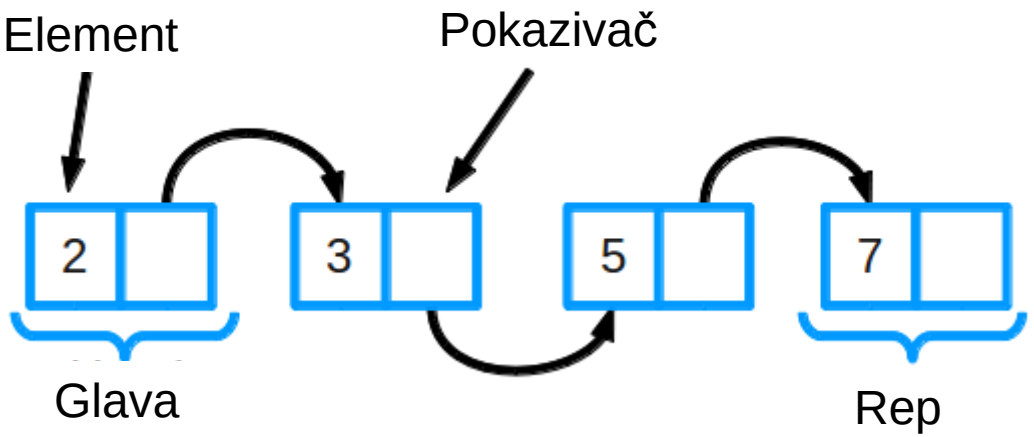
Lista (List)

- Da bi se koristile liste u STL-u kao strukture podataka, mora da se doda sledeća direktiva na početku programa:

```
#include <list>
```

- Da bi se deklarisala lista određenog tipa T koristimo sledeću liniju programa:

```
list<T> q;
```



Lista (List)

Dozvoljeni metodi

- Da bi se proverilo da li je lista prazna, koristi se metoda empty, kao na primer

```
ls.empty() //-> bool
```

- Da bi se saznao tekući broj elementi u listi, koristi se metod size

```
ls.size() //-> int
```

- Za dodavanje elementa t tipa T na kraj liste ls, koristi se naredba

```
ls.push_back(t) //-> void
```

- Za čitanje i uklanjanje elementa sa kraja liste ls, koristi se metod pop_back

```
ls.pop_back() //-> T
```

- Analogno, postoji sličan metod za dodavanje i uklanjanje elementa na početak liste ls

```
ls.push_front(t) //-> void
ls.pop_front() //-> T
```

- Za sortiranje liste se koristi metod sort, kao u sledećem primeru:

```
ls.sort() //-> void
```

- Za brisanje svih elemenata liste se koristi metod clear

```
ls.clear() //-> void
```

- Za “obrtanje“ liste se koristi metod reverse

```
ls.reverse() //-> void
```

- Za dodeljivanje kopije liste ls1 drugoj listi ls2, se koristi naredba

```
ls2=ls1;
```

Primer 1 - Lista

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

int main() {
     list<int> l;
     l.push_back(3);
     l.push_front(1);
     l.insert(++l.begin(),2);
     list<int>::iterator it;

     for(it=l.begin(); it!=l.end();
it++)
          cout << * it << ' ';

     return 0;
}
```

Primer 2 - Lista

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

void ispis (list<int> &li) {
     for(list<int>::iterator

     x=li.begin();x!=li.end();x++)
          cout << *x << ' ';

     cout << endl;
}

int main() {
     list<int> l;
     l.push_back(3); l.push_back(2);
     l.push_back(4); l.push_back(1);
     ispis(l);
     l.reverse();
     ispis(l);
     l.sort();
     ispis(l);
     l.reverse();
     ispis(l);

     return 0;
}
```

Vektor (Vector)

- Vektori se razlikuju od liste zato što postoji mogućnost za direktan pristup elementu vektora preko indeksa, t.j.

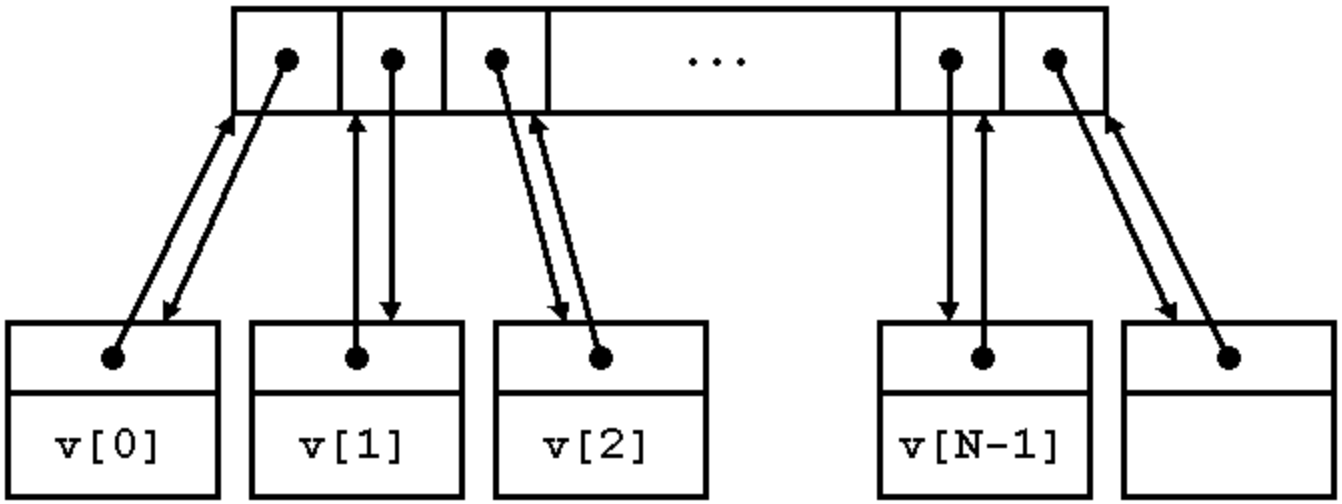
  operatora []

- Da bi mogla da se koristi ova struktura potrebno je u programu uključiti direktivu:

  #include <vector>

- Da bi deklarisali vektor određenog tipa T , naredba je:

  vector<T> v;



Vektor (Vector)

Dozvoljeni metodi

- Za proveru da li je vektor v prazan, koristi se metod empty

```
v.empty() //-> bool
```

- Da biste saznali trenutni broj elemenata unutar vektora v koristi se metod size

```
v.size() //-> int
```

- Da biste dodali novi element t tipa T na kraj vektora v, koristi se metod push_back

```
v.push_back(t) //-> void
```

- Ako želite da pročitate i uklonite element sa kraja vektora v, koristi se metod pop_back

```
v.pop_back() //-> T
```

- Za pristup (konstantna referenca) prvom elementu na početku vektora, koristi se metod front

```
v.front() //-> T
```

- Za pristup (konstantna referenca) poslednjem elementu na kraju vektora v koristi se metod back

```
v.back() //-> T
```

- Za direktan pristup i-tom elementu vektora v (0<= i<v.size()) bez provere da li element odnosno indeks postoji, koristi se baš isti način kao i kod običnog niza:

```
v[i]
```

- Za siguran pristup i-tom elementu koristi se metod at, kao što je pokazano u primeru

```
v.at(i)// -> referenca
```

- Za dodeljivanje kopije vektora v1 vektoru v2 koristi se naredba dodele

```
v2=v1;
```

Primer 1 - Vektor

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

int main(){

     vector<int> v(4);   //Inicijalizujmo vektor sa 4-ri celobrojne vrednosti
     //Inicijalizujmo vrednosti
     v[0]=100;
     v[1]=200;
     v[2]=300;
     v[3]=400;

     //metod push_back pravi proširivanje vektora za element
     //koji je argument metoda
     v.push_back(10);
     v.push_back(20);
     v.push_back(30);
     v.push_back(40);
     v.push_back(50);

     for(int i=0;i<v.size();i++)
          cout<<i<<" - ti element: "<<v[i]<<endl;

     return 0;
}
```

Primer 2 - Vektor

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

int main() {
     vector <int> v;
     v.push_back(3); v.push_back(5);
     v.push_back(7);

     for(int i=10;i<14;i++)
          v.push_back(i);

     v[0]--;
     v[5]=-200;

     for(unsigned i=0;i<v.size();i++)
          cout << v[i] << ' ';

     cout << endl;

     return 0;
}
```

Primer 3 - Vektor

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

int main() {
	vector <string> vs;
	vs.push_back("ja");
	vs.push_back("volim");
	vs.push_back("programiranje");

	for(unsigned i=0;i<vs.size();i++)
		cout << vs[i] << ' ';

	cout << endl;
	vector < vector<int> > matrica;

	for(int i=0;i<5;i++){
		vector <int> v;
		matrica.push_back(v);

		for (int j=0;j<=i;j++)
			matrica[i].push_back(j);
	}

	for(unsigned i=0;i<matrica.size();i++) {
		for(unsigned j=0;j<matrica[i].size();j++)
			cout << matrica[i][j] << ' ';

		cout << endl;
	}

	return 0;
}
```