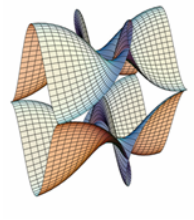

Prirodoslovno-matematički fakultet
Matematički odsjek
Sveučilište u Zagrebu

RAČUNARSKI PRAKTIKUM I

Vježbe 05 - Standard Template Library (1)

v2018/2019.

Sastavio: Zvonimir Bujanović

Zadatak 1

Implementirajte generičku strukturu **`stack`** koja će se moći koristiti ovako:

```
stack<int> stogIntova;
stack<string> stogStringova;

stogIntova.push( 3 );
stogStringova.push( "abc" );

int a = stogIntova.top();
string b = stogStringova.top();

stogIntova.pop();
```

Standard Template Library

Na sličan bismo kao generičke strukture mogli implementirati i ostale ATP-ove iz kolegija Strukture podataka i algoritmi:

- `List`, `Stack`, `Queue`, `Set`, `Mapping`, ...

Standardna biblioteka u C++-u već dolazi s implementacijom svih ovih (i brojnih drugih) ATP-ova!

To čini tzv. Standard Template Library (STL).
Dokumentacija: http://www.cplusplus.com/reference/stl/

Implementacija u STL-u je često efikasnija od one koja se radi na SPA.

stack, queue

Struktura `stack` u STL-u je gotovo identična kao u Zadatku 1.

```cpp
#include <stack>
...

stack<int> S;

S.push( 3 );
S.push( 5 );

int a = S.top();
S.pop();

if( S.empty() )
{ ... }

int zz = S.size();
```

```cpp
#include <queue>
...

queue<string> Q;

Q.push( "abc" );
Q.push( "xy" );

string a = Q.front();
Q.pop();

if( Q.empty() )
{ ... }

int zz = Q.size();
```

vector

`vector` je generička struktura čija uloga je zamijeniti polja iz C-a.

Najvažnija prednost u odnosu na obična polja je automatska alokacija/oslobađanje memorije prilikom dodavanja/brisanja elemenata.

```
#include <vector>
```

vector

Konstruktori.

- Vektor veličine 0.

```
vector<int> A;
```

- Vektor veličine N, za svaki element se pozove defaultni konstruktor.

```
int N = 100;
vector<int> A( N );     // A[0]=...=A[99]=0
vector<tocka> S( 23 ); // S[0]=...=S[22]=tocka()
```

- Vektor veličine N, svaki element se postavi na vrijednost x.

```
int N = 2, x = 5;
vector<int> A( N, x );          // A[0]=A[1]=5
vector<string> S( 10, "abc" ); // S[0]=...=S[9]="abc"
```

vector

Pristup pojedinim elementima: kao u polju.

```
vector<string> S( 3 );  // ["", "", ""]

S[1] = string( "abc" ); // ["", "abc", ""]

cout << S[1]; // ispise "abc"
cin >> S[2];
```

vector

Broj elemenata vektora.

```
vector<string> S( 10 );
int n = S.size(); // n = 10;
```

Promjena broja elemenata vektora.

```
vector<tocka> S(10); // S ima 10 elem.
S.resize( 20 ); // sad S ima 20 elem.
S.resize( 5 ); // sad S ima 5 elem. (zadnjih 15 se brise)
```

Ubacivanje/izbacivanje s kraja.
(Automatski alocira/oslobađa memoriju ako treba.)

```
vector<int> S( 2 ); // [0, 0]
S.push_back( 3 );   // [0, 0, 3]
S.pop_back();       // [0, 0]
```

Brisanje svih elemenata.

```
vector<int> S( 10 ); // S=[0,0,...,0]
S.clear();           // S=[] (prazan vektor)
```

Kopiranje vektora.

```
vector<int> S( 10, 3 ), A( 5 );
A = S; // A=[3,3,3,3,3,3,3,3,3,3]
```

Testiranje jednakosti.

```
vector<int> S, T;
S.push_back( 3 ); S.push_back( 1 ); // S=[3,1]
T.push_back( 3 ); T.push_back( 5 ); // T=[3,5]
if( S == T )
    cout << "Isti su";
else
    cout << "Nisu isti"; // ovo se ispise
```

Leksikografsko uspoređivanje.

```
vector<int> S, T;
S.push_back( 3 ); S.push_back( 1 ); // S=[3,1]
T.push_back( 3 ); T.push_back( 5 ); // T=[3,5]
if( S < T )
    cout << "S je manji od T"; // ovo se ispise
else
    cout << "S nije manji od T";
```

list

`list` odgovara ATP-u `List` iz SPA.

Velik broj funkcija je isti kao kod `vector`-a. Bitne razlike:

- Ne možemo indexirati elemente.
- Kretanje po listi sa `iterator`-om (usporedi s `position` iz SPA).

```
#include <list>
```

list

Konstruktori - kao kod `vector`-a.

```
list<int> A, B( 10 ), C( 20, 3 );
```

=, ==, <, `clear`, `resize` - kao kod `vector`-a.

```
list<string> A( 2, "ab" ), B( 3, "ab" );
// A=["ab", "ab"], B=["ab", "ab", "ab"]

if( A == B )
    cout << "isti"; // ne ispise se

if( A < B )
    cout << "A je manji od B"; // ispise se

B.clear(); // B=[]

A.resize( 5 ); // A=["ab", "ab", "", "", ""]
```

list

Ubacivanje/izbacivanje na početak/kraj.

```
list<int> L;          // L=[]

L.push_back( 5 );     // L=[5]
L.push_front( 7 );    // L=[7, 5]
L.push_back( 3 );     // L=[7, 5, 3]

int a = L.front();    // a = 7
int b = L.back();     // b = 3

L.pop_front();        // L=[5, 3]
L.pop_back();         // L=[5]
```

Kod `vector`-a ne postoje funkcije za ubacivanje/izbacivanje na početak. Zašto?

list

Indexiranje nije dozvoljeno.

```
list<string> L( 5, "abc" );
string s = L[3]; // !!! compile error !!!
```

Pristup unutarnjim elementima i kretanje kroz listu:
podstruktura `list::iterator`.

```
list<string> L( 3, "abc" ); // L=["abc", "abc", "abc"]
list<string>::iterator li;

li = L.begin(); // "first" - li="pointer" na 1.element
++li;           // "next"  - li="pointer" na 2.element

cout << *li; // "retrieve" - ispiše "abc"
*li = "xy";  // L=["abc", "xy", "abc"]

--li; // "previous" - li="pointer" na 1.element
```

list

U SPA postoji "header" ćelija u implementaciji ATP `List`.
Ovdje zamišljamo da postoji ćelija iza zadnjeg elementa: `L.end()`.
U toj ćeliji se ne nalazi element liste!

```
list<int> L( 3 ); // L=[0, 0, 0]

list<int>::iterator li;

// li="pointer" na prvi element liste
li = L.begin(); *li = 5; // OK: L=[5, 0, 0]

// li="pointer" na ćeliju iza zadnjeg elementa liste.
li = L.end();

// li="pointer" na zadnji element liste.
--li; *li = 8; // OK: L=[5, 0, 8]

// li="pointer" na ćeliju iza zadnjeg elementa liste.
li = L.end(); *li = 7; // NIJE OK (program se može srušiti)
```

list

Prolazak kroz listu i ispis elemenata.

```
list<string> L( 5, "abc" );

list<string>::iterator li;
for( li = L.begin(); li != L.end(); ++li )
{
    string element = *li;
    cout << element << " ";
}
```

Iteratore ne možemo uspoređivati sa <, nego samo sa == ili !=.

Brisanje elemenata.

```
list<int> L; ... // napuni nekako listu L
list<int>::iterator li, ltemp;

li = L.begin();
while( li != L.end() ) {
    if( *li == 5 ) { // brisemo sve elemente 5 iz liste
        ltemp = li; ++ltemp;
        L.erase( li );
        li = ltemp;
    }
    else
        ++li;
}
```

Funkcija `erase` uništi iterator `li` kojeg briše i nakon toga više nije dozvoljeno raditi `li++`.
Zbog toga prije brisanja u `ltemp` zapamtimo koja će biti pozicija iteratora nakon brisanja.

Brisanje elemenata.

- Funkcionalnost s prethodne stranice postižemo i ovim kodom:

```
list<int> L; ... // napuni nekako listu L
list<int>::iterator li, ltemp;

li = L.begin();
while( li != L.end() )
{
    if( *li == 5 ) // brisemo sve elemente 5 iz liste
        L.erase( li++ );
    else
        ++li;
}
```

Obrazložite! Uočite da bi bilo neispravno napisati `L.erase( ++li );`

Brisanje više elemenata.

- Poziv `L.erase(st, en)` briše sve elemente čiji iteratori su u intervalu $[$`st`, `en`$\rangle$.

```
list<tocka> L; ... // napuni L nekako
list<tocka>::iterator st, en;

st = L.begin(); ++st;
en = st; ++en; ++en; ++en; ++en;

// st pokazuje na 2., a en na 6. element liste
L.erase( st, en ); // brisemo 2., 3., 4., 5. element
```

Konvencija u STL-u je da funkcije koje primaju dva iteratora `st`, `en` rade na poluotvorenom intervalu $[$`st`, `en`$\rangle$.

Ubacivanje novih elemenata.

- `L.insert(li, x)` ubacuje novi element `x` ispred iteratora `li`.
- Ubacivanje ne uništi i ne promijeni iterator `li`.

```
list<string> L; ... // napuni L nekako
list<string>::iterator li;

// Ispred svakog "ABC" ubacimo "XY".
for( li = L.begin(); li != L.end(); ++li )
    if( *li == string( "ABC" ) )
        L.insert( li, "XY" );
```

Zadatak 2

1. Napišite program koji učitava i stavlja na kraj liste stringove sve dok se ne učita string `kraj`.
2. Ispred svakog stringa `S` u listi dodajte još onoliko čvorova koliko `S` ima slova. U svaki od tih čvorova upišite po jedno slovo od `S`.

   Na primjer, ako je učitana lista bila `[RP1,Jupi]`, onda rezultantna lista treba biti `[R,P,1,RP1,J,u,p,i,Jupi]`.
3. Na kraju premjestite iz liste u vektor sve stringove duljine veće od 1.

Za vježbu riješite isti zadatak, ali tako da nove čvorove dodavate iza učitanih (a ne ispred).