Standardna biblioteka - STL

Standardna biblioteka šablona

- STL – biblioteka šablona generičkih funkcija i klasa
- Efikasna realizacija često korišćenih struktura podataka i algoritama nezavisno od tipova podataka
- Svi identifikatori u STL se nalaze u std namespace-u
- Grupe šablona u STL
  - Algoritmi (npr. sort. find, merge, …)
  - Kontejnerske klase (vektor, lista, magacin, red, skup, mape, string, niz bitova)
  - Jednostavne strukture podataka (par, kompleksan broj)
  - Uslužne funkcije i klase (relacioni operatori, funkcijske klase, iteratori)

Standardna biblioteka STL

STL\Standard C++ Library Overview.htm

Klase opste namene
<utility>
<functional>
<memory>
<ctime>

Iterator
<iterator>

Algoritmi
<algorithm>
<cstdlib>

Dijagnostika
<exception>
<stdexcept>
<cassert>
<cerrno>

Stringovi
<string>
<cctype>
<cwctype>
<cstring>
<cwchar>
<cstdlib>

Ulaz/Izlaz
<iostream>
<ios>
<streambuf>
…

Lokalizacija
<locale>
<clocale>

Support
<limits>
<climits>
<cfloat>
<new>
<typeinfo>
<ctime>
<csignal>
….

Numericki
<complex>
<valarray>
<numeric>
<cmath>

Algoritmi

- Uključiti `<algorithm>`
- Vrste realizovanih algoritama
  - Nad pojedinačnim podacima (`min, max, swap`)
  - Obrada pojedinačnih elemenata kolekcija (`for_each, fill, generate,` ...)
  - Pretraživanje kolekcija (`find, min/max_element, count, search,` ...)
  - Obrade kolekcija (`copy, copy_backward, replace, replace_if, remove, remove_if, remove_copy, unique, reverse, reverse_copy, equal, lexicographical_compare, next_permutation, prev_permutation`, ...)
  - Obrada uređenih sekvencijalnih kolekcija (`sort, merge, binary_search`)

Standardna biblioteka STL

using namespace std

Kontejnerske klase

- Sequences
  - C++ Vectors
  - C++ Lists
  - C++ Double-Ended Queues
- Container Adapters
  - C++ Stacks
  - C++ Queues
  - C++ Priority Queues
- Associative Containers
  - C++ Bitsets
  - C++ Maps
  - C++ Multimaps
  - C++ Sets
  - C++ Multisets

Vector (1)

• Kontejnerska klasa: efikasan pristup proizvoljnom element; efikasno dodavanje na kraj i izbacivanje sa kraja; neefikasno dodavanje i izbacivanje sa početka i u sredini; automatsko podešavanje kapaciteta

• definisana u std namespace

• **#include <vector>**

```
• template<class T, class A = allocator<T> > class std::vector {
        public:
            typedef T value_type;
            typedef A allocator_type;
            typedef typename A::syze_type size_type;
            typedef typename A::difference_type difference_type;

            typedef implementation_dependent1 iterator;
            typedef implementation_dependent1 const_iterator;
            typedef std::reverse_iterator<iterator> reverse_iterator;
            typedef std::reverse_iterator<const iterator> const_reverse_iterator;
…
}
```

vector.h - STL\vectorHeader.txt

Vector (2)

- Klasa `vector` obuhvata
  - Konstruktore
    - Podrazumevani
    - Konstruktor za kopiranje
    - Konstruktor za premeštanje
    - `template<class It> vector(It prvi,It posl);` - kopira elemente neke kolekcije iz intervala `[prvi, posl)`
  - Destruktor
  - `operator=`
  - Veliki broj metoda za manipulaciju sa vektorom

Primer

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

 vector<int> intVector;
 vector<int>::iterator vIterator;

  for( int i=0; i < 10; i++ )
    intVector.push_back(i);

  int total = 0;
  vIterator = intVector.begin();

  while( vIterator != intVector.end() ) {
    total += *vIterator;
    vIterator++;
  }
  cout << "Total=" << total << endl;
```

metode klase STL\C++ Vectors.htm
operatori klase STL\STL Vector Class.htm
back() – opis STL\back.htm

Iteratori

Klase čiji objekti predstavljaju pozicije elemenata u različitim kontejnerskim klasama

- Iteratori se uvek pridružuju nekoj kontejnerskoj klasi
- Tip zavisi od kontejnerske klase kojoj su pridruženi

```
#include <list>
#include <vector>
using namespace std;
...
list<int> nums;
list<int>::iterator nums_iter;
vector<double> values;
vector<double>::iterator values_iter;
vector<double>::const_iterator const_values_iter;
```

Tip Iteratora	Osobina
• Forward	Može da specificira poziciju jednog elementa u kontejneru.
	Može da menja vrednost od elementa do elementa u jednom smeru.
• Bidirectional	Isto kao i Forward ali može da menja vrednost u oba smera
• Random access	Može da se menja u oba smera u krupnijim koracima

Iterator:

- Konstantni - samo pregled sadržaja bez promene
- Nekonstantni – i promena sadržaja je omogućena

Operatori iteratora: STL\STL Iterator Classes.htm

Liste

- `#include <list>`
- `template <typename T> class list`
- Za razliku od `vector` moguće je efikasno umetanje i izbacivanje elemenata sa bilo koje pozicije
- Efikasan sekvencijalni pristup elementima
- Neefikasan pristup po proizvoljnom redosledu
- Koriste dvosmerne iteratore za kretanje po listi

```
#include <list>              // list class library
using namespace std;
...
  list<int> nums;
  list<int>::iterator nums_iter;
  nums.push_back (0);
  nums.push_back (4);
  nums.push_front (7);

  cout << endl << "List 'nums' now becomes:" << endl;
  for(nums_iter=nums.begin();nums_iter!=nums.end();nums_iter++)
  {
    *nums_iter = *nums_iter + 3; // Modifikuje svaki element.
    cout << (*nums_iter) << endl;
  }
  cout << endl;
```

```
#include <list>     // list class library
#include <string> // string class library
using namespace std;
...
  list<string> words;
  list<string>::iterator words_iter;
...
  unsigned int total_length = 0;
  for (words_iter=words.begin(); words_iter != words.end();
       words_iter++)
  {
    total_length += (*words_iter).length();  // correct
//  total_length += *words_iter.length();    // incorrect !!
  }
  cout << "Total length is " << total_length << endl;
```

```
#include <vector>       // vector class library
#include <list>          // list class library
#include <algorithm>              // STL algorithms class library
using namespace std;
...
  vector<string> vec1;

  string state1 = "Wisconsin";
  string state2 = "Minnesota";
  vec1.push_back (state1);
  vec1.push_back (state2);
  vec1.push_back ("Illinois");
  vec1.push_back ("Michigan");
  sort(vec1.begin(),vec1.end()); // Sortiranje vektora stringova.

  vec1.erase(vec1.begin(),vec1.end());
...
  list<int> nums;
  list<int>::iterator nums_iter;
...
  nums.erase (nums.begin(), nums.end()); // Brisanje svih elemenata.
...
  nums_iter = find(nums.begin(), nums.end(), 3); // Traženje u listi.
  if (nums_iter != nums.end())
  {
    cout << "Number " << (*nums_iter) << " found." << endl;
  }
...
  // Ako je pronađen element briše se iz liste.
  if (nums_iter != nums.end()) nums.erase(nums_iter);
```

List - metode

Liste su sekvence elemenata smeštenih u povezane liste

Container constructors	create lists and initialize them with some data
Container operators	assign and compare lists
assign	assign elements to a list
back	returns a reference to last element of a list
begin	returns an iterator to the beginning of the list
clear	removes all elements from the list
empty	true if the list has no elements
end	returns an iterator just past the last element of a list
erase	removes elements from a list
front	returns a reference to the first element of a list
insert	inserts elements into the list
max_size	returns the maximum number of elements that the list can hold
merge	merge two lists

List - metode

pop_back	removes the last element of a list
pop_front	removes the first element of the list
push_back	add an element to the end of the list
push_front	add an element to the front of the list
rbegin	returns a reverse_iterator to the end of the list
remove	removes elements from a list
remove_if	removes elements conditionally
rend	returns a reverse_iterator to the beginning of the list
resize	change the size of the list
reverse	reverse the list
size	returns the number of items in the list
sort	sorts a list into ascending order
splice	merge two lists in constant time
swap	swap the contents of this list with another
unique	removes consecutive duplicate elements

Algoritmi – sort, find

- u <algorithm> STL\C++ Algorithms.htm

Primer za sort()

```
#include <vector>
#include <algorithm>  // Include algorithms
using namespace std;

vector<int> vec;
vec.push_back (10);
vec.push_back (3);
vec.push_back (7);

sort(vec.begin(), vec.end()); // Sort the vector

// The vector now contains: 3, 7, 10
```

Primer: find( )

```
#include <vector>  // vector class library
#include <list>           // list class library
#include <algorithm>      // STL algorithms class library
using namespace std;
...
list<int> nums;
list<int>::iterator nums_iter;

nums.push_back (3);
nums.push_back (7);
nums.push_front (10);

nums_iter = find(nums.begin(), nums.end(), 3); // Search the list.
if (nums_iter != nums.end()) {
    cout << "Number " << (*nums_iter) << " found." << endl; // 3
}
else {
    cout << "Number not found." << endl;
}

// If we found the element, erase it from the list.
if (nums_iter != nums.end()) nums.erase(nums_iter);
// List now contains: 10 7
```

String

- #include <string>

- STL\ANSI String Class.htm

Namespace –prostori imena

Potencijalni konflikti sa imenima

- Često se pojavljuju isti identifikatori u različitim fajlovima jednog te istog projekta
- Povezivanje ovih istih identifikatora
  - **Spoljašnje** – identifikatori koji predstavljaju isti entitet u projektu
  - **Unutrašnje** – identifikatori koji su isti ali označavaju različite entitete
- C++ i C – globalni identifikatori imaju spoljašnje povezivanje (ako se korisit static onda je unutrašnje) dok lokalni indentifikatori imaju unutrašnje povezivanje (ako se korisit extern tada je spoljašnje)
- **Nedozvoljena situacija** – 2 globalna identifikatora sa spoljašnjim povezivanjem u 2 fajla !
- **Rešenje** – namespace (prostor imena)

Namespace

```
// program za demonstraciju potrebe za namespace
int main()
{
  int value;
  value = 0;
  double value; // Greška!!!
  value = 0.0;
}
```

```
Compiler Error:
'value' has a previous declaration
 as 'int value'
```

```
// Promenljiva kreirana unutar namespace
namespace first
{
  int prom = 500;
}

int prom = 100; // Globalna promenljiva


int main()
{
  int prom = 200; // Lokalna promenljiva

  // promenljivoj može da se pristupi van namespace
  // pomoću scope operatora ::
  cout << first::prom << '\n';

  return 0;
}
```

Definisanje namespace

- `namespace ImeNS { sadržaj }`
- Korišćenje sadržaja prostora imena:
  - `using namespace ImeNS // nadalje`
  - `using ImeNS::identifikator // nadalje`
  - `ImeNS::identifikator  // samo na tom mestu`
- Više definicija prostora imena sa istim imenom obrazuju jedan prostor imena
- Jedinstven prostor imena može biti u više datoteka
- Identifikatori iz prostora imena se mogu koristiti samo nakon definicije istih u prostoru imena
- Prostor imena ne menja način povezivanja globalnih imena niti životni vek globalnih podataka

Namespace

```
namespace X {
        int i,j,k;
}

int k;

void f1()
{
        int i=0;
        using namespace X;
        i++;    //lokalno i
        j++;    //X::j
        k++;    //greska X::k  ili globalno k?
        ::k++; //globalno k
        X::k++;         //k iz X
}
```

```
void  f2(){

int i=0;

using X::i;   //greska dva puta
deklarisano

using X::j;

using X::k;  //skriva globalno k

i++;

j++;

k++;}
```

Moguće je:

- Definisati više namespace
- U okviru jednog namespace definisati više drugih
- Deklarisati ili definisati klasu unutar namesapce
- …

Anonimni prostor imena

- Anonimni prostor imena – prostor imena bez imena
  - Globalni identifikatori sa unutrašnjim povezivanjem
  - Identifikatori iz anonimnog prostora imena
    - Imaju doseg na nivou datoteke
    - Moraju se razlikovati od globalnih identifikatora
    - Mogu se koristiti bez navođenja rezolucionog operatroa ::
- Isti naziv identifikatora iz anonimnog prostroa imena u dve datoteke označava dva različita entiteta
- Preporuka:
  - Koristiti anonimni prostro imena umesto proglašavanja globalnih imena za `static`

Anonimni prostor imena - primer

```
namespace {
  int a=1;
}

int b=2;

namespace A{
    int a=3;
    int b=4;
}
```

```
void f(){
  int o=a;   //::a (iz anonimnog prostora)
  int p=b;   //::b (globalna promenljiva)
  int q=A::a;
  using namespace A;
  int r=a;   //GRESKA ::a ili A::a ?
  int s=::a;
  int t=A::a;
  int u=b;   //GRESKA: ::b ili A::b ?
  int v=::b;
  int w=A::b;
  int b=5;   // def. lokalne promenljive b
  int x=b;   // lokalno b
  int y=::b; // globalno b
  int z=A::b;
}
```