Вовед во

C++ STL :: Standard Template Library

Стек

За да може да се користи структурата стек од ***STL***-то, на почеток од програмата мора да се зададе директивата

```
#include <stack>
```

Стекот содржи елементи од одреден тип. За декларирање стек во којшто ќе бидат внесувани стрингови како податоци, се користи следниов код

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

Важно е да се нагласи дека за да се користи ***string***-от како податочна структура, мора во почетниот дел од програмата да се вклучи дире-ктивата

```
#include <string>
```

Доколку пак сакаме да дефинираме стек од целобројни вредности тогаш стекот се декларира на следниов начин

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

или општо запишано за произволен тип на податок (примитивен тип на податок, класа) ***T***

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

Дозволени методи

- За проверка дали стекот е празен, се користи методата ***empty***, т.е.

  ```
  s.empty() //→ bool
  ```

  и истата враќа ***true*** ако стекот е празен или ***false*** во случај кога стекот има барем еден елемент.
- За да се провери колку вкупно елементи се наоѓаат во стекот се користи методата ***size***

  ```
  s.size() //→ int
  ```
- За внесување на нов елемент ***t*** од тип ***T*** на врвот на стекот се користи методата ***push***, како на пример

  ```
  s.push(t) //→ void
  ```
- За читање и одстранување на елемент од врвот на стекот се користи методата ***pop***, како во случајот

  ```
  s.pop() //→ T
  ```
- За да го прочитаме, но не и за да го одстраниме елементот од врвот од стекот се користи методата ***top***

  ```
  s.top() //→ T
  ```

Пример:

```cpp
//direktivi za vklucuvanje na potrebnite klasi
#include <iostream>
#include <stack>
#include <string>
using namespace std;

int main() {

	//deklarirame stek struktura so elementi od tip string
	stack<string> s;

	//Vnesuvame tri string-a po red
	s.push("PROBA 1");
	s.push("PROBA 2");
	s.push("PROBA 3");

	string w;
	cin>>w;

	s.push(w);

	//Pecatime broj na zborovi koisto sme gi vnesle
	cout << "Broj na zborovi = " << s.size() << endl;

	//Se dodeka stekot ne e prazen
	while (!s.empty()) {
		//Go citame zborot od vrvot na stekot i go pecatime
		cout << s.top() << endl;
		//Go citame i odstranuvame stringot od vrvot na stekot
		s.pop();
	}

	return 0;
}
```

Ред

Директивата со чија помош се овозможува користењето на структурата ***queue*** од ***STL***-то е следнава:

```
#include <queue>
```

За да декларираме ред за елементи од тип ***T*** ја користиме следнава линија код:

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

Дозволени методи

- За проверка на тоа дали редот е празен, се користи методата ***empty***

  ```
  q.empty() //→ bool
  ```

- За да се дознае колку вкупно елементи се внесени во редот се користи методата ***size***

  ```
  q.size() //→ int
  ```

- За поставување на елемент ***t*** од тип ***T*** на крајот на редот ***q***, се користи следниов код

  ```
  q.push(t) //→ void
  ```

- За читање и одстранување на елемент од почетокот на редот ***q*** се користи методата ***pop***, како во наведениот случај

  ```
  q.pop() //→ T
  ```

- За читање, но не и одстранување на последниот елемент од редот се користи методата ***back***, т.е.

  ```
  q.back() //→ T
  ```

- За читање, но не и одстранување на првиот елемент (преден дел на редот) се користи методата ***front***, како во случајов

  ```
  q.front() //→ T
  ```

Пример:

```
//direktivi za vklucuvanje na potrebnite klasi
#include <iostream>
#include <queue>
#include <string>
using namespace std;

int main() {

	//deklarirame stek struktura so elementi od tip string
	queue<string> q;

	//Vnesuvame tri string-a po red
	q.push("PROBA 1");
	q.push("PROBA 2");
```

```
	q.push("PROBA 3");

	string w;
	cin>>w;

	q.push(w);

	//Pecatime broj na zborovi koisto sme gi vnesle
	cout << "Broj na zborovi = " << q.size() << endl;

	//Se dodeka stekot ne e prazen
	while (!q.empty()) {
		//Go citame zborot od vrvot na stekot i go pecatime
			cout << q.front() << endl;
		//Go citame i odstranuvame stringot od vrvot na stekot
			q.pop();
	}

	return 0;
}
```

Забелешка

Многу е важно да се нагласи дека функциите ***top*** од структурата ***stack***, потоа ***front*** и ***back*** враќаат референци кон елементите од соодветните структури. Уште еднаш да повториме, истите не враќаат покажувач, ниту копија од елементот, туку враќаат референца. Тоа значи дека преку референцата којашто ви се враќа, можете да повикате било која метода или да извршите било која од дозволените операции. На овој начин, директно ќе го измените елементот, во рамки на структурата, само затоа што овие методи враќаат **константна референца**.

Пример:

```
	s.top()=t1;//go izmenuvame elementot na vrvot od stekot

	q.back()=t2;//go izmenuvame elementot na krajot od redot
	q.front()=t3;//go izmenuvame elementot na pocetokot na redot
```

Листи

За да може да се користи листата од ***STL***-то како структура, мора да се додаде следнава директива на почетокот од програмата, и тоа

```
#include <list>
```

За да декларираме листа од одреден тип ***T*** ја користиме следнава линија код:

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

Дозволени методи

- За да се провери дали листата е празна се користи методата ***empty***, како на пример

  ```
  ls.empty() //→ bool
  ```

- За да се дознае тековниот број на елементи во листата се користи методата ***size***

  ```
  ls.size() //→ int
  ```

- За додавање на елемент ***t*** од тип ***T*** на крајот од листата ***ls*** се користи следниов код

  ```
  ls.push_back(t) //→ void
  ```

- За читање и одстранување на елемент од крајот на листата ***ls***, се користи методата ***pop_back***

  ```
  ls.pop_back() //→ T
  ```

- Аналогно, постојат соодветни методи за додавање и одстранување на елемент на почетокот од листата ***ls***.

  ```
  ls.push_front(t) //→ void
  ls.pop_front() //→ T
  ```

- За сортирање на листата се користи методата ***sort***, како во следниот пример:

  ```
  ls.sort() //→ void
  ```

- За бришење на сите елементи од листата се користи методата ***clear***

  ```
  ls.clear() //→ void
  ```

- За “превртување“ на листа се користи методата ***reverse***

  ```
  ls.reverse() //→ void
  ```

- За доделување копија од листа ***ls1*** на друга листа ***ls2***, се користи следниов код

  ```
  ls2=ls1;
  ```

Вектори (или прво-класни низи)

Векторите се разликуваат од листите по тоа што постои можност за директен пристап кон елемент од структурата преку користење на индекси, т.е. операторот [].

За да може да се користи оваа структура потребна е следнава директива:

```
#include <vector>
```

За да декларираме вектор од одреден тип ***T*** ја користиме следнава линија код:

```
vector<T> v;
```

Дозволени методи

- За проверка на тоа дали векторот е празен се користи методата ***empty***

  ```
  v.empty() //→ bool
  ```

- За да се дознае тековниот број на елементи во рамки на векторот ***v*** се користи методата ***size***

  ```
  v.size() //→ int
  ```

- За да се додаде нов елемент ***t*** од тип ***T*** на крајот од векторот ***v*** се користи методата ***push_back***, како што е покажано преку следниов код

  ```
  v.push_back(t) //→ void
  ```

- За да се прочита и одстрани елемент од крајот на векторот ***v*** се користи методата ***pop_back***

  ```
  v.pop_back() //→ T
  ```

- За пристап (константна референца) кон првиот елемент од почетокот на векторот ***v*** се користи методата ***front***

  ```
  v.front() //→ T
  ```

- За пристап (константна референца) кон последниот елемент од крајот на векторот ***v*** се користи методата back

  ```
  v.back() //→ T
  ```

- За директен пристап на ***i***-тиот елемент од векторот ***v*** (0<= ***i***<***v.size()***) без проверка на тоа дали елементот односно индексот постои, се користи наполно ист начин како кај обичните низи:

  ```
  v[i]
  ```

- За сигурен пристап кон ***i***-тиот елемент се користи методата ***at***, како што е покажано преку кодот што следува

  ```
  v.at(i)//→ void
  ```

- За доделување на копија од векторот ***v1*** на векторот ***v2*** се користи следниов код

  ```
  v2=v1;
  ```

Пример:

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

int main(){

    //Inicijalizirame vektor so 4-ri celobrojni vrednosti
    vector<int> v(4);

    //Gi inicijalizirame vrednostite
    v[0]=100;
    v[1]=200;
    v[2]=300;
    v[3]=400;

    //metodata push_back pravi prosiruvanje(dupliranje na prostorot) na vektorot
dokolku ima potreba za toa
    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;

}
```

Контејнери и итератори

Контејнер е секоја податочна структура којашто содржи елементи од ист тип. Листата, векторот, стекот и редот се контејнери.

Елементите во рамки на контејнерите се пристапуваат преку итератори. Така на пример за да декларираме итератор за вектор со целобројни вредности се користи кодот

```
vecor<int>::iterator
```

Итераторите се особено важни за листите бидејќи не постои начин за директен пристап кон елементите од листата. Важно е да се напомене дека итераторите се всушност покажувачи кон елементите од контејнерот.

Така, нека е декларирана листа ***ls*** за елементи од тип ***int***.

```
list<int> ls;
```

За да ги поминеме елементите од оваа листа, мора на елементите од листата да им пристапиме преку итератор, односно преку покажувач.

```
list<int>::iterator p;
for(p=ls.begin();p!=ls.end();p++)
     cout<<*p<<endl;
```

- Декларација на итератор за контејнер од тип ***C***

  ```
  C::iterator p
  ```

- Поместување на итераторот кон следниот елемент од контејнерот се врши со

  ```
  p++
  ```

- Самиот итератор ***p*** е покажувач. Вредноста кон која покажува ***p*** е

  ```
  *p
  ```

- За поставување на итераторот да покажува кон првиот елемент од контејнерот ***C*** се користи методата

  ```
  c.begin()
  ```

- За поставување на итераторот да покажува кон последниот елемент од контејнерот ***C*** се користи методата

  ```
  c.end()
  ```

Листи и итератори

Кога станува збор за листата како контејнер и итераторите, постојат методи коишто дозволуваат вметнување елемент во листа и бришење на елемент во листа.

Додавање на елемент ***x*** во листата ***ls*** пред итераторот ***p*** се врши со методата ***insert***:

```
ls.insert(p,x);
```

Бришење на елемент од листата ***ls***, кон којшто покажува итераторот ***p*** се врши со методата ***erase***:

```
ls.erase(p);
```

Пример:

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

void print(list<char> ls){

     for(list<char>::iterator p=ls.begin();p!=ls.end();p++)
                              cout<<*p<<" ";
     cout<<endl;

}

int main(){

      list<char> ls;
      list <char>::iterator p;

      ls.push_back('o');
      ls.push_back('a');
      ls.push_back('t');

      p=ls.begin();

      // p pokazuva na 'o' vo ('o', 'a', 't')
      cout <<" "<< *p<<endl;

      print(ls);

      ls.insert(p, 'c');

      // ls sega izgleda ('c', 'o', 'a', 't') a p seuste pokazuva na 'o'
      cout <<" "<< *p<<endl;

      print(ls);

      ls.erase(p);
```

```
    // p pokazuva na 'o' koesto veke ne e vo ls
    cout <<" "<< *p<<endl;

    print(ls);

    //za odstranuvanje na prviot element od listata ls
    ls.erase(ls.begin());

    print(ls);

    return 0;

}
```

Забелешка

Со вклучување на директивата

```
    #include<algorithm>
```

и употреба на итераторите и векторите можно е ефикасно сортирање преку користење на методата ***sort***.

Пример:

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

//Funkcija za pecatenje na elementite od vektorot a
void print( vector<int> a){

    for(vector<int>::iterator ai=a.begin(); ai!=a.end(); ++ai)

     cout << *ai << " ";

    cout << endl;

    cout << "----------------"<<endl;

}

int main()
{
    //Deklarirame vektor a
    vector<int> a;
    // Gi vnesuvame broevite 9,8,7,6,5,4,3,2,1 redosledno vo vektorot
    for(int i=0; i<9;++i)
     a.push_back(9-i);

    //Elementite od vektorot se podredeni po sledniov redosled 9,8,7,6,5,4,3,2,1
    print(a);
```

```
    //Gi sortirame elementite od vektorot, preku povikuvanje na metodata sort od
STL <algorithm> bibliotekata
    sort( a.begin(), a.end() );

    //Elementite se podredeni
    print(a);

    return 0;

}
```

Multimap

Multimap контејнерот може да се користи наместо ***hash*** структура. Во случајов станува збор за структура за мапирање. Секој елемент е составен од два дела: ***клуч*** и ***вредност***. Структурата дозволува повторување на клучевите по коишто се пребарува.

Клучевите по коишто се пребарува мора да бидат споредливи односно да биде дефинирана операцијата <. За стандардните типови податоци, вклучувајќи ја и библиотеката ***string***, ваквиот оператор за споредба е веќе дефиниран.

За да може да се користи оваа структура, потребно е на почетокот од програмата да се вклучи следнава директива

```
#include <map>
```

Пример

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

int main()
{

  //  Definirame multimap
  multimap<string,int> mymm;
  // Definirame iterator za vakov tip na multimap
  multimap<string,int>::iterator it;

  // Vnesuvame elementi vo multimap-ot
  mymm.insert(pair<string,int>("niksa",10));
  mymm.insert(pair<string,int>("bliksa",20));
  mymm.insert(pair<string,int>("bliksa",30));
  mymm.insert(pair<string,int>("aha",40));
  mymm.insert(pair<string,int>("tdmm",50));
  mymm.insert(pair<string,int>("nel",60));
  mymm.insert(pair<string,int>("bliksa",60));

  // Gi pecatime elementite od multimap-ot
  for (it = mymm.begin();it != mymm.end();++it)
       cout << "  [" << (*it).first << ", " << (*it).second << "]" << endl;

  // Za da gi ispecatime site vrednosti pri site pojavuvanja na klucot bliksa
  pair<multimap<string,int>::iterator,multimap<string,int>::iterator> ret;

  ret = mymm.equal_range("bliksa");
  for (it=ret.first; it!=ret.second; ++it)
    cout << " " << (*it).second;
  cout << endl;

  // Broj na elementi so kluc bliksa
  cout<<endl<<"Broj na elementi so kluc 'bliksa' : "<<mymm.count("bliksa");
```

```
  // Dodavanje na element na pocetok od multiset-ot
  it=mymm.begin();

  mymm.insert(it,pair<string,int>("bla",123));

  // Gi pecatime elementite od multimap-ot
  for (it = mymm.begin();it != mymm.end();++it)
       cout << "  [" << (*it).first << ", " << (*it).second << "]" << endl;

  return 0;
}
```

Користена литература

1. http://www.csci.csusb.edu/dick/samples/
2. http://www.yolinux.com/TUTORIALS/LinuxTutorialC++STL.html
3. http://www.cplusplus.com/reference/stl/