პრაქტიკული მეცადინეობა 4

სააუდიტორიო სამუშაო:

- ამოცანა 1 /სტეკის მეთოდების გააზრება/
- ამოცანა 2 /რიგის მეთოდების გააზრება/ >>>
- სავარჯიშოები >>>

ამოცანა 1 /სტეკის მეთოდების გააზრება/. შევადგინოთ მარტივი პროგრამა, რომელიც იყენებს სტეკის ხუთ მეთოდს: `size(), push(), pop(), top(),empty().`

ამოხსნა: პროგრამა, ოდნავი ცვლილებით, აღებულია წიგნიდან STL-ის შესახებ, რომელიც მითითებულია სილაბუსში.

```
//stack  container adaptor
#include<iostream>
#include<vector>
#include<stack>
#include<iterator>
using namespace std;

int main()
{
    vector<int> v;
    for(int i=0; i<9; i++)
        v.push_back(rand()%77);
    ostream_iterator<int> oi(cout, "\t");
    copy(v.begin(), v.end(),oi);
    cout << endl;

    stack<int> s;
    cout << "The stack size is now: " << s.size() << endl;
    cout << "pushing 4 elements from vector: " << endl;
    for(int i=0; i<4; i++)
        s.push(v[i]);
    cout << "The stack size is now: " << s.size() << endl;

    cout << "poping 3 elements from stack: " << endl;
    for(int i=0; i<3; i++)
    {
        cout << s.top()<< '\t';
        s.pop();
    }
    cout << endl << "The stack size is now: " << s.size() << endl;

    cout << "pushing 5 next elements from vector: " << endl;
    for(int i=4; i<9; i++)
        s.push(v[i]);
    cout << "The stack size is now: " << s.size() << endl;

    cout << "poping all elements from stack: " << endl;
    while(!s.empty())
    {
        cout << s.top() << '\t';
        s.pop();
    }
    cout << endl << "The stack size is now: " << s.size() << endl;
}
```

პროგრამის შედეგს აქვს შემდეგი სახე:

```
41  64  20  12  73  16  5  21  12
The stack size is now: 0
pushing 4 elements from vector:
The stack size is now: 4
poping 3 elements from stack:
12     20     64
The stack size is now: 1
pushing 5 next elements from vector:
The stack size is now: 6
poping all elements from stack:
12     21     5      16     73     41
The stack size is now: 0
Press any key to continue . . .
```

პირველ რიგში, სტეკის ინიციალიზება ვექტორისგან განაცხადის გაკეთების მომენტში ასლის კონსტრუქტორით არ ხერხდება, ამიტომ იძულებული ვართ სტეკის შევავსოთ `push()` მეთოდით. შევნიშნოთ, რომ ახალი სტეკის აგება შეგვიძლია უკვე არსებულის ასლის საფუძველზე.

რადგან სტეკის მოქმედების პრინციპი აღიწერება დევიზით: ბოლო მოვიდა - პირველი წავიდა (Last-in, first-out ანუ LIFO), ამიტომ ელემენტები ემატება მარჯვნიდან.

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

განაცხადი იგივეა რაც

```
stack< int, deque< int > > s;
```

ანუ ამ სტეკის `.push()` მეთოდი იგივეა, რაც ადაპტირებული დეკის `.push_back()`. თუ სტეკზე განაცხადის გავაკეთებდით ასე:

```
stack< int, list< int > > s;
```

მაშინ ამ სტეკის `push()` მეთოდი იგივეა, რაც ადაპტირებული სიის `push_back()`.

შესაბამისი ცვლილებებით, იგივეს თქმა შეიძლება `pop()` მეთოდზე.

სტეკის დაცარიელება შეგვეძლო `for` შეტყობინების გამოყენებით, მაგრამ

```
while(!s.empty())
```

შეტყობინება იგივე საქმეს აკეთებს უფრო ბუნებრივად. ეს შეტყობინება შეწყდება, როდესაც სტეკი დაცარიელდება. ამ დროს `s.empty()` გახდება ჭეშმარიტი, ხოლო `!s.empty()` მცდარი.

ცხადია აგრეთვე, რომ `stack<int> s;` განაცხადის შემთხვევაში ამ სტეკის `top()` მეთოდი იგივეა, რაც ადაპტირებული დეკის `back()` მეთოდი.

<<< ამოცანა 2 /რიგის მეთოდების გააზრება/. შევადგინოთ მარტივი პროგრამა, რომელიც იყენებს რიგის მეთოდებს: `size()`, `push()`, `pop()`,`front()`,`empty()`.

ამოხსნა: ეს პროგრამაც იგივე წყაროდანაა აღებული.

```
//stack  container adaptor
#include<iostream>
#include<list>
#include<queue>
#include<iterator>
using namespace std;

int main()
{
    vector<int> v;
    for(int i=0; i<9; i++)
        v.push_back(rand()%77);
    ostream_iterator<int> oi(cout, "\t");
    copy(v.begin(), v.end(),oi);
    cout << endl;

    queue<int, list<int> > q;
    cout << "The queue size is now: " << q.size() << endl;
    cout << "Pushing 4 elements from vector: " << endl;
    for(int i=0; i<4; i++)
```

```
        q.push(v[i]);
    cout << "The queue size is now: " << q.size() << endl;

    cout << "poping 3 elements from queue: " << endl;
    for(int i=0; i<3; i++)
    {
        cout << q.front() << '\t';
        q.pop();
    }
    cout << endl << "The queue size is now: " << q.size() << endl;

    cout << "Pushing 5 next elements from vector: " << endl;
    for(int i=4; i<9; i++)
        q.push(v[i]);
    cout << "The queue size is now: " << q.size() << endl;

    cout << "poping all elements from queue: " << endl;
    while(!q.empty())
    {
        cout << q.front()  << '\t';
        q.pop();
    }
     cout << endl << "The queue size is now: " << q.size() << endl;
}
```

პროგრამის შესრულების შედეგი არის:

```
41  64  20  12  73  16  5   21   12
The queue size is now: 0
Pushing 4 elements from vector:
The queue size is now: 4
poping 3 elements from queue:
41    64    20
The queue size is now: 1
Pushing 5 next elements from vector:
The queue size is now: 6
poping all elements from queue:
12    73    16    5     21    12
The queue size is now: 0
Press any key to continue . . .
```

პირველ რიგში, რიგის ინიციალიზება განაცხადის გაკეთების მომენტში ვექტორის ასლის საფუძვლეზე, ამიტომ იძულებული ვართ სტეკი შევვავსოთ `push()` მეთოდით. რადგან რიგის მოქმედების პრინციპი აღიწერება დევიზით: პირველი მოვიდა - პირველი წავიდა (first-in, first-out ანუ FIFO), ამიტომ ელემენტები ემატება მარცხნიდან ანუ ფრონტიდან. ამავე მიზეზის გამო გამოიყენება აქ `front()` მეთოდი და არა `top()`. რიგს შეუძლია დეკის ადაპტირებაც. მაგრამ არ შეუძლია ვექტორის ადაბტირება, რადგან ვექტორს არ შეუძლია ელემენტების სწრაფად ამოღება თავიდან (ანუ, არა აქვს მეთოდი `pop_front()`).

რიგის `push()` მეთოდი იგივეა, რაც ადაპტირებული სიის `push_back()`.

რიგის დაცარიელება შეგვეძლო `for` შეტყობინების გამოყენებით, მაგრამ

```
while(!q.empty())
```

შეტყობინება იგივე საქმეს აკეთებს უფრო ბუნებრივად. ეს შეტყობინება შეწყდება, როდესაც რიგი დაცარიელდება. ამ დროს `q.empty()` გახდება ჭეშმარიტი, ხოლო `!q.empty()` მცდარი.

<<< სავარჯიშოები

1. შექმენით ანალოგიური პროგრამა, ოღონდ სტეკი შექმენით სიის საფუძველზე
   ```
   stack< int, list< int > > s;
   ```
2. შექმენით ანალოგიური პროგრამა, ოღონდ სტეკი შექმენით ვექტორის საფუძველზე
3. შექმენით ანალოგიური პროგრამა, ოღონდ რიგი შექმენით ორმხრივი რიგის საფუძველზე:
   ```
   queue< int, deqe< int > > q;
   ```
4. გაარჩიეთ პროგრამა http://cs.stmarys.ca/~porter/csc/ref/stl/cont_queue.html -დან:

```
#include <iostream>
```

```
#include <iomanip>
#include <queue>
using namespace std;

int main()
{
	cout << "\nThis program illustrates the assignment of one queue "
		"to another,\nand the comparison of queues.";
	cout << "\nPress Enter to continue ... ";  cin.ignore(80, '\n');

	queue<int> q1;
	q1.push(1);
	q1.push(2);
	q1.push(3);
	q1.push(4);
	cout << "\nThe queue q1 contains " << q1.size() << " values.";
	cout << "\nPress Enter to continue ... ";  cin.ignore(80, '\n');

	cout << "\nNow we create three new empty queues-q2, q3, q4-"
		"and assign q1 to all three.\nThen we display the contents of "
		"q1 to show what q2, q3 and q4 all contain.\nNote that the "
		"process of displaying the values in q1 empties q1.";
	queue<int> q2, q3, q4;
	q4 = q3 = q2 = q1;
	cout << "\nPress Enter to continue ... ";  cin.ignore(80, '\n');

	cout << "\nHere are the values of q1, in \"FIFO\" order:\n";
	//This is the proper way to access the elements of a queue:
	while (!q1.empty())
	{
		cout << "Popping: ";
		cout << q1.front() << "\n";
		q1.pop();
	}
	cout << "Press Enter to continue ... ";  cin.ignore(80, '\n');

	cout << "\nNext we display the contents of q2 to confirm that "
		"q1 did get assigned to q2.";
	cout << "\nPress Enter to continue ... ";  cin.ignore(80, '\n');

	cout << "\nHere are the values of q2, in \"FIFO\" order:\n";
	while (!q2.empty())
	{
		cout << "Popping: ";
		cout << q2.front() << "\n";
		q2.pop();
	}
	cout << "Press Enter to continue ... ";  cin.ignore(80, '\n');

	cout << "\nFinallly, we push the value 5 onto q4, and then we "
		"output the result of\ncomparing q3 and q4 using each of the "
		"relational operators.";
	cout << "\nPress Enter to continue ... ";  cin.ignore(80, '\n');

	q4.push(5);
	cout << "\nq3 == q4 is " << boolalpha << (q3 == q4) << ".";
	cout << "\nq3 != q4 is " << boolalpha << (q3 != q4) << ".";
	cout << "\nq3 <  q4 is " << boolalpha << (q3 <  q4) << ".";
	cout << "\nq3 <= q4 is " << boolalpha << (q3 <= q4) << ".";
	cout << "\nq3 >  q4 is " << boolalpha << (q3 >  q4) << ".";
	cout << "\nq3 >= q4 is " << boolalpha << (q3 >= q4) << ".";
	cout << "\nPress Enter to continue ... ";  cin.ignore(80, '\n');

}
```