ლაბორატორიული 5: კონტეინერთა ადაპტერები

საკითხები:

- ამოცანა /კლასის ობიექტების რიგის შექმნა და ტესტირება
    1. მონაცემების ფაილის შიგთავსი >>>
    2. ფეხბურთის კლუბის კლასი და მისი იმპლემენტაცია >>>
    3. პროგრამა-დრაივერი >>>
- სავარჯიშოები >>>

ამოცანა 1 /კლასის ობიექტების რიგის შექმნა და ტესტირება/. შევქმნათ მარტივი პროგრამა, რომელიც იყენებს რიგის მეთოდებს: `size()`, `push()`, `pop()`, `front()`, `empty()`. რიგი შედგება ჩვენს მიერ შექმნილი კლასის ობიექტებისგან.

ამოხსნა

<<< მონაცემების ფაილების შიგთავსი

”**data.txt**” ფაილში წერია:

```
Liverpool    1892
Real         1902
Manchester   1880
barcelona    1899
Milan        1899
```

<<< ფეხბურთის კლუბის კლასი და მისი იმპლემენტაცია

კლასის აღწერა მოთავსებულია “FC.h“ ფაილში და აქვს სახე:

```
#pragma once
class FC
{
public:
	string name;
	int year;
	FC(void);
	~FC(void);
	FC(istream & is);
	friend istream & operator>>(istream &, FC &);
	friend ostream & operator<<(ostream &, FC const &);
};
```

რადგან რიგი მარტივი სტრუქტურაა, ამიტომ კლასს არ სჭირდება ბევრი მეთოდი.

კლასის იმპლემენტაცია მოთავსებლია “FC.cpp“ ფაილში და მას აქვს სახე:

```
#include<iostream>
#include<string>
using namespace std;
#include "FC.h"

FC::FC(void)
: year(0)
{
}
FC::~FC(void)
{
}
FC::FC(istream & is)
{
	is >> name >> year;
}
istream & operator>>(istream & is, FC &a)
```

```
{
	is >> a.name >> a.year;
	return is;
}
ostream & operator<<(ostream & os, const FC & a)
{
	os << a.name << "\t" << a.year << endl;
	return os;
}
```

<<< პროგრამა-დრაივერი:

```
#include<iostream>
#include<fstream>
#include<list>
#include<queue>
using namespace std;
#include "FC.h"

int main()
{
	queue<FC> que;
	FC tmp;
	ifstream fin("data.txt");
	while (fin >> tmp)
		que.push(tmp);

	while (!que.empty())
	{
		cout << que.front();
		que.pop();
	}
}
```

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

რიგის დაცარიელება შეგვეძლო `for` შეტყობინების გამოყენებით, მაგრამ

```
while(!que.empty())
```

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

შეტყობინება `cout << que.front();` გვიჩვენებს იმ ობიექტის მონაცემებს, რომელიც მოქცეულია რიგის თავში.

უშუალოდ რიგს და სტეკს არ სჭირდება, რომ მის ობიექტებზე გადატვირთული იყოს შეტანის და გამოტანის ოპერატორები. მაგალითად, თუ კლასს მოვაშორებთ შეტანა-გამოტანის ოპერატორებს, მაგრამ პროგრამა-დრაივერს შევცვლით შემდეგნაირად:

```
#include<iostream>
#include<fstream>
#include<list>
#include<queue>
#include<string>
using namespace std;
#include "FC.h"

int main()
{
	queue<FC> que;
	FC tmp;
```

```
        ifstream fin("data.txt");
        while (fin >> tmp.name >> tmp.year)
               que.push(tmp);

        while (!que.empty())
        {
               cout << que.front().name << que.front().year << endl;
               que.pop();
        }
}
```

მივიღებთ იგივე შედეგს.

<<< სავარჯიშოები:

1. შექმენით ანალოგიური პროგრამა სტეკისთვის.
2. რა გვიშლის ხელს, რომ შევქმნათ ანალოგიური პროგრამა პრიორიტეტების რიგისთვის?
3. შექმენით ობიექტების პრიორიტეტებიანი რიგი.
4. გავუშვათ პრაქტიკული მეცადინეობის ამოცანები, შევიტანოთ გარკვეული ცვლელებები.
5. ინტერნეტსი მოძებნეთ აღნიშნულ საკითხთან დაკავშირებული პროგრამული კოდები და გაარჩიეთ.
6. გაარჩიეთ შემდეგი პროგრამა (იხ. http://cs.stmarys.ca/~porter/csc/ref/stl/cont_stack.html)

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

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

    stack<int> s1;
    s1.push(1);
    s1.push(2);
    s1.push(3);
    s1.push(4);
    cout << "\nThe stack s1 contains " << s1.size() << " values.";
    cout << "\nPress Enter to continue ... ";  cin.ignore(80, '\n');

    cout << "\nNow we create three new empty stacks-s2, s3, s4-"
          "and assign s1 to all three.\nThen we display the contents of "
          "s1 to show what s2, s3 and s4 all contain.\nNote that the "
          "process of displaying the values in s1 empties s1.";
    stack<int> s2, s3, s4;
    s4 = s3 = s2 = s1;
    cout << "\nPress Enter to continue ... ";  cin.ignore(80, '\n');

    cout << "\nHere are the values of s1, in \"LIFO\" order:\n";
    //This is the proper way to access the elements of a stack:
    while (!s1.empty())
    {
          cout << "Popping: ";
          cout << s1.top() << "\n";
          s1.pop();
    }
```

```
    cout << "Press Enter to continue ... ";  cin.ignore(80, '\n');

    cout << "\nNext we display the contents of s2 to confirm that "
            "s1 did get assigned to s2.";
    cout << "\nPress Enter to continue ... ";  cin.ignore(80, '\n');

    cout << "\nHere are the values of s2, in \"LIFO\" order:\n";
    while (!s2.empty())
    {
           cout << "Popping: ";
           cout << s2.top() << "\n";
           s2.pop();
    }
    cout << "Press Enter to continue ... ";  cin.ignore(80, '\n');

    cout << "\nFinallly, we push the value 5 onto s4, and then we "
           "output the result of\ncomparing s3 and s4 using each of the "
           "relational operators.";
    cout << "\nPress Enter to continue ... ";  cin.ignore(80, '\n');

    s4.push(5);
    cout << "\ns3 == s4 is " << boolalpha << (s3 == s4) << ".";
    cout << "\ns3 != s4 is " << boolalpha << (s3 != s4) << ".";
    cout << "\ns3 <  s4 is " << boolalpha << (s3 <  s4) << ".";
    cout << "\ns3 <= s4 is " << boolalpha << (s3 <= s4) << ".";
    cout << "\ns3 >  s4 is " << boolalpha << (s3 >  s4) << ".";
    cout << "\ns3 >= s4 is " << boolalpha << (s3 >= s4) << ".";
    cout << "\nPress Enter to continue ... ";  cin.ignore(80, '\n');
}
```