Ministerul Educației al Republicii Moldova

Universitatea Tehnică a Moldovei
Facultatea CIM

Departamentul Ingineria Software și Automatică

RAPORT

Lucrare de laborator Nr. 1
La APPOO
Tema: Biblioteca standartă a șabloanelor.

A efectuat: st. Gr. TI-142
Cojucari Dan

A verificat: Gavrișco Alexandru
Andrei Postaru

Chișinău 2017

Lucrarea de laborator nr.1

Tema: *Biblioteca standartă a şabloanelor*

Scopul lucrării: *Studierea tehnologiei de programare folosind Standard Template Library(STL) a limbajului C++.*

1. Indicaţii teoretice

STL se bazează pe trei concepte centrale: containeri, iteratori şi algoritmi. Ca tehnică de programare folosită, e bine de ştiut că orientarea pe obiecte aproape că lipseşte. În schimb se utilizează din plin polimorfismul parametric. În C++ numele acestuia este „template", în C# şi Java se obişnuieşte să se spuna „generice".

Containeri

Containerii STL sunt de trei tipruiŞ containeri de secvenţă, containeri asociativi şi adaptori containeri.

Containerii secvenţă

- vector – fişierul header <vector> – implementarea unui tablou dinamic, cu redimensionare automată la inserarea unui nou element sau la ştergerea unui element;
- list – fişierul header <list> - listă dublu înlănţuită cu inserare şi ştergere rapidă;
- deque – fişierul header <deque> - coadă în care elementele pot fi adăugate sau şterse din ambele capete; diferă de coada obişnuită prin faptul că acolo adăugarea şi ştergerea elementelor se face la un singur capăt.

Containerii asociativi

- map – fişierul header <map> – păstrează asocieri între perechi de valori şi chei, mapare unu-la-unu;
- multimap – fişierul header <map> – este similar cu map, dar acceptă duplicate, mapare unu-la-mai multe;
- set – fişierul header <set> – cheile sunt chiar valorile păstrate;
- multiset – fişierul header <set> – este similar cu set, dar acceptă duplicate.

Adaptori container

- stack – fişierul header <stack> – implementează o stivă – last-in-first-out (LIFO);
- queue – fişierul header <queue> – implementează o coadă – first-in-first-out (FIFO);
- priority_queue – fişierul header <queue> –coadă în care elementul cu prioritatea cea mai mare este întotdeauna primul element extras.

Iteratori

Iteratorii au multe caracteristici comune cu pointerii şi sunt folosiţi, printre altele, pentru a pointa la elemente ale containerilor de tip first-class. Un iterator este un obiect care permite programatorului să parcurgă toate elementele unei colecţii, indiferent de implementarea acesteia. Există iteratori STL a căror implementare este adaptată unor anumiţi containeri.

Sunt, însă, iteratori care pot opera asupra tuturor containerilor. De exemplu, operatorul de dereferenţiere * se poate aplica unui iterator pentru a folosi elementul asupra căruia pointează. Operatorul ++ aplicat unui iterator mută poziţia iteratorului asupra următorului element din container.

Folosim iteratorii împreună cu secvenţele. Secvenţele pot fi organizate în containeri sau pot fi secvenţe de intrare ori ieşire. Programul de mai jos demonstrează citirea datelor folosind istream_iterator de la intrarea standard care poate fi privită ca o secvenţă de date de intrare în program. Programul ilustrează, de asemenea, tipărirea datelor folosind ostream_iterator la ieşirea standard care poate fi privită ca o secvenţă de date de ieşire din program. Programul citeşte de la tastatură doi întregi şi afişează suma lor.

Algoritmi

Un aspect esenţial al STL este că oferă algoritmi care pot fi folosiţi generic pentru o mare varietate de containeri: inserare, ştergere, căutare, sortare etc. STL include aproximativ 70 de algoritmi standard. Algoritmii operează asupra elementelor unei secvenţe doar indirect, prin intermediul iteratorilor. Deoarece STL este extensibil, se pot adăuga cu uşurinţă noi algoritmi fără a opera nicio modificare asupra containerilor. Algoritmii pot opera şi asupra tablourilor declarate în formatul promovat de limbajul C ca pointeri.

Algoritmii STL sunt de două tipuri. Cei de tip mutating-sequence fac modificări asupra containerilor pe care sunt aplicati, în timp ce algoritmii non-mutatingsequence se execută fără a schimba conţinutul containerilor.

2. Sarcina lucrării

De elaborat 3 programe. Primul program demostrează folisrea contaierilor și a tipurilor de date standart. Programul 2 demonstrează folosirea containerilor și a claselor definite de utilizator. Programul 3 demonstrează folosirea algoritmilor STL.

Programul 1 va efectua:

1. Să se creeze contanier conform variantei și să se umple cu date conform variantei.
2. Să se vizualizeze containerul.
3. Să se modifice containerul, prin ștergerea unui element și inserarea altui.
4. Să se vizualizeze containerul utilizînd iteratorul.
5. Să se creeze al 2-lea container cu date de același tip.
6. Să se modifice primul container, ștergînd elemente de la al n-lea element și să se adauge toate elementele containerului 2.
7. Să se vizualizeze primul și al 2-lea container.

Programul 2 va efectua aceleași funcții ca și primul program, dar cu date de tip utilizator.

Programul 3 va efectua:

1. Să se creeze contanier conform variantei și să se umple cu date de tip utilizator.
2. Să se sorteze containerul descrescător.
3. Să se vizualizeze containerul.

4. Să se găsească un element în container conform condiției căutate.
5. Elementele care satisfac condiția să fie copiete în container de al 2-lea tip.
6. Să se vizualizeze containerul 2.
7. Să se sorteze containerele descrescător.
8. Să se vizualizeze ambele containere.
9. Să se obțină al 3-lea container prin fuziunea primelor două.
10. Să se afișeze containerul 3.
11 .Să se afișeze cîte elemente satisfac condiția.
12.Să se determine dacă contanerul coține elementul căutat.

3. Realizarea sarcinii

Codul sursă este anexat în anexă.

Pentru al dolea program am definit o clasă Carte ce conține o variabilă integer cu numărul paginilor, dar și numele cărții. Am definit în cadrul acestei clase constructorii dar și am supraîncărcat operatorii de citire și afișare, dar și comparare, pentru că acesta este folosit în cadrul sortării.

```
class Carte {
private:
    int pagesNumber;
    char* name;
public:
    Carte() {
        this->pagesNumber = 0;
        this->setName("Fara_Nume");
    }

    Carte(char* name, int pages) {
        this->pagesNumber = pages;
        this->setName(name);
    }

    void setPagesNumber(int nr){
        this->pagesNumber = nr;
    }

    int getPagesNumber(){
        return this->pagesNumber;
    }

    void setName(char* name) {
        this->name = new char[strlen(name) + 1];
        strcpy(this->name, name);
    }

    char* getName(){
        return this->name;
    }

    friend ostream& operator << (ostream& out, Carte obj) {
        out << "Nume   : " << obj.name <<endl;
        out << "Pagini : " << obj.pagesNumber <<endl;
        return out;
    }

    friend istream& operator >> (istream& in, Carte& obj) {
```

```
    cout << "Introduceti numele cartii    : ";
    in >> obj.name;
    cout << "Introduceti numarul de pagini: ";
    in >> obj.pagesNumber;
    return in;
}

friend bool operator < (Carte first, Carte second) {
    return first.pagesNumber < second.pagesNumber;
}
   };
```

Concluzie

În urma efectuării aceste lucrări de laborator am făcut un program cu meniu, care îndeplinește cele 3 condiții. Am lucrat cu biblioteca STL a limbajului C++. Am lucrat cu containerele queue și vector. Am observat că containerele queue si vector nu pot fi sortate. De asemenea, am observat ca coada nu are iterator, de aceea folosim un vector temporar, in care efectuam parcurgerea, apoi copiem inapoi in coada dupa finisarea parcurgerii.

Anexa

Codul sursă:

```
#include <iostream>
#include <conio.h>
#include <string.h>
#include <queue>
#include <vector>
#include <iterator>
#include <algorithm>

using namespace std;

class Carte {
private:
    int pagesNumber;
    char* name;
public:
    Carte() {
        this->pagesNumber = 0;
        this->setName("Fara_Nume");
    }

    Carte(char* name, int pages) {
        this->pagesNumber = pages;
        this->setName(name);
    }

    void setPagesNumber(int nr){
        this->pagesNumber = nr;
    }

    int getPagesNumber(){
        return this->pagesNumber;
    }

    void setName(char* name) {
        this->name = new char[strlen(name) + 1];
        strcpy(this->name, name);
    }

    char* getName(){
        return this->name;
    }

    friend ostream& operator << (ostream& out, Carte obj) {
        out << "Nume   : " << obj.name <<endl;
        out << "Pagini : " << obj.pagesNumber <<endl;
        return out;
    }

    friend istream& operator >> (istream& in, Carte& obj) {
        cout << "Introduceti numele cartii    : ";
        in >> obj.name;
        cout << "Introduceti numarul de pagini: ";
        in >> obj.pagesNumber;
        return in;
    }

    friend bool operator < (Carte first, Carte second) {
        return first.pagesNumber < second.pagesNumber;
    }
};
```

```
// Program 1
void pushDataInQueue(queue<char> &data, int size) {
    char tempChar;
    cout << "Introduceti " << size << " caractere" << endl;
    for(int i = 0; i < size; i++) {
        cin >> tempChar;
        data.push(tempChar);
    }
}

void showQueue(queue<char> &data) {
    queue<char> tmp(data);
    while(tmp.empty() == false){
        cout << tmp.front() << " ";
        tmp.pop();
    }
    cout << endl;
}

void program1() {
    int elementsNr, elementsForDelete;
    queue<char> queue1;

    system("cls");
    cout << "Introduceti numarul de elemente: ";
    cin >> elementsNr;
    pushDataInQueue(queue1, elementsNr);

    system("cls");
    cout << "Coada initiala:" << endl;
    showQueue(queue1);
    cout << endl;

    queue1.pop();
    queue1.push('s');
    cout << "(A fost sters primul element si a fost adaugat altul la sfirsitul cozii)" << endl;
    cout << "Coada 1 modificata:" << endl;
    showQueue(queue1);
    cout << endl;

    queue<char> queue2(queue1);
    cout << "(S-au copiat elementele din coada 1 in coada 2)" << endl;
    cout << "Coada 2:" << endl;
    showQueue(queue2);
    cout << endl;

    cout << "Introduceti numarul de elemente ce vor fi sterse (nr < " << elementsNr <<") : ";
    cin >> elementsForDelete;

    for (int i = 0; i < elementsForDelete; i++) {
        queue1.pop();
    }

    while (queue2.empty() == false) {
        queue1.push(queue2.front());
        queue2.pop();
    }

    cout << "(S-au copiat elementele din coada 2 in coada 1)" << endl;
    cout << "Coada 1: ";
    showQueue(queue1);
    cout << "Coada 2: ";
    showQueue(queue2);

    _getch();
```

```
}

// Program 2
void pushDataInQueue(queue<Carte> &data, int size) {
    cout << "Introduceti " << size << " elemente:" << endl;
    for (int i = 0; i < size; i++) {
        Carte tmp;
        cout << "Introduceti cartea " << i+1 << " :" << endl;
        cin >> tmp;
        data.push(tmp);
    }
}

void showQueue(queue<Carte> &data) {
    queue<Carte> tmp(data);
    while(tmp.empty() == false){
        cout << tmp.front();
        tmp.pop();
        cout << endl;
    }
}

void program2() {
    int elementsNr, elementsForDelete;
    queue<Carte> queue1;

    system("cls");
    cout << "Introduceti numarul de elemente: ";
    cin >> elementsNr;
    pushDataInQueue(queue1, elementsNr);

    system("cls");
    cout << "Coada initiala:" << endl;
    showQueue(queue1);
    cout << endl;

    queue1.pop();
    Carte tmp("Totul despre C/C++", 295);
    queue1.push(tmp);
    cout << "(A fost sters primul element si a fost adaugat altul la sfirsitul cozii)" << endl;
    cout << "Coada 1 modificata:" << endl;
    showQueue(queue1);
    cout << endl;

    queue<Carte> queue2(queue1);
    cout << "(S-au copiat elementele din coada 1 in coada 2)" << endl;
    cout << "Coada 2:" << endl;
    showQueue(queue2);
    cout << endl;

    cout << "Introduceti numarul de elemente ce vor fi sterse (nr < " << elementsNr <<") : ";
    cin >> elementsForDelete;

    for (int i = 0; i < elementsForDelete; i++) {
        queue1.pop();
    }

    while (queue2.empty() == false) {
        queue1.push(queue2.front());
        queue2.pop();
    }

    cout << "(S-au copiat elementele din coada 2 in coada 1)" << endl;
    cout << "Coada 1: " << endl;
    showQueue(queue1);
```

```
    cout << "Coada 2: " << endl;
    showQueue(queue2);

    _getch();
}

// Program 3
void sortQueue(queue<Carte> &data) {
    vector<Carte> tmp;

    while(data.empty() == false) {
        tmp.push_back(data.front());
        data.pop();
    }

    sort(tmp.begin(), tmp.end());

    for(int i = tmp.size()-1; i >= 0; i--) {
        data.push(tmp[i]);
    }
}

bool numarPar(Carte i) {
    return((i.getPagesNumber() % 2) == 0);
}

Carte findCarte(queue<Carte> &data){
       vector<Carte> tmp;

       while (data.empty() == false)     {
              tmp.push_back(data.front());
              data.pop();
       }

       for (vector<Carte>::iterator it = tmp.begin(); it != tmp.end(); ++it)     {
              data.push(*it);
       }

       return *(find_if(tmp.begin(), tmp.end(), numarPar));
}

void copyElements(queue<Carte> &data, vector<Carte> &vct) {
    vector<Carte> tmp;

    while (data.empty() == false) {
        tmp.push_back(data.front());
        data.pop();
    }

    for (vector<Carte>::iterator it = tmp.begin(); it != tmp.end(); ++it) {
        data.push(*it);
        if (numarPar(*it)) {
            vct.push_back(*it);
        }
    }
}

void showVector(vector<Carte> &data) {
    for (vector<Carte>::iterator it = data.begin(); it != data.end(); ++it) {
        cout << *it << endl;
    }
}

void sortAsc(queue<Carte> &queue1, vector<Carte> &vector1) {
    vector<Carte> tmp;
```

```
    while (queue1.empty() == false) {
        tmp.push_back(queue1.front());
        queue1.pop();
    }

    sort(tmp.begin(), tmp.end());
    sort(vector1.begin(), vector1.end());

    for(vector<Carte>::iterator it = tmp.begin(); it != tmp.end(); ++it) {
        queue1.push(*it);
    }
}

queue<Carte> concat(queue<Carte> &queue1, vector<Carte> &vector1) {
    queue<Carte> queue2(queue1);
    /*vector<Carte> tmp;

    while (queue1.empty() == false) {
        tmp.push_back(queue1.front());
        queue1.pop();
    }

    for (vector<Carte>::iterator it = tmp.begin(); it != tmp.end(); ++it) {
        queue1.push(*it);
        queue2.push(*it);
    }
    */
    for (vector<Carte>::iterator it = vector1.begin(); it != vector1.end(); ++it) {
        queue2.push(*it);
    }

    return queue2;
}

void program3() {
    int elementsNr;
    queue<Carte> queue1;

    system("cls");
    cout << "Introduceti numarul de elemente: ";
    cin >> elementsNr;
    pushDataInQueue(queue1, elementsNr);

    system("cls");
    cout << "Coada initiala:" << endl;
    showQueue(queue1);
    cout << endl;

    sortQueue(queue1);
    cout << "Coada sortata:" << endl;
    showQueue(queue1);
    cout << endl;

    Carte tmpCarte = findCarte(queue1);
    cout << "Elementul par este: " << endl << tmpCarte << endl;

    vector<Carte> *vct = new vector<Carte>();
    copyElements(queue1, *vct);
    cout << "Vectorul initial: " << endl;
    showVector(*vct);
    cout << endl;

    sortAsc(queue1, *vct);
    cout << "Coada sortata:" << endl;
```

```
    showQueue(queue1);
    cout << "Vectorul sortat:" << endl;
    showVector(*vct);
    cout << endl;

    queue<Carte> queue2 = concat(queue1, *vct);
    cout << "Coada + Vector :" << endl;
    showQueue(queue2);
    cout << endl;

    int length = 0;
       while (!queue2.empty())
       {
              if (numarPar(queue2.front()))
                     length++;
              queue2.pop();
       }

       if (!length)
              cout << "Nu sunt valori pare in coada" << endl;
       else
              cout << "Sunt " << length << " valori pare" << endl;

    _getch();
}

// Main
int main() {
    int command;
    while(true) {
        system("cls");
        cout << "     Menu:" << endl << endl;
        cout << "(1) Program 1" << endl;
        cout << "(2) Program 2" << endl;
        cout << "(3) Program 3" << endl << endl;
        cout << "(0) Iesire" << endl << endl;
        cout << "Commanda: " ;
        cin >> command;
        system("cls");
        switch (command) {
            case 0:
                exit(0);
                break;
            case 1:
                program1();
                break;
            case 2:
                program2();
                break;
            case 3:
                program3();
                break;
            default:
                cout << "Comanda gresita! Incearca din nou!" << endl;
                _getch();
                break;
        }
    }
    return 0;
}
```

Bibliografie:

1. Standard Template Library (STL) [Resursă electronică]. – Regim de access: http://www.infoarena.ro/stl
2. Standard Template Library (STL) [Resursă electronică]. – Regim de access: http://vega.unitbv.ro/~cataron/Courses/PCLPII/PCLP2_Capitolul9.pdf
3. Queue C++ [Resursă electronică]. – Regim de access: http://www.cplusplus.com/reference/queue/queue/
4. Vector C++ [Resursă electronică]. – Regim de access: http://www.cplusplus.com/reference/vector/vector/