Ministerul Educației al Republicii Moldova
Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică şi Microelectronică
Departamentul Inginerie Software si Automatica

RAPORT

Lucrare de laborator Nr.1
la Analiza, Ptoiectarea și Programarea Orientată pe Obiecte

A efectuat: st. gr. TI-142
Caldare Veaceslav

A verificat: lect. univ.
Alexandru Gavrisco

Chișinău 2017

Lucrare de laborator Nr.1

Tema: Standard Template Library

Scopul lucrării:

Stăpânirea tehnologiei de programare generică folosind Standard Template Library (STL) în limbajul C++.

Formularea condiției problemei (sarcina de lucru):

Scrieți 2 programe cu ajutorul STL. Prima trebuie să demonstreze lucrul cu containerele STL, a 2-a – utilizarea algoritmilor STL.

Varianta 1

Primul container	Al 2 container	Tipul implicit de date
vector	**list**	**int**

Note Teoretice:

Introducere

STL se bazeaza pe trei concepte centrale: containeri, iteratori si algoritmi. Ca tehnica de programare folosita, e bine sa stiti ca orientarea pe obiecte aproape ca lipseste. In schimb se utilizeaza din plin polimorfismul parametric. In C++ numele acestuia este "template"; in C# si Java se obisnuieste sa se spuna "generice".

Containeri

In limbajul de baza avem la dispozitie tablourile pentru reprezentarea unei secvente indexate de elemente. Dimensiunea unui tablou trebuie fie sa fie cunoscuta la compilare, fie sa fie gestionata explicit de programator prin alocare dinamica. Principalul avantaj al STL este ca ne scapa de aceasta grija. In afara de adresarea indexata obisnuita, secventele STL au si operatia *push_back*: adauga un element la sfarsit (si, evident, creste dimensiunea cu *1*). Exista si operatia inversa *pop_back*: elimina ultimul element. Cele trei secvente STL care suporta aceste operatii sunt *vector*, *deque* si *list*. Un exemplu de utilizare este:

```
1 // v are 10 elemente egale cu
2 0
3 vector <int> v(10);
  // acces indexat
4 v[1] = 2;
5 cout << v[0] << endl;
6 v.push_back(3);
```

```
7 cout << v[10] << endl;
8 v.pop_back();
```

Iteratori

Ganditi-va la algoritmul de gasire a maximului. El nu depinde de implementarea folosita pentru reprezentarea multimii! Tot ceea ce trebuie sa faci este sa accesezi toate elementele... nici macar nu conteaza ordinea. Ei bine, iteratorii permit o astfel de decuplare a structurilor de date de algoritmi.

Exemplu:

```
1 template <typename T>
2 typename T::value_type max(typename T::const_iterator begin, typename
3 T::const_iterator end) {
4     assert(begin != end); // container vid
5     typename T::const_iterator it;
6     typename T::value_type r = *begin;
7     for(it = begin; it != end; ++it)
8         if(*it > r)
9             r = *it;
10    return r;
}
```

1 Program

Am creat un vector, si am adaugat date conform variantei in cazul meu(date de tip int) si afisarea vectorului.

```
vector<int> vec;
vector <int>::iterator vec1;

//implimentez vectorul
for(int i=0; i<10; i++){
vec.push_back(i);
}

    cout <<"Primul Vector";
    cout <<endl;

//afisam vectorul
for(int i=0;i<vec.size();i++){

      cout<<vec[i]<<" ";
}
cout<<endl;
```

Creeam un vector de tip int, adaugam in el date cu ajutorul functiei push_back si il implem cu cifre pina la 10, apoi afisam vectorul prin for cu functia cout.

Stergea dintr-un vector se efectueaza cu ajutorul functiei erase(); cu indicarea parametrilor acestei functii care determina care elemente vor fi sterse, iar adaugarea inapoi a unor date efectuam cu ajutorul unui for cu functia push_back

```
vec.erase(vec.begin(),vec.begin()+2);
```

```
for(int i=0; i<2; i++){
vec.push_back(0);
```

Creeam al doilea vector si il implem la fel cu date de tip int ca in primul caz.

```
vector<int> vec2;
for( int i=0;i<10;i++){

vec2.push_back(i);

}
//afisez al e vector
cout<<" Vectorului 2";
cout<<endl;

for(int i=0;i<vec2.size();i++){

       cout<<vec2[i]<<" ";
       }
```

Stergem din primul vector un numar de elemente cerut de la tastatura iar apoi bagam in el elemtentele din al 2 vector.

```
vec.erase(vec.begin(),vec.begin()+n);
vec.insert(vec.begin(),vec2.begin(),vec2.end());
```

2 Program

In a 2 program functionalul este partial ca in prima programa deosebirea consta in faptul ca vectorii din a 2 programa nu sunt de tip implicit dar sunt de tip user, adica creeam o clasa aparte in cazul meu clasa Masini.

```
class Masini
{public:

string marca;
string model;
float  pret;

public:
       Masini ();
       Masini(string,string,float);
       Masini(const Masini&);
       void Input();
       void Show();
       friend iostream& operator>>(iostream&,Masini&);
       friend ostream&  operator<<(ostream&,Masini&);
       Masini& operator = (const Masini&);

};
```

Avem clasa Masini cu cele trei atribute ca marca, model si pret, totodata avem 3 contructori 2 functii friend si un overload de operatorul =.

```
Masini mas;
for(int i=0;i<n;i++){
cout<<"Introduceti datele vectorului";
cout<<endl;
       cin>>mas;
       car1.push_back(mas);

}
```

Creeam o variabila de tip Masini si prin for introducem datele in aceasta variabila iar apoi tot in acest for introducem datele din variabila in vector care la fel este de tip masini.

Ca urmare functionalul este fix ca in prima programa astfel nu se modifica nimic doar tipul de date cu care este implut vectorii.

3 Program

Pentru programul 3 am utilizat clasa deja creata Masini, mai adaugind functiile de care avem nevoie.

In primul rind am avut de creat un vector de tip Masini, care il introducem manual avind controlul asupra marimii vectorului si datelor din el.

```
cout << "Introduceti nr de elemente a vectorului ";
    cout << endl;
    cin >> n;
    //implem vectorul
    for (int i = 0; i < n; i++) {
        cout << "Introduceti vectorul ";
        cout << endl;
        cin >> mas;
        Vec1.push_back(mas);
           }
```

Apoi am avut de sortat descrescator acest vector, am ales ca elementul de referinta atributul pretul.

```
sort(Vec1.begin(), Vec1.end(), compareDescrescator());
```

Utilizam functia sort(); indicind inceputul, sfirsitul vectorului si functia predicat. Functia predicat se afla in clasa chemata.

```
class compareDescrescator {
public:
    bool operator()(Laptop& b1, Laptop& b2)
    {
        if (b1.price < b2.price)
            return false;
        else
            return true;
    }
};
```

Observam ca aceasta clasa contine doar o functie care returneaza o valoare bool in dependent de parametrii de intrare, care sunt obiectele clasei masini, se compara atributul pret si baza lui se returneaza valoarea true sau false.

Pentru afisarea vectorilor si listelor am creat 2 functii pentru a optimiza codul.

```
void PrintVector(string str, car d)
{
    vector<Masini>::iterator p = d.begin();
    cout << (str) << endl;
    if (d.empty())
        cout << ("\nVectorul este gol\n");
    else
        for (; p != d.end(); p++)
            cout << *p << endl;
}

void PrintList(string str, L1 s)
{

    cout << (str) << endl;
    list<Masini>::iterator p = s.begin();
    if (s.empty())
        cout << ("\nLista este goala\n");
    else
```

```
        for (; p != s.end(); p++)
            cout << *p << endl;
}
```

Observam ca pe linga afisare avem si o exceptie in cazul in care lista sau vectorul este gol. Afisarea are loc prin utilizarea iteratorilor.

Pentru gasirea unui element in vector apelam functi FindMarca cu 2 parametri, vectorul propriuzis si marca cautata.

```
cout << "Introduceti brandul cautat";
    cin >> brand;
    FindBrand(Vec1, brand);
```

Functia FindBrand cauta stringul adica marca necesara cu ajutorul functiei find_if care are ca predicat clasa comp_brand.

```
void FindMarca(car d, string s)
{
    cout << ("\nMasini de tip") << s << endl;
    iter j = d.begin();
    int i = 0;
    while (j != d.end()) {
        j = find_if(d.begin(), d.end(), comp_marca(s));
        if (j != d.end()) {
            cout << *j << endl;
            d.erase(j);
            i++;
        }
    }
    if (i == 0)
        cout << ("\nNu sunt asa tip de Masini\n");
}
```

Aceasta clasa are un constructor cu un parametru si o functie de tip bool care apeleaza functia GetBrand.

Urmatoarea conditie reprezinta mutarea din vector intr-o lista obiectele conform unei functii predicate.

```
Copy_if(Vec1, List1, comp_price(10000));
    Vec1.erase(std::remove_if(Vec1.begin(), Vec1.end(), comp_price(10000)),
        Vec1.end());
```

Astfel avem functia copy_if care efectueaza inserarea obiectelor conform unei functi predicate din vector intr-o lista, iar apoi aceste elemente le sterg din vector.

```
void Copy_if(car& d, L1& s, comp_price p)
{

    iter j;
    j = d.begin();
    while (j != d.end()) {
        if (p(*j)) {
            s.push_front(*j);

        }

        j++;
    };

}
```

Copierea are loc cu ajutorul unui iterator j carui ii indicam inceputul vectorului d, iar apoi prin while atita timp cit iteratorul este diferit de sfirsitul vectorului efectueaza conditia if (p(*j)) care returneaza true cu ajutorul functiei predicat comp_price.

Sortarea listei are loc cu ajutorul functiei Sort.

```
void Sort(L1& s)
{
    vector<Masini> d;
    while (!s.empty()) {
        d.push_back(s.front());
        s.pop_front();
    }
    sort(d.begin(), d.end(), compareCrescator());
    for (int i = 0; i < d.size(); i++)
        s.push_back(d[i]);
}
```

Sortarea se efectueaza in felul urmator, copiem lista intr-un vector, sortam vectorul iar apoi copiem inapoi datele in lista si astfel avem lista sortata.

Intr-o conditie este nevoie sa unim lista si vectorul intr-un al vector, pentru aceasta am creat o functie merge

```
car Merge(car& Vec, L1& List)
{
    car temp, ret;
    while (!List.empty()) {
        temp.push_back(List.front());
        List.pop_front();
    }
    ret.resize(Vec.size() + temp.size());
    merge(Vec.begin(), Vec.end(), temp.begin(), temp.end(), ret.begin());
    return ret;
}
```

Care creeaza in interiorul functiei 2 vectori, unul temp pentur copierea listei in el si un vector ret care se returneaza in urma efectuarii functiei, vectorul ret ii predefinim marimea care consta din marimea vectorului care vine ca parametru functiei + marimea vectorului temp, apoi cu ajutorul functiei merge predifinita in stl si obtinem vectorul ret.

Functia count_if numara obiectele care satisfac o oarecare conditie, in cazul dat este aceeasi functie predicat.

```
count_if(Vec2.begin(), Vec2.end(), comp_price(10000)) << endl;
```

Concluzie:

Realizând lucrarea de laborator nr. 1 am dedus importanța a Librăriei Standard de Șabloane, care reprezintă prin sine un mecanism interesant și puternic. Sunt greutăți și complicații în studierea lor și în primii pași al utilizării, dar ele se răscumpără, așa cum permit crearea unui cod mai universal și mai eficace, aceste trăsături îl ridică la același nivel cu restul limbajelor de nivel înalt. Mai mult ca atât, avem la dispoziție aproape orice metodă și instrument necesar pentru manipularea acestor containere ceea ce ne simplifică semnificativ munca.

Bibliografie:

1. www.stackoverflow.com/
2. www.wikipedia.org
3. www.cplusplus.com
4. www.youtube.com