Ministerul Educației al Republicii Moldova

Universitatea Tehnică a Moldovei

Catedra Automatică și Tehnologii Informaționale

RAPORT

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

A efectuat: st. gr. TI-142
I. Croitoru

A verificat: lect. univ.
Gavrisco Alexandr

Chișinău 2017

Lucrare de laborator Nr.2

Tema: Standard Template Library

Scopul lucrării:

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

Formularea condiției problemei (sarcina de lucru):

Scrieți 3 programe cu ajutorul STL. Prima trebuie să demonstreze lucrul cu containerele STL, a 2,3-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);
4 // acces indexat
5 v[1] = 2;
6 cout << v[0] << endl;
7 v.push_back(3);
8 cout << v[10] << endl;
  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:

```
template <typename T>
typename T::value_type max(typename T::const_iterator begin, typename
T::const_iterator end) {
    assert(begin != end); // container vid
    typename T::const_iterator it;
    typename T::value_type r = *begin;
    for(it = begin; it != end; ++it)
        if(*it > r)
            r = *it;
    return r;
}
```

1 Program

Crearea unui vector, adaugarea unor date conform variantei(date de tip int) si afisarea vectorului.

```
Vector<Integer> v = new Vector<>();
        for (int i=0; i<10; i++){
            v.addElement(i);
        }

        System.out.println("Vectorul initial este:");
        Enumeration e = v.elements();
      while (e.hasMoreElements()){
          System.out.print(e.nextElement() + " ");
      }
```

Creeam un vector de tip int, adaugam in el date cu ajutorul metodei addElement si il implem cu cifre random de la 0 pina la 9, apoi afisam vectorul cu ajutorul enumaratorului cu metodata print(next.emelent());

Stergerea din vector se efectueaza cu ajutorul metodei clear a unei sub liste cu 2 parametri, acesti 2 parametri indica range-ul de stergere, introducerea a elementelor noi se realizeaza iarasi cu ajutorul addElement();

```
v.subList(0,3).clear();
for (int i=0; i<3; i++){
              v.addElement(0);
}
```

Creeam al doilea vector si il implem la fel cu date de tip int insa deja vom avea cifre mari de la 100 la 500.

```
Vector<Integer> v2 = new Vector<>();
for (int i=1; i<=5; i++){
    v2.addElement(i*100);
}
```

Stergem din primul vector un numar de elemente cerut de la tastatura iar apoi bagam in el elementele din al 2 vector cu ajutorul metodei addAll.

```
  System.out.print("Introduceti numarul de elemente ce vor fi sterse");
       int numar = scanner.nextInt();
        v.subList(0,numar).clear();;
        v.addAll(v2);
```

2 Program

In a 2 aplicatie functionalul este ca in prima programa insa diferenta consta in faptul ca vectorii din a 2 programa nu sunt de tip implicit dar sunt de tip user, adica creeam o clasa Laptop cu 3 atribute.

```
public class Laptop implements Comparable<Laptop> {
    private  String brand;
    private String model;
    private  int price;
```

Avem clasa Laptop cu 3 atribute ca Brandul, modelul si pretul, totodata avem 2 contructori functii get, set, o fuctie de afisare si o functie de citire datelor de la tastatura.

```
Laptop() {
    brand = "Default";
    model="Default";
    price = 0;

}
Laptop(int Price, String Brand, String Model) {
    brand = Brand;
    model = Model;
    price = Price;
}
public void print() {
    System.out.println("Brand    : " + brand);
    System.out.println("Model  : " + model);
    System.out.println("Pret : " + price);
}
public void read() {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Introduceti brandul Laptopului    : ");
    brand = scanner.nextLine();
    System.out.print("Introduceti modelul Laptopului    : ");
    model = scanner.nextLine();
    System.out.print("Introduceti pretul Laptopului    : ");
    price = scanner.nextInt();
}
```

In al doilea program am creat 2 functii care ne usureaza lucrul cu vectorii, aceasta fiind introducerea datelor si afisarea lor.

```
private void AddVector(Vector<Laptop> data, int size){
    System.out.println("Introduceti" + size + " elemente: ");
    for(int i = 0; i < size; i++) {
        Laptop tmp = new Laptop();
        System.out.println("Introduceti Laptopul: ");
        tmp.read();
        data.addElement(tmp);
    }
}
```

Astfel aceasta metoda primeste 2 parametri vectorul creat de tip Laptop si variabila size care indica marimea acestui vector, deci obtinem o metodata destul de simpla de introducerea datelor de la tastatura.

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 Laptop, mai adaugind functiile de care avem nevoie.

In primul rind am avut de creat un vector de tip Laptop, care il introducem manual avind controlul asupra marimii vectorului si datelor din el, pentru aceasta utilizam deja metoda creeata AddVector();.

```
nrobj = scanner.nextInt();
AddVector(v, nrobj);
```

Apoi am avut de sortat descrescator acest vector, am ales ca elementul de referinta atributul pretul, pentru aceasta am utilizat metoda sort si am adaugat ca clasa noastra sa implementeze interfata Comparable<>.

```
Collections.sort(v);

@Override
public int compareTo(Laptop that) {
    return that.price - this.price;
}
```

Pentru a gasit un element in vectorul nostru de date indicind un atribut de exemplu marca laptopului am creat o functie FindBrand(); care are 2 parametri de intrare care reprezinta insasi vectorul in care cautam si marca sub forma de string cautata.

```
Vector<Laptop> tmpFind = FindBrand(v,brand);
 if (tmpFind!=null) {
     System.out.println("Elementele gasite:");
     for(Laptop t:tmpFind){
         System.out.println(t.toString());
     }

 }
private Vector<Laptop> FindBrand(Vector<Laptop> brand , String s){
      Vector<Laptop> result = new Vector<>();
       Iterator iter = brand.iterator();
        while(iter.hasNext()){
    Laptop temp = (Laptop) iter.next();
    if(temp.getBrand().equals(s))
        result.add(temp);
}
```

Observam ca aceasta functie proceseaza vectorul cu ajutorul iteratorului si fiecare element il verifica cu ajutorul metodei getBrand a elementului cu equals(); astfel vom adauga fiecare element cu marca cautata intr-un vetor result care mai apoi il afisam.

Creeam o lista si mutam in lista careva elemente, pentru acest scop am creat functia CopyEl();

```
List<Laptop> list = new ArrayList<Laptop>();
CopyEl(v, list);
```

Aceasta functie proceseaza vectorului si cu ajutorul unei conditii (functie boolean PretScump()) adauga elementele in lista daca aceste elemente indeplinesc conditia data, in cazul meu laptop-urile cu pret mai mare de 12000 lei sunt clasificate ca fiind laptop-uri scumple

```
for(int i= 0; i<v.size(); i++){
  if(PretScump(v.elementAt(i))){
      list.add(v.elementAt(i));}
}
```

Pentru sortare apelam clasa comparatorASC care are doar o singura metoda si compara laptopurile dupa nume aranjindule dupa alfabet adica crescator

```
v.sort(new comparatorASC());
list.sort(new comparatorASC());
```

Apoi am creat al 3 vector dupa conditie si am facut concatenarea celor 2 containere in el

```
Vector<Laptop> finalVec = concat(v, list);
System.out.println("Vector + lista :");
```

Am avut de numarat cite elemente satisfac o conditie, astfel am creat o functie count(); care are ca paramentru un vector de tip Laptop si o variabila contor care se incrementeaza odata ce if se valideaza si ne afiseaza variabila data.

```
count(finalVec);
private void count(Vector<Laptop> v){
    int count =0;
    for(int i = 0 ; i<v.size(); i++){
        if(PretScump(v.elementAt(i))){
           count+=1; }
    }
    System.out.print("Au fost gasite: "+count+" Laptop-uri scumpe");
    }
```

Concluzie:

Realizând lucrarea de laborator nr. 2 am dedus importanța a Librăriei Standart 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. https://www.tutorialspoint.com/java/util/java_util_arraylist.htm
3. https://www.tutorialspoint.com/java/java_vector_class.htm