БИБЛИОТЕКА СТАНДАРДНИХ ШАБЛОНА: ВЕКТОР, ИТЕРАТОР, ЛИСТА

Једно од каснијих проширења стандарда језика Це++ је библиотека стандардних шаблона (енгл. *STL – Standard Template Library*). STL је скуп апстрактних типова података, функција и алгоритама пројектованих да подрже типове података дефинисане од стране корисника. Сваки апстрактни тип такође садржи скуп функција и преклопљених оператора прилагођених за приступ конкретном типу податка. Идеја је да се имплементирају алгоритми и структуре података независне од конкретно употребљеног типа податка. На пример, апстрактни тип ***vector*** може се употребити за складиштење низа променљивих било ког типа података. Коришћење стандардних шаблона такође доноси и аутоматско заузимање потребне меморије за податке, као и механизме заштите од приступа ван опсега структуре.

Употреба стандардних шаблона упрошћава дизајн пројекта, у смислу да нема потребе за посебном имплементацијом структура као што су низови, редови, листе и слично. Осим тога, садржи и моћне алгоритме независне од типа података, укључујући ту сортирање и претраге. Те структуре уносе релативно мало успорења приликом извршавања, док са друге стране њихова употреба најчешће значајно смањује број грешака у програмском коду.

Иако логички није део библиотеке стандардних шаблона (јер није шаблон), у њу је укључена и ***string*** класа. Ова класа доноси знатна олакшања у раду са словним низовима.

Шаблони (енгл. *templates*) у Це++ језику представљају механизам који омогућује писање уопштеног (генеричког) кода који подразумева да код није непосредно зависан од коришћених типова података. Употребом механизма шаблона могуће је написати уопштене класе, функције и алгоритме који не зависе од коришћених типова података. Библиотека стандардних шаблона која садржи различите структуре података, итераторе и алгоритме као основ за имплеметацију истих користи механизам шаблона. Шаблон-класа дефинише се на следећи начин:

```
template<class type> class class-name {...};
```

Пример:

```
template<class A_Type> class Calc
{
  public:
    A_Type multiply(A_Type x, A_Type y);
    A_Type add(A_Type x, A_Type y);
};
```

```
template<class A_Type> A_Type Calc<A_Type>::multiply(A_Type x, A_Type
y) {
  return x*y;
}

template<class A_Type> A_Type Calc<A_Type>::add(A_Type x, A_Type y)
{
  return x+y;
}
```

STL Vector Class – шаблонска класа вектора

Једна од основних класа из STL-а је ***vector*** класа. У основи, вектор је низ променљиве величине, са омогућеним директним приступом преко оператора []. STL вектор дефинише се на следећи начин:

```
std::vector<class type> object_name;
```

Мора се обратити пажња да у оператор [], као и код приступа обичним низовима, не садржи контролу опсега. Међутим, вектор класа поседује методу ***at()***, која има ову контролу (али уз смањену брзину приступа). У наставку је дата листа неких од функција садржаних у класи вектор:

```
unsigned int size();          број елемената структуре
push_back(type element);      додаје елемент на крај вектора
bool empty();                 провера да ли је вектор празан
void clear();                 брише све елементе вектора
type at(int n);               враћа елемент на позицији n, са провером опсега
```

Такође, ту је и листа неколико основних оператора класе вектор:

=	Додела замењује елементе вектора са елементима другог вектора
==	Поређење два вектора елемент по елемент
[]	Приступ елементу вектора као код класичног низа, без провере опсега

Напомена: Пошто је STL део стандардног Це++ окружења, информације о класама (атрибути, методе, детаљи око имплементације, итд.) могу се наћи у документацији која прати Visual Studio окружење. Приступање тој помоћној документацији (тзв. „Help“) је брзо и једноставно: притиском на тастер F1. Корисна ствар коју окружење подржава је могућност да у свом програмском коду обележите део текста (име функције, променљиве, резервисане речи, итд.) у вези са којим желите сазнати више, и онда притиском на F1 ће се отворити део документације која се бави управо обележеним појмом. За вежбу, пронађите у помоћној документацији информације о вектор класи. Употреба помоћне документације у баратању са STL-ом (али и иначе) се изузетно препоручује. Немојте се устручавати да јој често приступате у потрази за одговорима.

Дат је једноставан пример употребе шаблон класе вектор.

```
#include <iostream>
#include <vector>

using namespace std;
int main()
{
    vector<int> example;          //Вектор за целобројни тип података
    example.push_back(3);         //Додај 3 на крају
    example.push_back(10);        //Додај 10 на крају
    example.push_back(33);        //Додај 33 на крају
    for (int x = 0; x < example.size(); x++)
    {
        cout << example[x] << " ";    //Испис на екрану: 3 10 33
    }
    if (!example.empty())          //Провера да ли није празан
        example.clear();           //Брише садржај
    vector<int> another_vector;    //Још један вектор целобројног типа
    another_vector.push_back(10);  //Додаје 10 на крај
    example.push_back(10);         //
    if (example == another_vector) //Провера једнакости два
    вектора {
        example.push_back(20);
    }
    for (int y = 0; y < example.size(); y++)
    {
        cout << example[y] << " ";    //Испис на екрану: 10 20
    }
    return 0;
}
```

Итератори библиотеке стандардних шаблона

Концепт итератора је фундаментални концепт STL-а, јер омогућава једнак начин приступа појединачним елементима различитих структура података као што су низови, листе, вектори, пресликавања, и тако даље. Постоји неколико типова итератора:

- Излазни итератор (*Output*): служи за упис, итерира унапред.
- Улазни итератор (*Input*): служи за читање, итерира унапред.
- Итератор унапред (*Forward*): итерира унапред, користи се за читање и упис.
- Бидирекциони (*Bidirectional*): итерира уназад и унапред, користи се за читање и упис, обезбеђен за класе *list*, *set*, *multiset*, *map*, и *multimap*.
- Итератор случајног приступа (*Random access*): омогућен приступ било којој позицији (не морају бити сукцесивне), користи се за читање и упис, обезбеђен за класе *vector*, *deque*, *string*, and *array*.

Свака структура има дефинисану функцију ***begin()***, која враћа итератор који показује на први елемент те структуре, као и функцију ***end()***, која враћа итератор еквивалентан итератору који је дошао на крај структуре (обратите пажњу да то **није** итератор који показује на крај структуре, у смислу да показује на последњи, тј. крајњи, елемент структуре, већ се може замислити као да показује „иза“ последњег елемента структуре). Елементу структуре се приступа преко дереференцирања итератора, као да је у питању обично показивач.

Итератор неке класе шаблона се дефинише на следећи начин:

```
std::class_name<template_parameters>::iterator name
```

где ***name*** представља име итератора, ***class_name*** је име шаблон класе за коју дефинишемо итератор (нпр. ***vector***), а ***template_parameters*** су параметри шаблона коришћени при дефиницији објекта са којем ће приступати итератор (нпр ***int***).

Пример дефиниције вектора и итератора за класу вектор над целобројним типом:

```
std::vector<int> myIntVector;
std::vector<int>::iterator myIntVectorIterator;
```

Постоји више различитих класа итератора, свака са незнатно другачијим својствима. Основна разлика је да ли се итератор користи за читање или упис елемената. Неки итератори дозвољавају обе операције, али не у исто време. Неки од најважнијих типова итератора су: за приступ елементима од првог ка последњем (forward), од последњег ка првом (backward) и двосмерни. За кретање напред може се користити оператор ++. Аналогно, за кретање у назад може се користити оператор --.

Укратко, за приступ елементима неке STL структуре, уместо класичног Це++ начина програмирања, најбоље је користити итераторе. Позивом ***begin()*** функције добија се итератор, употребом ++ оператора креће се кроз структуру, * се користи за приступ елементу, а крај итерација се детектује провером једнакости са итератором који враћа функција ***end()***.

За испис елемената вектора могла би се написати оваква функција:

```
using namespace std;

vector<int> intVect;
unsigned int i;

intVect.push_back(1);
intVect.push_back(4);
intVect.push_back(8);
```

```
for (i = 0; i < intVest.size(); i++)
{
    cout << intVect[i] << " ";
    //исписује на екран: 1 4 8
}
```

Иста та фукција би са употребом итератора изгледала овако:

```
using namespace std;

vector<int> intVect;
vector<int>::iterator intVectIt;

intVect.push_back(1);
intVect.push_back(4);
intVect.push_back(8);

for (intVectIt = intVect.begin(); intVectIt != intVect.end();
intVectIt++) {
    cout << *intVectIt << " ";
    //исписује на екран: 1 4 8
}
```

У објекат класе вектор могуће је и уметнути елемент на одређеној позицији. То се остварује на следећи начин:

```
intVect.insert(intVectIt, 5);
```

У том смислу, метода ***push_back*** је еквиалентна са:

```
intVect.insert(intVect.end(), 5);
```

Пошто се елементи вектора налазе редом један до другог у меморији, као код обичног низа, треба имати у виду да додавање елемента било где осим на крај вектора подразумева извршење додатног кода ради прерасподеле података у меморији.

Итератори су посебно погодни када је потребно дефинисати одређени опсег над којим се жели радити. Итератори, који раде над структурама које омогућавају директан приступ елементима као што су вектори, имају могућност померања за одређени број елемената у структури. Ова операција изводи се једноставним сабирањем итератора и константе. Резултат је итератор померен за константу у односу на тренутни елемент.

Напомена: Са овим треба бити пажљив јер се може прекорачити опсег структуре.

Тако на пример ако желимо да обришемо елементе из вектора, помоћу итератора можемо задати опсег елемената које желимо избрисати. Сваки елемент између два итератора ће бити обрисан. Брисање свих елемената је приказано у следећем примеру:

```
vector<int> myIntVector;
myIntVector.erase(myIntVector.begin(), myIntVector.end());
```

Ако желимо да обришемо само прва два елемента, то можемо урадити на следећи начин:

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

STL List Class – шаблонска класа листе

Класа листе у STL-у представља шаблонску изведбу двоструко спрегнуте листе. Декларација листе целобројних елемената изгледа овако:

```
std::list<int> intList;
```

Класа листе поседује методу ***push_back***, као и вектор, али поседује и ***push_front***, с обзиром да је у питању двоструко спрегнута листа.

Питање: Зашто вектор нема методу ***push_front***? (Одговор је имплицитно дат у наредном пасусу)

Класе вектора и листе се са становишта спреге према кориснику врло мало разликују (основна разлика је то што листа нема могућност директног приступа произвољном елементу, док се код вектора то постиже индексирањем). Њихова изведба, међутим, значајно је другачија. То се пре свега одражава на брзину извршења појединих акција над објектима тих класа. Тако на пример, иако се елементи у објекте обе класе умећу на исти начин:

```
intList.insert(intListIt, 5);
intVect.insert(intVectIt, 5);
```

брзина којом ће се те две акције извршити може се разликовати поприлично. Логично, уметање у листу је далеко брже, јер при уметању елемента у вектор, сви елементи на позицијама после позиције уметања се морају померити. Са друге стране, величина листе и вектора се проверава методом идентичног назива:

```
intList.size();
intVect.size();
```

али се провера величине листе обавља побројавањем њених елемената, док се код вектора своди на једноставну акцију. Због тога, на пример, није најповољније проверавати да ли је листа празна на следећи начин:

```
if (intList.size() == 0)
```

Уместо тога, боље је користити методу ***empty***, која ту проверу обавља далеко ефикасније.

```
if (intList.empty())
```

STL листа поседује још неколико корисних метода, као што су ***unique***, ***reverse***, и тако даље. Погледајте у помоћној документацији шта оне раде.

Испис свих елемената листе може се урадити на следећи начин:

```
using namespace std;

list<int> intList;
list<int>::iterator intListIt;

intList.push_back(1);
intList.push_back(4);
intList.push_back(8);

for (intListIt = intList.begin(); intListIt != intList.end();
intListIt++) {
    cout << *intListIt << " ";
    //исписује на екран: 1 4 8
}
```

Приметите да је разлика између исписа елемената вектора и листе коришћењем итератора само у класи која је декларисана. Због тога се, што се тиче неке основне функционалности, може лако прећи са употребе вектора на листу, или обрнуто; уколико се јави потреба за тим. Овде се јасније увиђа предност рада са итераторима. Међутим, предност није толико у томе што се може једноставније прећи са употребе једне класе на другу (то је случај са ове две класе, али не важи и за остале класе из STL-а, јер се њихова употреба логички знато разликује). Предност је у томе што су одређени поступци у својој форми еквивалентни и једном научени поступак је једноставно применљив и на остале класе из STL-а.

Вежбе

Отворити приложени пројекат и на основу примера из текста додати делове кода на места који су означени са @TODO.