**Міністерство Освіти І НАУКИ України**

**Національний університет "Львівська політехніка"**

Інститут **ІКНІ**

Кафедра **ПЗ**



### ЗВІТ

До лабораторної роботи № 11

**На тему:** *“* *Стандартна бібліотека шаблонів. Контейнери та алгоритми.”*

**З дисципліни:** *“Об’єктно-орієнтоване програмування”*

**Лектор:**

доцент каф.ПЗ

Коротєєва Т. О.

**Виконав:**

ст. гр. ПЗ-16

Шеремета А.І.

**Прийняв:**

асист. каф. ПЗ

Дивак І.В.

« \_\_\_\_ » \_\_\_\_\_\_\_\_ 2022 р.

∑= \_\_\_\_ .

Львів – 2022

**Тема роботи:** Стандартна бібліотека шаблонів. Контейнери та алгоритми.

**Мета** **роботи:** Навчитись використовувати контейнери стандартної бібліотеки шаблонів та вбудовані алгоритми.

**Теоретичні відомості**

Механізм шаблонів вбудований у компілятор C++, щоб дати можливість програмістам робити свій код коротше за рахунок узагальненого програмування. Звичайно, існують і стандартні бібліотеки, що реалізують цей механізм. STL є найефективнішою бібліотекою C++ на сьогоднішній день.

Стандартна бібліотека шаблонів це частина Стандартної бібліотеки С++, яка містить набір шаблонів контейнерних класів, алгоритмів та ітераторів. Спочатку вона була сторонньою розробкою, але пізніше була включена до Стандартної бібліотеки С++ . Якщо вам потрібен якийсь загальний клас чи алгоритм, то, швидше за все, у Стандартній бібліотеці шаблонів він уже є. Позитивним моментом є те, що ви можете використовувати ці класи без необхідності писати та налагоджувати їх самостійно. Крім того, ви отримуєте досить ефективні версії цих класів.

Послідовні контейнери - це контейнерні класи, елементи яких знаходяться в послідовності. Їхньою визначальною характеристикою є те, що ви можете додати свій елемент у будь-яке місце контейнера. Найбільш поширеним прикладом послідовного контейнера є масив: при додаванні 4 елементів масив, ці елементи будуть перебувати (в масиві) в точно такому ж порядку, в якому ви їх додали.

Починаючи з C++11, STL містить 6 контейнерів послідовності:

std::vector;

std::deque;

std::array;

std::list;

std::forward\_list;

std::basic\_string.

Клас vector - це динамічний масив, здатний збільшуватися при необхідності для утримання всіх своїх елементів. Клас vector забезпечує довільний доступ до своїх елементів через оператор індексації [], а також підтримує додавання та видалення елементів.

Клас deque - це двостороння черга, реалізована у вигляді динамічного масиву, який може рости з обох кінців.

List - це двозв'язний список, кожен елемент якого містить 2 вказівники: один вказує на наступний елемент списку, а інший - на попередній елемент списку. List надає доступ лише до початку та до кінця списку – довільний доступ заборонено. Якщо ви хочете знайти значення десь у середині, то ви повинні почати з одного кінця та перебирати кожен елемент списку доти, доки не знайдете те, що шукаєте. Перевагою двозв'язного списку є те, що додавання елементів відбувається дуже швидко, якщо ви, звичайно, знаєте, куди хочете додавати. Зазвичай, для перебору елементів двозв'язного списку використовуються ітератори.

Асоціативні контейнери – це контейнерні класи, які автоматично сортують усі свої елементи. За умовчанням, асоціативні контейнери виконують сортування елементів, використовуючи оператор порівняння <.

set - це контейнер, в якому зберігаються тільки унікальні елементи, і заборонені повторення. Елементи сортуються відповідно до їх значень.

multiset — це set, але в якому допускаються елементи, що повторюються.

map - це set, в якому кожен елемент є парою "ключ-значення". "Ключ" використовується для сортування та індексації даних і має бути унікальним, а "значення" - це фактичні дані.

multimap - це map, який допускає дублювання ключів. Всі ключі відсортовані в порядку зростання, і ви можете переглянути значення ключа.

Адаптери - це спеціальні зумовлені контейнерні класи, адаптовані для виконання конкретних завдань. Найцікавіше полягає в тому, що ви можете вибрати, який послідовний контейнер повинен використовувати адаптер.

stack - це контейнерний клас, елементи якого працюють за принципом LIFO (англ. "Last In, First Out" = "останнім прийшов, першим пішов"), тобто. елементи додаються (вносяться) у кінець контейнера і видаляються (виштовхуються) звідти ж (з кінця контейнера). Зазвичай у стеках використовується deque як послідовний контейнер за замовчуванням (що трохи дивно, оскільки vector був би більш підходящим варіантом), але ви також можете використовувати vector або list.

queue - це контейнерний клас, елементи якого працюють за принципом First In, First Out, тобто. елементи додаються (вносяться) у кінець контейнера, але видаляються з початку контейнера. За умовчанням у черзі використовується deque як послідовний контейнер, але також може використовуватися і list.

priority\_queue - це тип черги, де всі елементи відсортовані (за допомогою оператора порівняння <). При додаванні елемента він автоматично сортується. Елемент з найвищим пріоритетом (найбільший елемент) знаходиться на початку черги з пріоритетом, так само, як і видалення елементів виконується від початку черги з пріоритетом.

**Лабораторне завдання**

Написати програму з використанням бібліотеки STL.

В програмі реалізувати наступні функції:

1. Створити об’єкт-контейнер (1) у відповідності до індивідуального варіанту і заповнити його даними користувацього типу, згідно варіанту.

2. Вивести контейнер.

3. Змінити контейнер, видаливши з нього одні елементи і замінивши інші.

4. Проглянути контейнер, використовуючи для доступу до його елементів ітератори.

5. Створити другий контейнер цього ж класу і заповнити його даними того ж типу, що і перший контейнер.

6. Змінити перший контейнер, видаливши з нього n елементів після заданого і добавивши опісля в нього всі елементи із другого контейнера.

7. Вивести перший і другий контейнери.

8. Відсортувати контейнер по спаданню елементів та вивести результати.

9. Використовуючи необхідний алгоритм, знайти в контейнері елемент, який задовільняє заданій умові.

10. Перемістити елементи, що задовільняють умові в інший, попередньо пустий контейнер (2). Тип цього контейнера визначається згідно варіанту.

11. Проглянути другий контейнер.

13. Відсортувати перший і другий контейнери по зростанню елементів, вивести результати.

15. Отримати третій контейнер шляхом злиття перших двох.

16. Вивести на екран третій контейнер.

17. Підрахувати, скільки елементів, що задовільянють заданій умові, містить третій контейнер.

Оформити звіт до лабораторної роботи. Звіт має містити варіант завдання, код розробленої програми, результати роботи програми (скріншоти), висновок.

|  |  |  |  |
| --- | --- | --- | --- |
| **Варіант** | **Перший контейнер(1)** | **Другий контейнер(2)** | **Вбудований тип** |
| 10 | multimap | list | float |

**Хід роботи**

1. Реалізовую програму:

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <windows.h>

#include <QDebug>

#include <iterator>

#include <algorithm>

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(*parent*)

, ui(**new** Ui::MainWindow)

{

ui->setupUi(**this**);

}

MainWindow::~***MainWindow***()

{

**delete** ui;

}

std::multimap <*int*, *float*> FloatMultimap;

std::multimap <*int*, *float*> FloatMultimap2;

std::list <*float*> FloatList;

std::vector <*float*> FloatVector;

*void* MainWindow::**on\_Insert\_clicked**()

{

*int* count = 0;

*float* val = ui->textEdit->toPlainText().toFloat();

ui->Multimap->insertRow ( ui->Multimap->rowCount() );

ui->Multimap->setItem (ui->Multimap->rowCount()-1,

0,

**new** QTableWidgetItem*(ui->textEdit->toPlainText())*);

**if**(ui->Multimap->rowCount()<3){

FloatMultimap.insert(std::make\_pair(1, *val*));

}

**else** **if** (ui->Multimap->rowCount()>=3 && ui->Multimap->rowCount()<6){

FloatMultimap.insert(std::make\_pair(2, *val*));

}

**else** FloatMultimap.insert(std::make\_pair(3, *val*));

ui->textEdit->clear();

count++;

ui->textEdit->setFocus();

}

*void* MainWindow::**on\_DoStuff\_clicked**()

{

qDebug() << "1. Container 1 created";

qDebug() << "2. Output array";

qDebug() << "3. Item deleted";

std::multimap<*int*, *float*>::iterator it = FloatMultimap.begin();

FloatMultimap.erase(it);

qDebug() << "4. Iterator used";

**for**(it = FloatMultimap.begin(); it !=FloatMultimap.end();++it){

qDebug() << QString(QString::number((\*it).first));

**//just** **does** **something** **:)**

}

ui->Multimap->clear();

ui->Multimap->setRowCount(0);

**for**(**auto**& elm: FloatMultimap){

ui->Multimap->insertRow( ui->Multimap->rowCount() );

ui->Multimap->setItem(ui->Multimap->rowCount()-1,

0,

**new** QTableWidgetItem*(*QString*::number(elm.second))*);

}

**for**(**auto**& elm: FloatMultimap){

FloatMultimap2.insert(std::make\_pair(elm.first, *elm.second*));

}

qDebug() << "5. Second container filled";

*int* Nfrom6 = 2;

std::multimap<*int*, *float*>::iterator itToDeleteFrom = FloatMultimap.begin();

**for**(*int* i = 0; i < Nfrom6; i++){

++itToDeleteFrom;

}

FloatMultimap.erase(itToDeleteFrom, FloatMultimap.end());

**for**(**auto**& elm: FloatMultimap2){

FloatMultimap.insert(std::make\_pair(elm.first, *elm.second*));

}

qDebug() << "6. Deleted some n elem, inserted multimap2";

**for**(**auto** **const** &i: FloatMultimap2){

ui->Multimap\_2->insertRow( ui->Multimap\_2->rowCount() );

ui->Multimap\_2->setItem(ui->Multimap\_2->rowCount()-1,

0,

**new** QTableWidgetItem*(*QString*::number(i.second))*);

}

ui->Multimap->clear();

ui->Multimap->setRowCount(0);

**for**(**auto**& elm: FloatMultimap){

ui->Multimap->insertRow( ui->Multimap->rowCount() );

ui->Multimap->setItem(ui->Multimap->rowCount()-1,

0,

**new** QTableWidgetItem*(*QString*::number(elm.second))*);

}

ui->Multimap\_2->clear();

ui->Multimap\_2->setRowCount(0);

**for**(**auto**& elm: FloatMultimap2){

ui->Multimap\_2->insertRow( ui->Multimap\_2->rowCount() );

ui->Multimap\_2->setItem(ui->Multimap\_2->rowCount()-1,

0,

**new** QTableWidgetItem*(*QString*::number(elm.second))*);

}

qDebug() << "7.Done";

std::multimap<*int*, *float*, std::less<*int*>> FloatMultimapTemp;

**for**(**auto** &elm:FloatMultimap){

FloatMultimapTemp.insert(std::make\_pair(elm.first, *elm.second*));

}

FloatMultimap.clear();

**for**(**auto**&elm:FloatMultimapTemp){

FloatMultimap.insert(std::make\_pair(elm.first, *elm.second*));

}

qDebug() << "8. Sorted.";

ui->Multimap->clear();

ui->Multimap->setRowCount(0);

**for**(**auto**& elm: FloatMultimap){

ui->Multimap->insertRow( ui->Multimap->rowCount() );

ui->Multimap->setItem(ui->Multimap->rowCount()-1,

0,

**new** QTableWidgetItem*(*QString*::number(elm.second))*);

}

**auto** range = FloatMultimap.equal\_range(2);

qDebug() << "9. Algorithm used";

**for**(**auto** i = range.first; i != range.second; ++i){

std::list<*float*>::iterator count = FloatList.begin();

FloatList.push\_back(i->second);

count++;

}

**for**(**auto**& elm: FloatList){

ui->List->insertRow( ui->List->rowCount() );

ui->List->setItem(ui->List->rowCount()-1,

0,

**new** QTableWidgetItem*(*QString*::number(elm))*);

}

qDebug() << "10. Elements transported to List";

qDebug() << "11. Prohlyanuto?";

std::multimap<*float*, *float*> FloatMultimapTemp2;

**for**(**auto** &elm:FloatMultimap){

FloatMultimapTemp2.insert(std::make\_pair(*elm.second*, elm.first));

}

FloatMultimap.clear();

**for**(**auto**&elm:FloatMultimapTemp2){

FloatMultimap.insert(std::make\_pair(*elm.second*, elm.first));

}

ui->Multimap->clear();

ui->Multimap->setRowCount(0);

**for**(**auto**& elm: FloatMultimap){

ui->Multimap->insertRow( ui->Multimap->rowCount() );

ui->Multimap->setItem(ui->Multimap->rowCount()-1,

0,

**new** QTableWidgetItem*(*QString*::number(elm.second))*);

}

FloatList.sort();

ui->List->clear();

ui->List->setRowCount(0);

**for**(**auto**& elm: FloatList){

ui->List->insertRow( ui->List->rowCount() );

ui->List->setItem(ui->List->rowCount()-1,

0,

**new** QTableWidgetItem*(*QString*::number(elm))*);

}

qDebug() << "13.sorted";

**for**(**auto**& elm:FloatMultimap){

FloatVector.push\_back(elm.second);

}

**for**(**auto**& elm:FloatList){

FloatVector.push\_back(elm);

}

**for**(**auto**& elm: FloatVector){

ui->Third\_Container->insertRow( ui->Third\_Container->rowCount() );

ui->Third\_Container->setItem(ui->Third\_Container->rowCount()-1,

0,

**new** QTableWidgetItem*(*QString*::number(elm))*);

}

qDebug() << "15. Third container get.";

*int* ElemCount = 0;

**for**(**auto**& elm: FloatVector){

**if**(elm > 3){

ElemCount ++;

}

}

qDebug() << QString(QString("17. Done, count: ") + QString::number(ElemCount));

}

**Виконання лабораторної роботи**

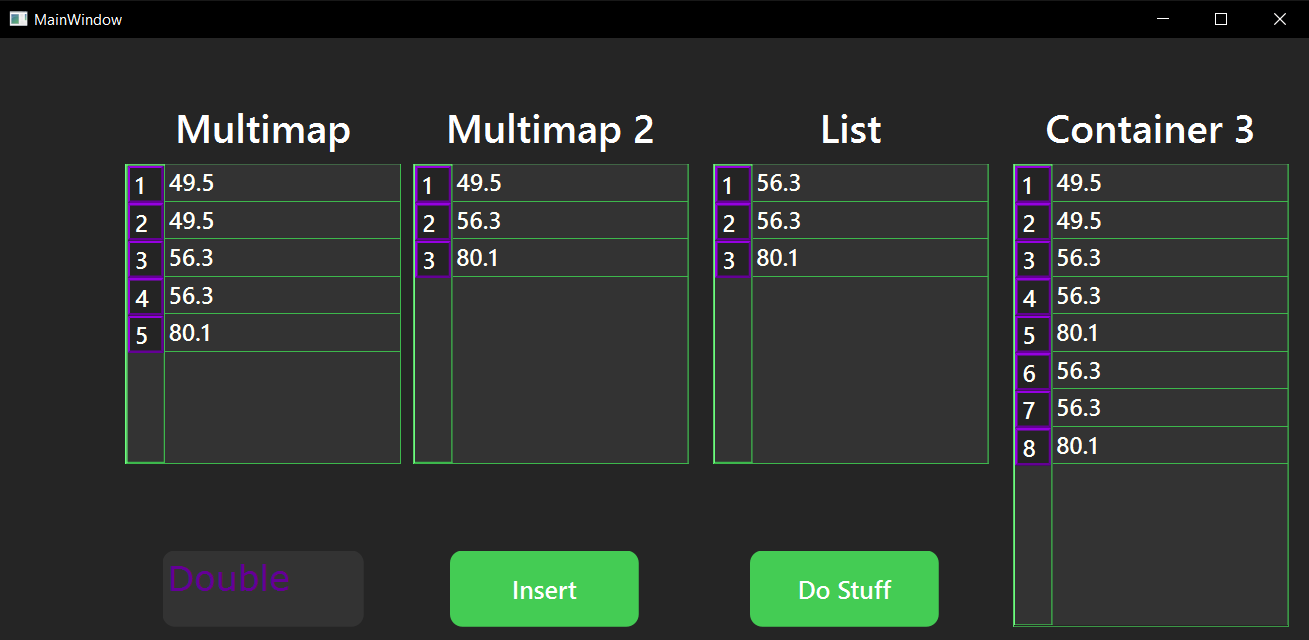


Рис. 1 Виконання програми

**Висновки**

На цій лабораторній роботі я навчився використовувати контейнери стандартної бібліотеки шаблонів та вбудовані алгоритми, використовував готові бібліотеку STL та алгоритми для роботи з контейнерами.