

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»**

**Інститут комп'ютерних наук та інформаційних технологій
Кафедра програмного забезпечення**



ЗВІТ

До лабораторної роботи № 11

На тему: «Стандартна бібліотека шаблонів. Контейнери та алгоритми»

З дисципліни: «Об'єктно-орієнтоване програмування»

Лектор:

доц. кафедри ПЗ
Коротєєва Т. О.

Виконав:

ст. гр. ПЗ-16
Чаус О. М.

Прийняв:

асист. кафедри ПЗ
Дивак І. В.

« ____ » _____ 2022 р.

Σ = ____

Тема роботи: Стандартна бібліотека шаблонів. Контейнери та алгоритми

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

Теоретичні відомості

Стандартна бібліотека шаблонів (скор. "STL" від "Standard Template Library") — це частина Стандартної бібліотеки C++, яка містить набір шаблонів контейнерних класів, алгоритмів та ітераторів. Задача STL полягає в тому, щоб звільнити програміста від чергового винаходу колеса. Бібліотеку STL можна багаторазово використовувати для розробки власних програм. Контейнери

Контейнер – це об'єкт, який включає інші об'єкти. Розрізняють три типи класів контейнерів бібліотеки STL :

- послідовні;
- асоціативні;
- адаптери.

Окремо розглядають поняття ітератора. Ітератор — це об'єкт, здатний перебирати елементи контейнерного класу. Ітератор можна порівняти з вказівником на певний елемент контейнерного класу. Ітератори завжди прив'язані до конкретних контейнерних класів. кожний контейнерний клас має 4 основні методи для роботи з оператором =:

метод `begin()` — повертає ітератор, який представляє початковий елемент контейнера;

метод `end()` — повертає ітератор, який представляє елемент, який знаходиться після останнього елемента в контейнері; це зроблено з метою спрощення використання циклів: цикл перебирає елементи до тих пір, поки ітератор не досягне методу `end()`.

метод `cbegin()` — повертає константний (тільки для читання) ітератор, який представляє початковий елемент контейнера;

метод `send()` — повертає константний (тільки для читання) ітератор, який представляє елемент, що знаходиться після останнього елемента в контейнері.

Контейнер Deque

Двостороння черга deque подібна двонаправленому вектору. Вона наслідує ефективність класу-контейнеру вектор по операціям послідовного читання та запису. Крім того клас-контейнер deque включає оптимізоване включення та видалення блоків з обох кінців черги.

Клас deque і асоційований з ним клас iterator містить усі інтерфейси методів, які містить клас vector і його клас iterator, а також два додаткові методи:

```
void push_front (const T& x);
```

Копія x вставляється в початок даної двосторонньої черги.

```
void pop_front ();
```

Елемент на початку даної двосторонньої черги видаляється.

Контейнер Multiset

У контейнері типу Multiset кожний елемент складається тільки із ключа, й допускаються повторювання елементів. Відповідно метод insert повертає ітератор, установлений на новий вставлений елемент. Оскільки може бути кілька копій елемента, у класі Multiset передбачений метод lower_bound, який повертає ітератор, установлений на першу позицію, у яку може бути вставлений елемент без порушення порядку проходження елементів у мультимножині:

```
iterator lower_bound (const T& x) const;
```

Аналогічно, метод upper_bound повертає ітератор, установлений на останній позиції, у яку може бути вставлений елемент без порушення порядку проходження елементів у мультимножині. Метод equal_range повертає пари ітераторів, установлених на першій й останній таких позиціях для даного елемента.

Контейнер algorithm

Алгоритми STL реалізовані у вигляді глобальних функцій, які працюють з використанням ітераторів. Це означає, що кожен алгоритм потрібно реалізувати всього лише один раз, і він працюватиме з усіма контейнерами, які надають набір ітераторів (включаючи і ваші власні (користувацькі) контейнерні класи).

1. Алгоритми min_element() і max_element() знаходять мінімальний і максимальний елементи в контейнері
2. алгоритм find(), щоб знайти певне значення в списку.
3. Алгоритми sort() і reverse().

Завдання для лабораторної роботи

В програмі реалізувати наступні функції:

1. Створити об'єкт-контейнер (1) у відповідності до індивідуального варіанту і заповнити його даними користувацького 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 задовільняють заданій умові, містить третій контейнер.

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

Хід роботи

Файл **mainwindow.h**:

```
...
private:
    Ui::MainWindow *ui;
    std::deque<char> m_deque;
    std::multiset<char> m_multiset;
    std::multiset<char> m_newMultiset;
```

Файл **mainwindow.cpp**:

```
#include "mainwindow.h"
#include "../ui_mainwindow.h"
#include <QMessageBox>
#include <algorithm>

QString currentText = "";

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    ui->multisetTable->setEditTriggers(QAbstractItemView::NoEditTriggers);
    ui->multiset2Table->setEditTriggers(QAbstractItemView::NoEditTriggers);
    ui->label_3->setVisible(false);
    ui->label_4->setVisible(false);
    QRegularExpressionValidator *validator = new
QRegularExpressionValidator(QRegularExpression("^.{1}$"));
    ui->lineEdit->setValidator(validator);
    QIntValidator *validator1 = new QIntValidator(0, 100);
    ui->lineEdit_2->setValidator(validator1);
    srand(time(NULL));
    std::size_t dequeSize = rand() % 10 + 1;
    std::size_t multisetSize = rand() % 10 + 1;
    for(std::size_t i = 0; i < dequeSize; i++)
        m_deque.push_back((rand() % 93 + 33));
    for(std::size_t i = 0; i < multisetSize; i++)
        m_multiset.insert((rand() % 93 + 33));
    update();
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::update()
{
    ui->dequeTable->setColumnCount(m_deque.size());
    ui->dequeTable->resize(ui->dequeTable->columnCount() * 70, ui->dequeTable->height());
    int i = 0;
    ui->dequeTable->blockSignals(true);
    for(std::deque<char>::iterator it = m_deque.begin(); it != m_deque.end(); it++)
    {
        QTableWidgetItem *item = new QTableWidgetItem();
        item->setText(QChar::fromLatin1(*it));
        ui->dequeTable->setItem(0, i, item);
        i++;
    }
    ui->dequeTable->blockSignals(false);
    i = 0;
    ui->multisetTable->setColumnCount(m_multiset.size());
```

```

        ui->multisetTable->resize(ui->multisetTable->columnCount() * 70, ui->multisetTable-
>height());
        for(std::multiset<char>::iterator it = m_multiset.begin(); it != m_multiset.end();
it++)
        {
            QTableWidgetItem *item = new QTableWidgetItem();
            item->setText(QChar::fromLatin1(*it));
            ui->multisetTable->setItem(0, i, item);
            i++;
        }
        ui->multiset2Table->setColumnCount(m_newMultiset.size());
        ui->multiset2Table->resize(ui->multiset2Table->columnCount() * 70, ui-
>multiset2Table->height());
        i = 0;
        for(std::multiset<char>::iterator it = m_newMultiset.begin(); it !=
m_newMultiset.end(); it++)
        {
            QTableWidgetItem *item = new QTableWidgetItem();
            item->setText(QChar::fromLatin1(*it));
            ui->multiset2Table->setItem(0, i, item);
            i++;
        }
        if(ui->dequeTable->width() > ui->multisetTable->width() && ui->dequeTable->width() >
this->width())
            this->resize(ui->dequeTable->width() + 20, this->height());
        else if (ui->multisetTable->width() > this->width())
            this->resize(ui->multisetTable->width() + 20, this->height());
    }

void MainWindow::on_dequeTable_cellClicked(int row, int column)
{
    currentText = ui->dequeTable->item(row, column)->text();
}

void MainWindow::on_dequeTable_cellChanged(int row, int column)
{
    ui->dequeTable->blockSignals(true);
    for(int i = 0; i < ui->dequeTable->columnCount(); i++)
        if(ui->dequeTable->item(0, i)->text().length() == 1)
        {
            if(ui->dequeTable->item(0, i)->text() != m_deque[i])
                m_deque[i] = ui->dequeTable->item(0, i)->text().at(0).toLatin1();
        }
        else
        {
            QMessageBox::critical(this, "Error", "Please check if your input is correct",
QMessageBox::Ok);
        }
    update();
    ui->dequeTable->blockSignals(false);
}

void MainWindow::on_deleteButton_clicked()
{
    if(m_deque.size())
    {
        ui->dequeTable->blockSignals(true);
        if(auto it = std::find(m_deque.begin(), m_deque.end(), ui->dequeTable-
>currentItem()->text()[0]); it != m_deque.end())
            m_deque.erase(it);
        ui->dequeTable->blockSignals(false);
        update();
    }
}

```

```

void MainWindow::on_addButton_clicked()
{
    if(!ui->lineEdit->text().isEmpty())
    {
        m_deque.push_back(ui->lineEdit->text().at(0).toLatin1());
        update();
    }
}

void MainWindow::on_deleteNButton_clicked()
{
    if(!ui->lineEdit_2->text().isEmpty() && (ui->dequeTable->currentColumn() + ui->
lineEdit_2->text().toInt()) < ui->dequeTable->columnCount())
    {
        std::deque<char>::iterator startPoint = std::find(m_deque.begin(), m_deque.end(),
ui->dequeTable->currentItem()->text()[0]);
        m_deque.erase(startPoint, startPoint + ui->lineEdit_2->text().toInt() + 1);
        std::copy(m_multiset.begin(), m_multiset.end(), std::back_inserter(m_deque));
        update();
    }
}

void MainWindow::on_clearButton_clicked()
{
    m_multiset.clear();
    update();
}

void MainWindow::on_desSortButton_clicked()
{
    std::sort(m_deque.begin(), m_deque.end(), [](char a, char b) {
        return a > b;
    });
    update();
}

void MainWindow::on_ascSortButton_clicked()
{
    std::sort(m_deque.begin(), m_deque.end());
    update();
}

void MainWindow::on_mergeButton_clicked()
{
    m_newMultiset.clear();
    std::copy(m_deque.begin(), m_deque.end(), std::inserter(m_newMultiset,
m_newMultiset.end()));
    std::copy(m_multiset.begin(), m_multiset.end(), std::inserter(m_newMultiset,
m_newMultiset.end()));
    if(m_newMultiset.size())
    {
        ui->label_3->setVisible(true);
        ui->label_4->setVisible(true);
        ui->label_4->setText("Vowels: " +
QString::number(std::count_if(m_newMultiset.begin(), m_newMultiset.end(), [](char value){
    return (value == 'a' || value == 'e' || value == 'i' || value == 'o' || value
== 'u' ||
        value == 'A' || value == 'E' || value == 'I' || value == 'O' || value
== 'U' ));
    }));
    update();
}
}

```

```

void MainWindow::on_numCopyButton_clicked()
{
    std::copy_if(m_deque.begin(), m_deque.end(), std::inserter(m_multiset,
m_multiset.end()), [](char value) {
        return isdigit(value);
    });
    update();
}

```

Deque

Multiset

5	:	A	U	V	~
---	---	---	---	---	---

New Multiset

5	:	A	U	V	~	x
---	---	---	---	---	---	---

Vowels: 2

Зображення програми

Висновок: навчився використовувати контейнери стандартної бібліотеки шаблонів та вбудовані алгоритми.