МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Інститут **КНІТ**
Кафедра **ПЗ**

ЗВІТ

До лабораторної роботи № 7
На тему: "*Робота з динамічною пам'яттю*"
З дисципліни: "Об'єктно-орієнтоване програмування"

Лектор:
доцент кафедри ПЗ
Коротєєва Т.О.

Виконав:
студент групи ПЗ-16
Коваленко Д.М.

Прийняв:
доцент кафедри ПЗ
Яцишин С.І.

«_____» _____ 2022 р.
$\sum$ = _____.....

Львів — 2022

Тема. Робота з динамічною пам'яттю.

Мета. Навчитися виділяти місце під об'єкти динамічно. Навчитися створювати та використовувати конструктор копіювання, перевантажувати оператор присвоєння. Ознайомитися з принципами створення та функціонування деструкторів.

Лабораторне завдання

1. Переглянути лістинг коду в прикладі. Пояснити вивід програми
2. Створити клас відповідно до завдання
3. Розробити для класу конструктор за замовчуванням та декілька звичайних конструкторів. Реалізувати функції-члени відповідно до завдання
4. Створити конструктор копіювання
5. Перевантажити операцію присвоєння
6. Створити деструктор для вивільнення динамічно виділеної пам'яті
7. Об'єкти класу розмістити в динамічній пам'яті
8. Продемонструвати розроблені можливості класу завдяки створеному віконному застосуванню
9. Оформити звіт до лабораторної роботи

Індивідуальне завдання

Клас Deque – двонаправлена черга. Пам'ять під елементи черги повинна виділятися динамічно. Реалізувати такі функції члени:
Отримання кількості елементів у черзі.
Знаходження максимального значення.
Знаходження мінімального значення.
Знаходження середнього арифметичного значення черги.
Очищення черги.
Перевірка, чи черга порожня.
Перевантажити операції. При цьому вибір механізму перевантаження обрати самостійно (чи метод, чи дружня-функція):
Додавання зліва (почленне додавання елементів до черги)
Додавання справа (почленне додавання елементів до черги)
Віднімання зліва (почленне видалення елементів з черги)
Віднімання справа (почленне видалення елементів з черги)
Множення на скаляр.
Введення черги з StringGrid (»)
Виведення черги у StringGrid («)
Введення черги з ListBox (»)
Виведення черги у ListBox («)
Виведення черги у Memo («)

Теоретичні відомості

Кожна змінна чи константа програми розміщується в адресному просторі програми в одному з видів пам'яті: статичній, локальній (стек) чи динамічній.

В статичній пам'яті розміщуються глобальні змінні (оголошені поза всіма блоками – функцією, методом, класом) і статичні змінні (перед типом яких вказується ключове слово static, при цьому змінна може знаходиться де завгодно, в тому числі і в тілі функції, методу чи класу). Різниця між статичною та глобальною змінними проявляється, коли програма складається з декількох файлів: глобальні змінні доступні в будь-яких файлах вихідного коду, а статичні – тільки в тому файлі, де були оголошені. В статичній пам'яті не рекомендується тримати великі об'єкти (наприклад, масиви), а хорошим кодом з використанням ООП вважається програмний код, в якому використання глобальних і статичних змінних зведено до мінімуму. Локальна пам'ять або стек – частина адресного простору програми, де розміщуються

змінні функцій та методів. Пам'ять для них виділяється при вході в блок програми і вивільняється при виході з нього.

Динамічна пам'ять – решта адресного простору програми, де можуть бути розміщені дані. Вона виділяється і вивільняється за допомогою спеціальних інструкцій, які може використовувати розробник ПЗ. Це дозволяє в ході виконання програми контролювати і коригувати об'єм використовуваної пам'яті і, відповідно, створювати програми, котрі можуть опрацьовувати великі об'єми даних, обходячи обмеженість розміру реально доступної фізичної пам'яті.

Доступ до динамічної пам'яті можливий тільки через вказівники, які програміст може зв'язувати з виділеною ділянкою пам'яті. Динамічна пам'ять в мові C++ виділяється за допомогою оператора new і звільняється за допомогою оператора delete. Можна також використовувати с-функції, такі як alloc, malloc, calloc, realloc для виділення/перевиділення пам'яті і відповідну їм функцію free для звільнення виділеної пам'яті. Проте специфіка роботи оператора new і наведених вище функцій відрізняється. Тому не можна змішувати виклики оператора new і функції free, чи навпаки функції malloc, наприклад, і оператора delete. Якщо не звільняти виділену динамічну пам'ять, то вона буде зайнята до закінчення програми, що зменшує доступний обсяг вільної пам'яті і може призводити до некоректної роботи програми чи до її непередбачуваного завершення. Тому завжди, як тільки виділена пам'ять стає непотрібною, її необхідно звільняти.

Код програми

Назва файлу: *deque.h*

```
#ifndef DEQUE_H
#define DEQUE_H

#include "ui_mainwindow.h"
#include "QTableWidget"
#include "QListWidget"
#include "QLineEdit"

class Node {
    private:
    int value;
    Node* prev;
    Node* next;
    public:
    Node(int v):
    value(v),
    prev(nullptr),
    next(nullptr)
    {};
    int getValue() { return value; };
    Node* getPrev() { return prev; };
    Node* getNext() { return next; };
    void setValue(int v) { value = v; };
    void setPrev(Node* p) { prev = p; };
    void setNext(Node* n) { next = n; };
};

class Deque {
    private:
    Node* head;
    Node* tail;
    public:
    Deque():
    head(nullptr),
    tail(nullptr)
    {};
    Deque(Node* head, Node* tail) {
        this->head = head;
```

```
        this->tail = tail;
    };
    Deque(const Deque& d) { head = d.head; tail = d.tail; };
    ~Deque();
    void display() const;
    void clear();
    void multiply(int n);
    bool isEmpty() const;
    int countValues() const;
    int maxValue() const;
    int minValue() const;
    double avgValue() const;

    void operator<(int value); // pushFront
    void operator>(int value); // pushBack
    int operator--();          // prefix popFront
    int operator--(int);       // postfix popBack
    void operator *=(int n);
    Deque* operator=(Deque& d);

    void operator>>(QTableWidget* table);
    void operator<<(QTableWidget* table);
    void operator>>(QListWidget* list);
    void operator<<(QListWidget* list);
    void operator>>(QLineEdit* line);
    void operator<<(QLineEdit* line);
};

#endif // DEQUE_H
```

Назва файлу: *deque.cpp*

```
#include "deque.h"
#include <iostream>
#include <string>
#include <sstream>

using namespace std;

Deque::~Deque() {
    Node* head = this->head;
    while (head != nullptr) {
        Node* tmp = head;
        head = head->getNext();
        free(tmp);
    }
    this->head = this->tail = nullptr;
}

bool Deque::isEmpty() const {
    return head == nullptr && tail == nullptr;
}

void Deque::display() const {
    Node* head = this->head;
    while (head != nullptr) {
        cout << head->getValue() << " ";
        head = head->getNext();
    }
    cout << endl;
    Node* tail = this->tail;
```

```
    while (tail != nullptr) {
        cout << tail->getValue() << " ";
        tail = tail->getPrev();
    }
    cout << endl;
}

void Deque::clear() {
    Deque::~Deque();
}

int Deque::countValues() const {
    Node* head = this->head;
    int count = 0;
    while (head != nullptr) {
        count++;
        head = head->getNext();
    }
    return count;
}

int Deque::maxValue() const {
    Node* head = this->head;
    int max = INT32_MIN;
    while (head != nullptr) {
        if (head->getValue() > max) max = head->getValue();
        head = head->getNext();
    }
    return max;
}

int Deque::minValue() const {
    Node* head = this->head;
    int min = INT32_MAX;
    while (head != nullptr) {
        if (head->getValue() < min) min = head->getValue();
        head = head->getNext();
    }
    return min;
}

double Deque::avgValue() const {
    Node* head = this->head;
    if (this->countValues() == 0) return 0.f;
    double sum = 0;
    while (head != nullptr) {
        sum += head->getValue();
        head = head->getNext();
    }
    return sum / this->countValues();
}

void Deque::multiply(int n) {
    Node* head = this->head;
    while (head != nullptr) {
        head->setValue(head->getValue() * n);
        head = head->getNext();
    }
}

void Deque::operator<(int value) {
```

```
    Node* node = (Node*)malloc(sizeof(Node));
    if (node == nullptr) return;
    node = new(node) Node(value);
    if (head == nullptr && tail == nullptr) {
        head = node;
        tail = head;
        return;
    }
    Node* tmp = head;
    head = node;
    head->setNext(tmp);
    tmp->setPrev(head);
}

void Deque::operator>(int value) {
    if (head == nullptr && tail == nullptr) {
        Node* node = (Node*)malloc(sizeof(Node));
        if (node == nullptr) return;
        tail = new(node) Node(value);
        head = tail;
        return;
    }
    Node* tmp = tail;
    Node* node = (Node*)malloc(sizeof(Node));
    if (node == nullptr) return;
    tail->setNext(new(node) Node(value));
    tail = tail->getNext();
    tail->setPrev(tmp);
}

int Deque::operator--() {
    if (head == nullptr) return INT32_MIN;
    Node* tmp = head;
    int value = tmp->getValue();
    head = head->getNext();
    head->setPrev(nullptr);
    free(tmp);
    return value;
}

int Deque::operator--(int) {
    if (tail == nullptr) return INT32_MIN;
    Node* tmp = tail;
    int value = tmp->getValue();
    tail = tail->getPrev();
    tail->setNext(nullptr);
    free(tmp);
    return value;
}

void Deque::operator*=(int n) {
    Node* head = this->head;
    while (head != nullptr) {
        head->setValue(head->getValue() * n);
        head = head->getNext();
    }
}

Deque* Deque::operator=(Deque& d) {
    Deque* deq = new Deque(d.head, d.tail);
    return deq;
```

```
}

void Deque::operator>>(QTableWidget* table) {
    Node* head = this->head;
    int i = 0;
    table->setColumnCount(0);
    while (head != nullptr) {
        table->insertColumn(i);
        QTableWidgetItem* item = new QTableWidgetItem();
        item->setText(QString::number(head->getValue()));
        table->setItem(0, i, item);
        i += 1;
        head = head->getNext();
    }
}

void Deque::operator<<(QTableWidget* table) {
    for (int i = 0; i < table->columnCount(); i++) {
        *this < table->item(0, i)->text().toInt();
    }
}

void Deque::operator>>(QListWidget* list) {
    Node* head = this->head;
    list->clear();
    while (head != nullptr) {
        QListWidgetItem* item = new QListWidgetItem();
        item->setFlags(item->flags() | Qt::ItemIsEditable);
        item->setText(QString::number(head->getValue()));
        list->addItem(item);
        head = head->getNext();
    }
}

void Deque::operator<<(QListWidget* list) {
    for (int i = 0; i < list->count(); i++) {
        *this < list->item(i)->text().toInt();
    }
}

void Deque::operator>>(QLineEdit* line) {
    Node* head = this->head;
    line->clear();
    while (head != nullptr) {
        line->setText(line->text() + " " + QString::number(head->getValue()));
        head = head->getNext();
    }
}

void Deque::operator<<(QLineEdit *line) {
    stringstream ss(line->text().toStdString());
    string n;
    while (ss >> n) {
        *this < QString::fromStdString(n).toInt();
    }
}
```

Назва файлу: *main.cpp*

```
#include "mainwindow.h"
```

```
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```

Назва файлу: *mainwindow.cpp*

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

#include "deque.h"
#include <iostream>

using namespace std;

Deque d = Deque();

MainWindow::MainWindow(QWidget *parent)
: QMainWindow(parent)
, ui(new Ui::MainWindow) {
    ui->setupUi(this);
}

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

void MainWindow::on_clearDequeButton_clicked() {
    d.clear();
}

void MainWindow::on_addColPushButton_clicked() {
    ui->tableWidget->setColumnCount(ui->tableWidget->columnCount() + 1);
}

void MainWindow::on_rmColPushButton_clicked() {
    ui->tableWidget->removeColumn(ui->tableWidget->columnCount() - 1);
}

void MainWindow::on_addRowPushButton_clicked() {
    QListWidgetItem* item = new QListWidgetItem();
    item->setFlags(item->flags() | Qt::ItemIsEditable);
    ui->listWidget->insertItem(ui->listWidget->count(), item);
}

void MainWindow::on_rmRowPushButton_clicked() {
    ui->listWidget->takeItem(ui->listWidget->count() - 1);
}

void MainWindow::on_getTablePushButton_clicked() {
    d >> ui->tableWidget;
}

void MainWindow::on_setTablePushButton_clicked() {
    d.clear();
    d << ui->tableWidget;
```

```
}

void MainWindow::on_getListPushButton_clicked() {
    d >> ui->listWidget;
}

void MainWindow::on_setListPushButton_clicked() {
    d.clear();
    d << ui->listWidget;
}

void MainWindow::on_getLinePushButton_clicked() {
    d >> ui->lineEdit;
}

void MainWindow::on_setLinePushButton_clicked() {
    d.clear();
    d << ui->lineEdit;
}

void MainWindow::on_minPushButton_clicked() {
    ui->minResLineEdit->setText(QString::number(d.minValue()));
}

void MainWindow::on_maxPushButton_clicked() {
    ui->maxResLineEdit->setText(QString::number(d.maxValue()));
}

void MainWindow::on_avgPushButton_clicked() {
    ui->avgLineEdit->setText(QString::number(d.avgValue()));
}

void MainWindow::on_multiplyPushButton_clicked() {
    d.multiply(ui->multiplyLineEdit->text().toInt());
}

void MainWindow::on_nPushButton_clicked() {
    ui->nResLineEdit->setText(QString::number(d.countValues()));
}

void MainWindow::on_emptyPushButton_clicked() {
    ui->emptyResLineEdit->setText(d.isEmpty() ? "true" : "false");
}
```

Назва файлу: *mainwindow.h*

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

    public:
    MainWindow(QWidget *parent = nullptr);
```

```
    ~MainWindow();

    private slots:
    void on_clearDequeButton_clicked();
    void on_getTablePushButton_clicked();
    void on_setTablePushButton_clicked();
    void on_addColPushButton_clicked();
    void on_rmColPushButton_clicked();
    void on_getListPushButton_clicked();
    void on_setListPushButton_clicked();
    void on_getLinePushButton_clicked();
    void on_setLinePushButton_clicked();
    void on_addRowPushButton_clicked();
    void on_rmRowPushButton_clicked();
    void on_minPushButton_clicked();
    void on_maxPushButton_clicked();
    void on_avgPushButton_clicked();
    void on_multiplyPushButton_clicked();
    void on_nPushButton_clicked();
    void on_emptyPushButton_clicked();

    private:
    Ui::MainWindow *ui;
};
#endif // MAINWINDOW_H
```

Висновок

На лабораторній роботі я навчився виділяти місце під об'єкти динамічно. Навчився створювати та використовувати конструктор копіювання, перевантажувати оператор присвоєння. Ознайомився з принципами створення та функціонування деструкторів.