Міністерство освіти і науки України

Національний університет “Львівська політехніка”

Інститут комп’ютерних наук та інформаційних технологій

Кафедра програмного забезпечення

**Звіт**

Про виконання лабораторної роботи №7

**На тему:**

«Робота з динамічною пам’яттю»

з дисципліни

«Об’єктно-орієнтоване програмування»

**Лектор:**

Доцент каф. ПЗ

Коротєєва Т. О.

**Виконав:**

ст. гр. ПЗ-11

Дзюба В. Є.

**Прийняла:**

Доцент каф. ПЗ

Коротєєва Т. О.

« \_\_ » \_\_\_\_\_\_\_\_ 2022 р.

∑ = \_\_\_\_\_ \_\_\_\_\_\_\_\_\_ .

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.

**Теоретичні відомості**

В мові С++ існує декілька основних **типів пам’яті**.

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.

**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. 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 С++ виділяється за допомогою оператора **new** і звільняється за допомогою 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.

**Завдання. Варіант №9**

1. Створити клас **Deque**.

2. Розробити для класу конструктор за замовчуванням та декілька звичайних конструкторів. Реалізувати функції-члени відповідно до завдання:

 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 TableWidget (>>)

 Виведення черги у TableWidget (<<)

 Введення черги з ListWidget (>>)

 Виведення черги у ListWidget (<<)

 Виведення черги у TextEdit (<<)

4. Створити конструктор копіювання.

5. Перевантажити операцію присвоєння.

6. Створити деструктор для вивільнення динамічно виділеної пам’яті.

**Хід роботи**

**Код програми:**

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\_AddItemButton\_clicked**();

void **on\_AddRowButton\_clicked**();

void **on\_ShowOnTextEdit\_clicked**();

void **on\_ShowOnTextEdit\_2\_clicked**();

void **on\_ClearButton\_clicked**();

void **on\_MaxButton\_clicked**();

void **on\_MinButton\_clicked**();

void **on\_AverageButton\_clicked**();

void **on\_MultiplyButton\_clicked**();

void **on\_delete\_1st\_clicked**();

void **on\_delete\_2nd\_clicked**();

void **on\_Add\_1st\_clicked**();

void **on\_Add\_last\_clicked**();

private:

Ui::MainWindow \*ui;

};

#endif // MAINWINDOW\_H

Deque.h:

#ifndef DEQUE\_H

#define DEQUE\_H

#include <QTableWidget>

#include <QListWidget>

#include <QTextEdit>

#include <QLabel>

class **Node** {

public:

double num;

Node \*pNext;

Node \*pPrev;

**Node**();

**Node**(double i);

**Node**(double i, Node\* pNext, Node\* pPrev);

};

class **Deque** {

private:

Node\* pHead;

Node\* pTail;

public:

**Deque**();

**Deque**(const Deque &);

**Deque**(double i);

~**Deque**();

int **GetSize**();

double **GetMin**();

double **GetMax**();

double **GetAverage**();

void **FreeList**();

bool **IsEmpty**();

void operator=(const Deque&);

friend void operator>>(QTableWidget\*, Deque&);

friend void operator<<(QTableWidget\*, Deque&);

friend void operator>>(QListWidget\*, Deque&);

friend void operator<<(QListWidget\*, Deque&);

friend void operator<<(QTextEdit\*, Deque&);

friend void operator<<(QLabel\* out, Deque &q);

friend void operator+(double i, Deque&);

friend void operator+(Deque&, double i);

friend void operator-(Deque&, Node\*);

friend void operator\*(Deque&, double i);

Node\* **GetHead**();

Node\* **GetTail**();

};

#endif // DEQUE\_H

Mainwindow.cpp:

#include "mainwindow.h"

#include "./ui\_mainwindow.h"

#include "deque.h"

#include <QVariant>

Deque MyDeque;

Deque MyDequeCopy;

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

: QMainWindow(*parent*)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

setWindowIcon(QIcon(":/pictures/Deque/DequeIcon.JPG"));

ui->tableWidget->insertColumn(0);

ui->tableWidget->setColumnWidth(0, 200);

}

void operator>>(QTableWidget\* in, Deque &q)

{

q.FreeList();

int amount = in->rowCount();

for(int i = 0; i < amount; i++)

{

q + in->item(i, 0)->text().toDouble();

}

}

void operator<<(QTableWidget\* out, Deque &q)

{

out->setRowCount(q.GetSize());

out->setColumnCount(1);

int i = 0;

Node \*temp = q.pHead;

while(temp)

{

QString text = QString::number(temp->num);

QTableWidgetItem \*item = new QTableWidgetItem();

item->setText(text);

out->setItem(i, 0, *item*);

temp = temp->pNext;

i++;

}

}

void operator>>(QListWidget\* in, Deque &q)

{

q.FreeList();

int amount = in->count();

for(int i = 0; i < amount; i++)

{

in->setCurrentRow(i);

q + in->currentItem()->text().toDouble();

}

}

void operator<<(QListWidget\* out, Deque &q)

{

out->clear();

Node \*temp = q.pHead;

while(temp)

{

out->addItem(QString::number(temp->num));

temp = temp->pNext;

}

}

void operator<<(QTextEdit\* out, Deque &q)

{

QString text = "";

Node\* temp = q.pHead;

while(temp)

{

text += QString::number(temp->num) + " ";

temp = temp->pNext;

}

out->setText(text);

}

void operator<<(QLabel\* out, Deque &q)

{

out->clear();

QString text = "Last entered queue is ";

Node \*temp = q.pHead;

while(temp)

{

text += QString::number(temp->num) + " ";

temp = temp->pNext;

}

text += ".";

out->setText(text);

}

void operator+(double i, Deque &q)

{

Node\* pCurr = new Node(i);

if(!q.IsEmpty())

{

q.pHead->pPrev = pCurr;

pCurr->pNext = q.pHead;

q.pHead = pCurr;

}

else

{

q.pHead = pCurr;

q.pTail = pCurr;

pCurr->pPrev = nullptr;

pCurr->pNext = nullptr;

}

}

void operator+(Deque &q, double i)

{

Node\* pCurr = new Node(i);

if(!q.IsEmpty())

{

q.pTail->pNext = pCurr;

pCurr->pPrev = q.pTail;

q.pTail = pCurr;

pCurr->pNext = nullptr;

}

else

{

q.pHead = pCurr;

q.pTail = pCurr;

pCurr->pPrev = nullptr;

pCurr->pNext = nullptr;

}

}

void operator-(Deque &q, Node\* side)

{

if(side->pPrev == nullptr && side)

{

if(q.pHead->pNext)

{

Node\* temp = q.pHead->pNext;

delete q.pHead;

q.pHead = temp;

q.pHead->pPrev = nullptr;

}

}

else if(side->pNext == nullptr && side)

{

if(q.pTail->pPrev)

{

Node\* temp = q.pTail->pPrev;

delete q.pTail;

q.pTail = temp;

q.pTail->pNext = nullptr;

}

}

else return;

}

void operator\*(Deque &q, double i)

{

Node\* temp = q.pHead;

while(temp)

{

temp->num = temp->num \* i;

temp = temp->pNext;

}

}

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

{

delete ui;

}

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

{

QListWidgetItem \*item = new QListWidgetItem();

ui->listWidget->addItem(*item*);

item->setFlags (item->flags () | Qt::ItemIsEditable);

}

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

{

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

}

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

{

ui->listWidget >> MyDeque;

MyDequeCopy.FreeList();

MyDequeCopy = MyDeque;

ui->textEdit << MyDeque;

ui->tableWidget << MyDeque;

ui->listWidget << MyDeque;

ui->results\_text->setText("");

}

void MainWindow::**on\_ShowOnTextEdit\_2\_clicked**()

{

ui->tableWidget >> MyDeque;

MyDequeCopy.FreeList();

MyDequeCopy = MyDeque;

ui->textEdit << MyDeque;

ui->tableWidget << MyDeque;

ui->listWidget << MyDeque;

ui->results\_text->setText("");

}

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

{

MyDeque.FreeList();

ui->textEdit->setText("");

ui->results\_text << MyDequeCopy;

MyDequeCopy.FreeList();

ui->listWidget->clear();

ui->tableWidget->clear();

ui->tableWidget->setRowCount(0);

}

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

{

if(MyDeque.GetHead())

ui->results\_text->setText("Max value of queue is " + QString::number( MyDeque.GetMax()) + ".");

}

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

{

if(MyDeque.GetHead())

ui->results\_text->setText("Min value of queue is " + QString::number( MyDeque.GetMin()) + ".");

}

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

{

if(MyDeque.GetHead())

ui->results\_text->setText("Average value of queue is " + QString::number( MyDeque.GetAverage()) + ".");

}

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

{

if(MyDeque.GetHead())

{

if(ui->MultiplyText->toPlainText().toDouble())

{

double i = ui->MultiplyText->toPlainText().toDouble();

MyDeque \* i;

ui->textEdit << MyDeque;

ui->tableWidget << MyDeque;

ui->listWidget << MyDeque;

}

}

}

void MainWindow::**on\_delete\_1st\_clicked**()

{

if(MyDeque.GetHead())

{

MyDeque - MyDeque.GetHead();

ui->textEdit << MyDeque;

ui->tableWidget << MyDeque;

ui->listWidget << MyDeque;

}

}

void MainWindow::**on\_delete\_2nd\_clicked**()

{

if(MyDeque.GetTail())

{

MyDeque - MyDeque.GetTail();

ui->textEdit << MyDeque;

ui->tableWidget << MyDeque;

ui->listWidget << MyDeque;

}

}

void MainWindow::**on\_Add\_1st\_clicked**()

{

ui->AddTopText->toPlainText().toDouble() + MyDeque;

MyDequeCopy.FreeList();

MyDequeCopy = MyDeque;

ui->textEdit << MyDeque;

ui->tableWidget << MyDeque;

ui->listWidget << MyDeque;

}

void MainWindow::**on\_Add\_last\_clicked**()

{

MyDeque + ui->AddBotText->toPlainText().toDouble();

MyDequeCopy.FreeList();

MyDequeCopy = MyDeque;

ui->textEdit << MyDeque;

ui->tableWidget << MyDeque;

ui->listWidget << MyDeque;

}

Deque.cpp:

#include "deque.h"

Node::**Node**()

{

pNext = nullptr;

pPrev = nullptr;

}

Node::**Node**(double i)

{

num = i;

pNext = nullptr;

pPrev = nullptr;

}

Node::**Node**(double i, Node \*pNextNode, Node \*pPrevNode)

{

num = i;

pNext = pNextNode;

pPrev = pPrevNode;

}

Deque::**Deque**()

{

pHead = nullptr;

pTail = nullptr;

}

Deque::**Deque**(const Deque &ToCopy)

{

\*this = ToCopy;

}

Deque::**Deque**(double i)

{

Node\* NewElement = new Node(i);

pHead = NewElement;

pTail = NewElement;

}

Deque::~**Deque**()

{

FreeList();

}

void Deque::operator=(const Deque &ToCopy)

{

Node\* pCurr = nullptr;

Node \*pNextNode = nullptr;

if(!ToCopy.pHead)

{

pHead = nullptr;

return;

}

else

{

pHead = new Node;

pHead->num = ToCopy.pHead->num;

pCurr = pHead;

pNextNode = ToCopy.pHead->pNext;

}

while(pNextNode)

{

pCurr->pNext = new Node;

pCurr = pCurr->pNext;

pCurr->num = pNextNode->num;

pNextNode = pNextNode->pNext;

}

pCurr->pNext = nullptr;

}

int Deque::**GetSize**()

{

int count = 0;

Node\* temp = pHead;

if(temp)

{

while(temp)

{

temp = temp->pNext;

count++;

}

return count;

}

else return 0;

}

double Deque::**GetMin**()

{

double min = pHead->num;

Node\* temp = pHead;

if(temp)

{

while(temp)

{

if(temp->num < min)

{

min = temp->num;

}

temp = temp->pNext;

}

return min;

}

else return 0;

}

double Deque::**GetMax**()

{

double max = pHead->num;

Node\* temp = pHead;

if(temp)

{

while(temp)

{

if(temp->num > max)

{

max = temp->num;

}

temp = temp->pNext;

}

return max;

}

else return 0;

}

double Deque::**GetAverage**()

{

double sum = 0.0;

Node\* temp = pHead;

if(temp)

{

while(temp)

{

sum += temp->num;

temp = temp->pNext;

}

return sum / (double)GetSize();

}

else return 0;

}

void Deque::**FreeList**()

{

if(!IsEmpty())

{

while(pHead != pTail)

{

Node\* temp = pHead->pNext;

delete pHead;

pHead = temp;

pHead->pPrev = nullptr;

}

}

pHead = nullptr;

pTail = nullptr;

}

bool Deque::**IsEmpty**()

{

if(!pHead || !pTail) return true;

else return false;

}

Node\* Deque::**GetHead**()

{

return pHead;

}

Node\* Deque::**GetTail**()

{

return pTail;

}

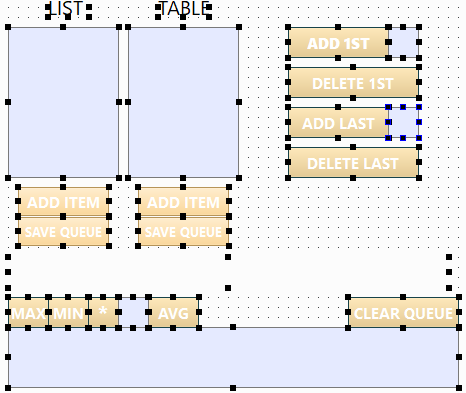


Рис. 1. MainWindow.ui

**Результати виконання програми**

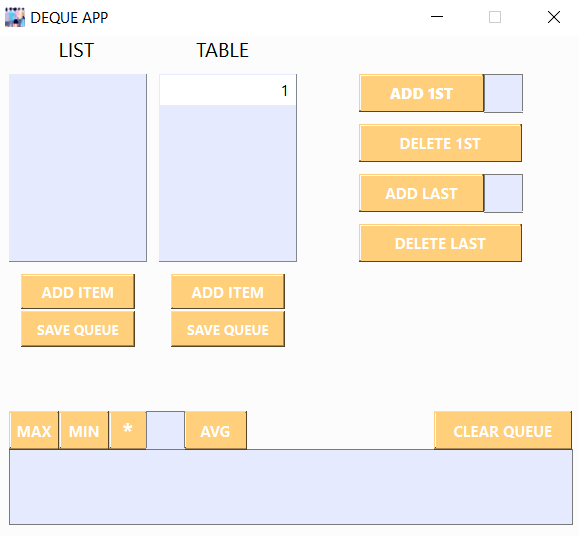


Рис. 2. Вікно програми при запуску

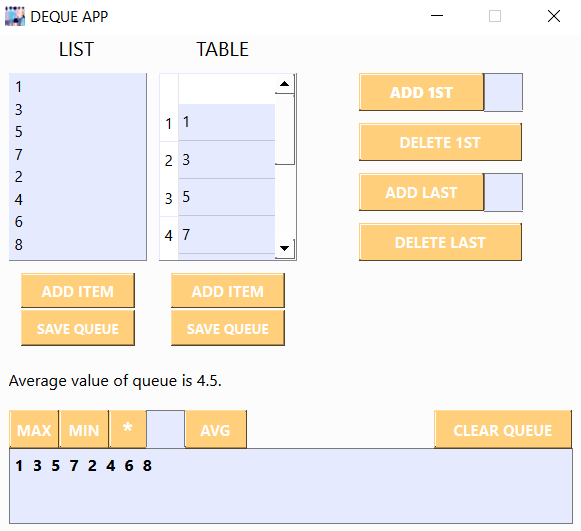


Рис. 3. Результати обчислень програми

**Висновок**

Виконуючи лабораторну роботу №7 я навчився працювати з динамічною пам’ятю в мові C++, створив власний клас Deque та продемонстрував його можливості на віконному застосуванні.