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
Солтисюк Д. А.
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 C++ 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 C++ 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:

customdeque.cpp:

```
#include "customdeque.h"
#include <iostream>

const int linkedListExtremum(Node *front,
                             const bool (*comparator)(double, double)) {
  int result;

  for (Node *temp = front; temp != nullptr; temp = temp->next) {
    if (comparator(temp->data, result)) {
      result = temp->data;
    }
  }

  return result;
}

const bool maxValueComparator(double value1, double value2) {
```

```
  return value1 > value2;
}

const bool minValueComparator(double value1, double value2) {
  return value1 < value2;
}

const double CustomDeque::getMaxValue() {
  return linkedListExtremum(this->front, maxValueComparator);
}
const double CustomDeque::getMinValue() {
  return linkedListExtremum(this->front, minValueComparator);
}

const double CustomDeque::getAverageValue() {
  double result = 0;
  int i = 0;

  for (Node *temp = front; temp != nullptr; temp = temp->next) {
    result += temp->data;
    i++;
  }

  return result / i;
}

void operator+(double data, CustomDeque &dq) { dq.insertFront(data); }
void operator+(CustomDeque &dq, double data) { dq.insertRear(data); }

void operator-(double data, CustomDeque &dq) { dq.deleteFront(); }
void operator-(CustomDeque &dq, Node *side) { dq.deleteRear(); }

void operator>>(QListWidget *in, CustomDeque &dq) {
  dq.erase();

  for (int i = 0; i < in->count(); i++) {
    in->setCurrentRow(i);
    dq + in->currentItem()->text().toDouble();
  }
}

void operator<<(QListWidget *out, CustomDeque &dq) {
  out->clear();

  for (Node *temp = dq.getFront(); temp != nullptr; temp = temp->next) {
    out->addItem(QString::number(temp->data));
  }
}

void operator>>(QLineEdit *in, CustomDeque &dq) { in->text().toDouble() + dq; };

void operator>>(CustomDeque &dq, QLineEdit *in) { dq + in->text().toDouble(); };
```

customdeque.h:

```
#include "deque.h"

#include <QLabel>
#include <QLineEdit>
#include <QListWidget>
#include <QTableWidget>

class CustomDeque : public Deque {
public:
  const double getMaxValue();
  const double getMinValue();
  const double getAverageValue();

  void operator=(const CustomDeque &);

  friend void operator>>(QListWidget *, CustomDeque &);
  friend void operator<<(QListWidget *, CustomDeque &);
  friend void operator>>(QLineEdit *, CustomDeque &);
  friend void operator>>(CustomDeque &, QLineEdit *);
  friend void operator+(double, CustomDeque &);
  friend void operator+(CustomDeque &, double);
  friend void operator-(CustomDeque &, Node *);
  friend void operator-(Node *, CustomDeque &);
};
```

deque.cpp:

```
#include "deque.h"
#include <stdexcept>

// Function to check whether deque
// is empty or not
bool Deque::isEmpty() { return (front == NULL); }

// Function to return the number of
// elements in the deque
double Deque::getSize() { return size; }

// Function to insert an element
// at the front end
void Deque::insertFront(int data) {
  Node *newNode = Node::getNode(data);
  // If true then new element cannot be added
  // and it is an 'Overflow' condition
  if (newNode == NULL)
    throw std::invalid_argument("overflow");
  else {
    // If deque is empty
    if (front == NULL)
      rear = front = newNode;

    // Inserts node at the front end
    else {
      newNode->next = front;
      front->prev = newNode;
      front = newNode;
    }
```

```
    // Increments count of elements by 1
    size++;
  }
}

// Function to insert an element
// at the rear end
void Deque::insertRear(int data) {
  Node *newNode = Node::getNode(data);
  // If true then new element cannot be added
  // and it is an 'Overflow' condition
  if (newNode == NULL)
    throw std::invalid_argument("overflow");
  else {
    // If deque is empty
    if (rear == NULL)
      front = rear = newNode;

    // Inserts node at the rear end
    else {
      newNode->prev = rear;
      rear->next = newNode;
      rear = newNode;
    }

    size++;
  }
}

// Function to delete the element
// from the front end
void Deque::deleteFront() {
  // If deque is empty then
  // 'Underflow' condition
  if (isEmpty())
    throw std::invalid_argument("underflow");

  // Deletes the node from the front end and makes
  // the adjustment in the links
  else {
    Node *temp = front;
    front = front->next;

    // If only one element was present
    if (front == NULL)
      rear = NULL;
    else
      front->prev = NULL;
    free(temp);

    // Decrements count of elements by 1
    size--;
  }
}

// Function to delete the element
// from the rear end
void Deque::deleteRear() {
  // If deque is empty then
  // 'Underflow' condition
  if (isEmpty())
```

```
    throw std::invalid_argument("underflow");

  // Deletes the node from the rear end and makes
  // the adjustment in the links
  else {
    Node *temp = rear;
    rear = rear->prev;

    // If only one element was present
    if (rear == NULL)
      front = NULL;
    else
      rear->next = NULL;
    free(temp);

    // Decrements count of elements by 1
    size--;
  }
}

Node *Deque::getFront() { return this->front; }

Node *Deque::getRear() { return this->rear; }

// Function to delete all the elements
// from Deque
void Deque::erase() {
  rear = NULL;
  while (front != NULL) {
    Node *temp = front;
    front = front->next;
    free(temp);
  }
  size = 0;
}
```

deque.h:

```
#include "node.h"

// A structure to represent a deque
class Deque {
protected:
  Node *front;
  Node *rear;
  int size;

public:
  Deque() {
    front = rear = NULL;
    size = 0;
  }

  // Copy contructor
  Deque(const Deque &dq) {
    this->front = dq.front;
    this->rear = dq.rear;
  }

  ~Deque() { this->erase(); }
```

```
  // Operations on Deque
  void insertFront(int data);
  void insertRear(int data);
  void deleteFront();
  void deleteRear();
  Node *getFront();
  Node *getRear();
  double getSize();
  bool isEmpty();
  void erase();
};
```

main.cpp:

```
#include "widget.h"

#include <QApplication>

int main(int argc, char *argv[]) {
  QApplication a(argc, argv);
  Widget w;

  w.show();

  return a.exec();
}
```

node.h:

```
#include <stdlib.h> /* malloc, free, rand */

// Node of a doubly linked list
struct Node {
  double data;
  Node *prev, *next;
  // Function to get a new node
  static Node *getNode(int data) {
    Node *newNode = (Node *)malloc(sizeof(Node));
    newNode->data = data;
    newNode->prev = newNode->next = NULL;
    return newNode;
  }
};
```

widget.cpp:

```
#include "widget.h"

#include <QGridLayout>

void Widget::onQueueClear() {
  this->dq->erase();

  emit valueChanged(this->dq);
}

void Widget::onFrontItemInsert() {
  this->newQueueItemInput >> *this->dq;

  emit valueChanged(this->dq);
};
```

```
void Widget::onRearItemInsert() {
  *this->dq >> this->newQueueItemInput;

  emit valueChanged(this->dq);
};

void Widget::onFrontItemDelete() {
  this->dq->deleteFront();

  emit valueChanged(this->dq);
}

void Widget::onRearItemDelete() {
  this->dq->deleteRear();

  emit valueChanged(this->dq);
}

void Widget::onValueChange(CustomDeque *dq) {
  this->newQueueItemInput->clear();

  this->biggestValueInput->setText(QString::number(dq->getMaxValue()));
  this->smallestValueInput->setText(QString::number(dq->getMinValue()));
  this->averageValueInput->setText(QString::number(dq->getAverageValue()));

  // if queue is empty
  if (dq->getSize() == 0) {
    this->deleteFrontItemButton->setDisabled(true);
    this->deleteRearItemButton->setDisabled(true);

    this->biggestValueInput->clear();
    this->smallestValueInput->clear();
    this->averageValueInput->clear();
  }

  this->listWidget << *dq;
}

Widget::Widget(QWidget *parent) : QWidget(parent) {
  QGridLayout *mainLayout = new QGridLayout;
  QGridLayout *queueControlButtonsLayout = new QGridLayout;

  QSizePolicy sizePolicy(QSizePolicy::Minimum, QSizePolicy::Minimum);
  sizePolicy.setHorizontalStretch(0);
  sizePolicy.setVerticalStretch(0);

  this->dq = new CustomDeque;
  this->listWidget = new QListWidget;
  this->listWidget->setItemAlignment(Qt::AlignCenter);

  this->newQueueItemInput = new QLineEdit;
  this->newQueueItemInput->setAlignment(Qt::AlignCenter);

  this->insertFrontItemButton = new QPushButton("Insert");
  this->insertRearItemButton = new QPushButton("Insert");
  this->deleteFrontItemButton = new QPushButton("Delete");
  this->deleteRearItemButton = new QPushButton("Delete");

  this->clearQueueButton = new QPushButton("Clear queue");
  this->clearQueueButton->setSizePolicy(sizePolicy);
```

```
    this->smallestValueInput = new QLineEdit;
    this->biggestValueInput = new QLineEdit;
    this->averageValueInput = new QLineEdit;

    QLabel *frontLabel = new QLabel(QString("Front").toUpper());
    frontLabel->setAlignment(Qt::AlignLeft);

    QLabel *rearLabel = new QLabel(QString("Rear").toUpper());
    rearLabel->setAlignment(Qt::AlignRight);

    smallestValueInput->setDisabled(true);
    biggestValueInput->setDisabled(true);
    averageValueInput->setDisabled(true);

    queueControlButtonsLayout->addWidget(frontLabel, 0, 0);
    queueControlButtonsLayout->addWidget(this->insertFrontItemButton, 1, 0);
    queueControlButtonsLayout->addWidget(this->deleteFrontItemButton, 2, 0);

    queueControlButtonsLayout->addWidget(this->clearQueueButton, 1, 1, 2, 1);

    queueControlButtonsLayout->addWidget(rearLabel, 0, 2);
    queueControlButtonsLayout->addWidget(this->insertRearItemButton, 1, 2);
    queueControlButtonsLayout->addWidget(this->deleteRearItemButton, 2, 2);

    mainLayout->addWidget(this->listWidget, 0, 0, 1, 2);
    mainLayout->addWidget(this->newQueueItemInput, 1, 0, 1, 2);
    mainLayout->addLayout(queueControlButtonsLayout, 2, 0, 1, 2);

    mainLayout->addWidget(new QLabel("Biggest value:"), 3, 0);
    mainLayout->addWidget(this->biggestValueInput, 3, 1);

    mainLayout->addWidget(new QLabel("Minimum value:"), 4, 0);
    mainLayout->addWidget(this->smallestValueInput, 4, 1);

    mainLayout->addWidget(new QLabel("Average value:"), 5, 0);
    mainLayout->addWidget(this->averageValueInput, 5, 1);

    connect(this->insertFrontItemButton, &QPushButton::released, this,
            &Widget::onFrontItemInsert);

    connect(this->insertRearItemButton, &QPushButton::released, this,
            &Widget::onRearItemInsert);

    connect(this->deleteFrontItemButton, &QPushButton::released, this,
            &Widget::onFrontItemDelete);

    connect(this->deleteRearItemButton, &QPushButton::released, this,
            &Widget::onRearItemDelete);

    connect(this->clearQueueButton, &QPushButton::released, this,
            &Widget::onQueueClear);

    connect(this, &Widget::valueChanged, &Widget::onValueChange);

    setLayout(mainLayout);
}
```

widget.h:

```
#pragma once

#include "customdeque.h"

#include <QLabel>
#include <QLineEdit>
#include <QPushButton>
#include <QTextEdit>
#include <QWidget>

class Widget : public QWidget {
  Q_OBJECT

public:
  Widget(QWidget *parent = nullptr);

private slots:
  void onFrontItemInsert();
  void onRearItemInsert();
  void onFrontItemDelete();
  void onRearItemDelete();
  void onQueueClear();
  void onValueChange(CustomDeque *dq);

signals:
  void valueChanged(CustomDeque *dq);

private:
  CustomDeque *dq;

  QListWidget *listWidget;
  QLineEdit *newQueueItemInput;

  QPushButton *insertFrontItemButton;
  QPushButton *insertRearItemButton;
  QPushButton *deleteFrontItemButton;
  QPushButton *deleteRearItemButton;
  QPushButton *clearQueueButton;

  QLineEdit *biggestValueInput;
  QLineEdit *smallestValueInput;
  QLineEdit *averageValueInput;
};
```

Результати виконання програми

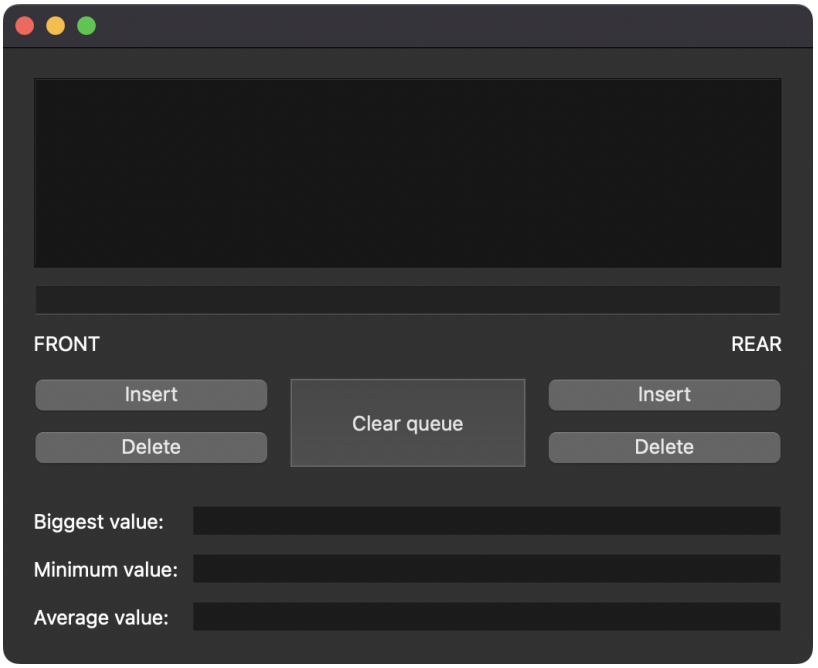

Рис. 1. Вікно програми при запуску

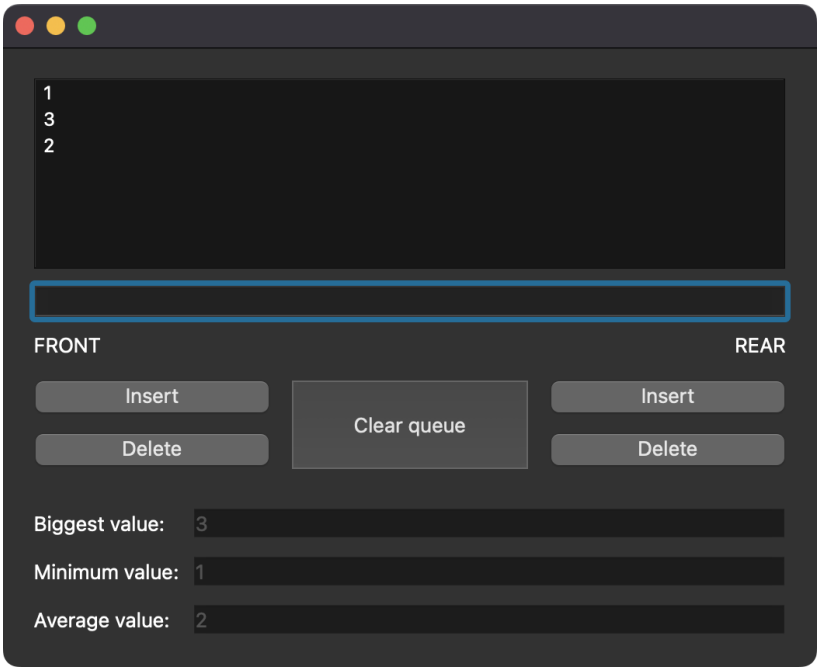

Рис. 2. Результати обчислень програми

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.