МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Інститут комп'ютерних наук та інформаційних технологій
Кафедра програмного забезпечення

ЗВІТ

До лабораторної роботи № 7
На тему: «Робота з динамічною пам'яттю»
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.

∑= _____ .....:

Львів-2022

Тема роботи: Робота з динамічною пам’яттю

Мета роботи: 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

Види 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. 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.

Робота з динамічною пам’яттю в об’єктах

Якщо в об’єкті виділяється динамічна пам’ять, на яку вказує його поле-вказівник (об’єкт володіє виділеною пам’яттю), то ця пам’ять обов’язково має бути звільнена об’єктом або передана у володіння в інше місце програми. При цьому потрібно пам’ятати, що стандартні конструктор копіювання та оператор присвоєння не виконують “глибокого” копіювання (не копіюють виділену пам’ять, а копіюють тільки вказівники на неї). Тому при їх виконанні можливі ситуації, коли різні об’єкти міститимуть вказівники на одну і ту ж ділянку пам’яті. При цьому, якщо об’єкти не знатимуть про таку ситуацію, можливе повторне звільнення уже звільненої пам’яті, що приведе до фатальної помилки виконання програми. Тому, якщо екземпляри класів виділяють динамічну пам’ять і

зберігають вказівники на неї у своїх полях, необхідно для них перевизначати конструктор копіювання та оператор присвоєння, а для звільнення пам’яті при потребі – деструктор.

Об’єкти в статичній, локальній та динамічній пам’яті

Об’єкти, як і змінні будь-якого стандартного типу, можна виділяти в усіх трьох видах пам’яті.

Завдання для лабораторної роботи

1. Переглянути лістинг коду в прикладі. Пояснити вивід програми.
2. Створити клас відповідно до завдання (див. Додаток).
3. Розробити для класу конструктор за замовчуванням та декілька звичайних конструкторів. Реалізувати функції-члени відповідно до завдання (див. Додаток).
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.

5. Клас List – однозв'язний список. Пам’ять під елементи списку повинна виділятися динамічно. Реалізувати такі функції члени:

- Отримання першого елемента
- Отримання останнього елемента
- Отримання кількості елементів у списку
- Знаходження максимального значення.
- Знаходження мінімального значення.
- Знаходження середнього арифметичного значення списку.
- Сортування елементів списку методом вибірки за спаданням.
- Сортування елементів списку методом бульбашки за зростанням.

Перевантажити операції. При цьому вибір механізму перевантаження обрати самостійно (чи метод, чи дружня-функція):

- Додавання (почленне додавання елементів до списку)
- Віднімання (почленне видалення елементів списку)
- Множення на скаляр.
- Введення списку з StringGrid (>>)
- Виведення списку у StringGrid (<<)
- Введення списку з ListBox (>>)
- Виведення списку у ListBox (<<)
- Виведення спику у Memo (<<)

Хід роботи

Файл list.h:

```
#ifndef LIST_H
#define LIST_H
#include <QString>
#include <QStringList>
#include <QLineEdit>
#include <QTextEdit>
#include <QTableWidget>
#include <QListWidget>

struct node
{
    int value;
```

```
        node * next;
    };
    struct returnValue
    {
        int value;
        bool error_indicator;
    };

    class List
    {
    private:
        node *head, *tail;
    public:
        List();
        List(List & list);
        ~List();
        void operator>>(QTextEdit * text);
        void operator>>(QTableWidget * table);
        void operator>>(QListWidget * list);
        void operator<<(QTextEdit * text);
        void operator<<(QTableWidget * table);
        void operator<<(QListWidget * list);
        void operator=(List & list);
        void operator+(int a);
        void operator-(int index);
        void operator*(int mult);
        void clear();
        int getElements();
        int getLastId();
        returnValue getFirst();
        returnValue getLast();
        int getMax();
        int getMin();
        double getAvg();
        void selectionSort();
        void bubbleSort();
    };

#endif // LIST_H
```

Файл list.cpp:

```
    #include "list.h"

    List::List()
    {
        head = nullptr;
        tail = nullptr;
    }
    List::List(List & list)
    {
        head = nullptr;
        tail = nullptr;
        if(list.head)
        {
            node * currI = list.head;
            operator+(currI->value);
            currI = currI->next;
        }
    }
    List::~List()
    {
        clear();
    }
    void List::operator<<(QTextEdit * text)
    {
        clear();
        QStringList list = text->toPlainText().split(" ");
        for (QString i : list)
```

```
    {
        operator+(i.toInt());
    }
}
void List::operator<<(QTableWidget * table)
{
    clear();
    for(int i = 0; i < table->rowCount(); i++)
    {
        for(int j = 0; j < table->columnCount(); j++)
        {
            if(table->item(i, j))
            {
                if(!table->item(i, j)->text().isEmpty())
                {
                    operator+(table->item(i, j)->text().toInt());
                }
            }
        }
    }
}
void List::operator<<(QListWidget * list)
{
    clear();
    for(int i = 0; i < list->count(); i++)
    {
        if(list->item(i))
        {
            if(!list->item(i)->text().isEmpty())
            {
                operator+(list->item(i)->text().toInt());
            }
        }
    }
}
void List::operator>>(QTextEdit * text)
{
    if(text->toPlainText() != "")
        text->clear();
    if(head)
    {
        node * currI = head;
        while(currI)
        {
            text->insertPlainText(QString::number(currI->value));
            currI = currI->next;
            if(currI)
            {
                text->insertPlainText(" ");
            }
        }
    }
    else
    {
        text->blockSignals(true);
        text->setText("Empty list");
        text->blockSignals(false);
    }
}
void List::operator>>(QTableWidget * table)
{
    int i = 0;
    node * currI = head;
    for(int i = 0; i < table->rowCount(); i++)
    {
        for(int j = 0; j < table->columnCount(); j++)
        {
            QTableWidgetItem * item = new QTableWidgetItem;
```

```
            item->setText(NULL);
            table->setItem(i, j, item);
        }
    }
    while(currI)
    {
        QTableWidgetItem * item = new QTableWidgetItem;
        item->setText(QString::number(currI->value));
        table->setItem((i / table->columnCount()), (i - (i / table->columnCount() *
table->columnCount())), item);
        i++;
        currI = currI->next;
    }
}
void List::operator>>(QListWidget * list)
{
    list->clear();
    if(head)
    {
        int i = 0;
        node * currI = head;
        while(currI)
        {
            QListWidgetItem * item = new QListWidgetItem(QString::number(currI-
>value));
            item->setFlags(item->flags() | Qt::ItemIsEditable);
            list->addItem(item);
            i++;
            currI = currI->next;
        }
    }
    else
    {
        QListWidgetItem * item = new QListWidgetItem("Empty list");
        item->setFlags(item->flags() | Qt::ItemIsEditable);
 list->addItem(item);
}
}
void List::operator=(List & list)
{
    clear();
    if(list.head)
    {
        node * currI = list.head;
        operator+(currI->value);
        currI = currI->next;
    }
}
void List::operator+(int a)
{
    node * newNode = new node();
    newNode->value = a;
    if(!head)
    {
        head = newNode;
        tail = newNode;
        newNode->next = nullptr;
    }
    else
    {
        tail->next = newNode;
        tail = newNode;
    }
}
void List::operator-(int index)
{
    node * currI = head;
    node * prevI = head;
```

```
    int i = 0;
    while(i != index)
    {
        prevI = currI;
        currI = currI->next;
        i++;
    }
    }
    if(index == 0)
    {
        head = currI->next;
        delete currI;
    }
    else
    {
        prevI->next = currI->next;
        if(currI == tail)
        {
            tail = prevI;
        }
        delete currI;
    }
}
void List::operator*(int mult)
{
    if(head)
    {
        node * currI = head;
        while(currI)
        {
            currI->value *= mult;
            currI = currI->next;
        }
    }
}
void List::clear()
{
   while(getLastId() >= 0)
   {
       this->operator-(getLastId());
   }
}
int List::getElements()
{
    int elements = 0;
    if(head)
    {
        node * currI = head;
        while(currI)
        {
            currI = currI->next;
            elements++;
        }
    }
    return elements;
}
int List::getLastId()
{
    int i = -1;
    if(head)
    {
        i = 0;
        node * currI = head;
        while(currI->next)
        {
            currI = currI->next;
            i++;
        }
    }
```

```
    return i;
}
returnValue List::getFirst()
{
    if(head)
    {
        return returnValue{head->value, false};
    }
    else return returnValue{0, true};
}
returnValue List::getLast()
{
    if(tail)
    {
        return returnValue{tail->value, false};
    }
    else return returnValue{0, true};
}
int List::getMax()
{
    int max = 0;
    if(head)
    {
        node * currI = head;
        while(currI)
        {
            if(currI->value > max)
                max = currI->value;
            currI = currI->next;
        }
    }
    return max;
}
int List::getMin()
{
    int min = 0;
    if(head)
    {
        min = head->value;
        node * currI = head;
        while(currI)
        {
            if(currI->value < min)
                min = currI->value;
            currI = currI->next;
        }
    }
    return min;
}
double List::getAvg()
{
    int sum = 0;
   if(head)
   {
       node * currI = head;
       while(currI)
       {
           sum += currI->value;
           currI = currI->next;
       }
       return (double)sum / getElements();
   }
   else return sum;
}
void List::selectionSort()
{
    if(head)
    {
```

```
            node * start = head;
            while(start->next)
            {
                node * max = start;
                node * currI = start;
                while(currI)
                {
                    if(currI->value > max->value)
                        max = currI;
                    currI = currI->next;
                }
                int temp = start->value;
                start->value = max->value;
                max->value = temp;
                start = start->next;
            }
        }
    }
    void List::bubbleSort()
    {
        if(head)
        {
            int i = 0;
            while(i < getElements() - 1)
            {
                node * currI = head;
                int j = 0;
                while(j < getElements() - 1 - i)
                {
                    if(currI->value > currI->next->value)
                    {
                        int temp = currI->value;
                        currI->value = currI->next->value;
                        currI->next->value = temp;
                    }
                    currI = currI->next;
                    j++;
                }
                i++;
            }
        }
}
```

Файл mainwindow.cpp:

```
    …
    MainWindow::MainWindow(QWidget *parent)
        : QMainWindow(parent)
        , ui(new Ui::MainWindow)
    {
        ui->setupUi(this);
        QIntValidator * validator = new QIntValidator(-10000, 10000);
        ui->lineEdit->setValidator(validator);
        ui->lineEdit_2->setValidator(validator);
        QListWidgetItem *item = new QListWidgetItem("");
        item->setFlags( item->flags() | Qt::ItemIsEditable );
        ui->listWidget->addItem(item);

    }

    MainWindow::~MainWindow()
    {
        delete ui;
    }
    List * list = NULL;
    void MainWindow::on_textEdit_textChanged()
    {
        QString f = ui->textEdit->toPlainText();
        f.remove(QRegularExpression("[A-Za-z!@#$%^&*\t\n]+"));
```

```
    ui->textEdit->blockSignals(true);
    ui->textEdit->setText(f);
    QTextCursor cursor = ui->textEdit->textCursor();
    cursor.setPosition(ui->textEdit->document()->characterCount() - 1);
    ui->textEdit->setTextCursor(cursor);
    ui->textEdit->blockSignals(false);
}


void MainWindow::on_pushButton_clicked()
{
    QMessageBox * message = new QMessageBox();
    message->setText("From which widget?");
    message->addButton("Table", QMessageBox::AcceptRole);
    message->addButton("Text editor", QMessageBox::RejectRole);
    message->addButton("List", QMessageBox::DestructiveRole);
    int button = message->exec();
    delete message;
    if(list)
    {
        delete list;
    }
    list = new List();
    ui->label->setEnabled(true);
    ui->label_2->setEnabled(true);
    ui->lineEdit->setEnabled(true);
    ui->lineEdit_2->setEnabled(true);
    ui->lineEdit_3->setEnabled(true);
    ui->pushButton_2->setEnabled(true);
    ui->pushButton_3->setEnabled(true);
    ui->pushButton_4->setEnabled(true);
    ui->pushButton_5->setEnabled(true);
    ui->pushButton_8->setEnabled(true);
    ui->pushButton_9->setEnabled(true);
    ui->pushButton_10->setEnabled(true);
    ui->pushButton_11->setEnabled(true);
    ui->pushButton_12->setEnabled(true);
    switch(button)
    {
    case 0:
        *list<<ui->tableWidget;
        break;
    case 1:
        *list<<ui->textEdit;
        break;
    case 2:
        *list<<ui->listWidget;
    default:
        break;
    }
    ui->label_3->setText("Total elements: " + QString::number(list->getElements()));
    ui->label_4->setText("Max value: " + QString::number(list->getMax()));
    ui->label_5->setText("Min value: " + QString::number(list->getMin()));
    ui->label_6->setText("Avg value: " + QString::number(list->getAvg()));
}


void MainWindow::on_pushButton_3_clicked()
{
    if(!(ui->lineEdit->text() == ""))
    {
        *list + ui->lineEdit->text().toInt();
        ui->lineEdit->clear();
        ui->label_3->setText("Total elements: " + QString::number(list-
>getElements()));
        ui->label_4->setText("Max value: " + QString::number(list->getMax()));
        ui->label_5->setText("Min value: " + QString::number(list->getMin()));
        ui->label_6->setText("Avg value: " + QString::number(list->getAvg()));
```

```
    }
}

void MainWindow::on_pushButton_12_clicked()
{
    if(!(ui->lineEdit->text() == ""))
    {
        *list * ui->lineEdit->text().toInt();
        ui->lineEdit->clear();
        ui->label_4->setText("Max value: " + QString::number(list->getMax()));
        ui->label_5->setText("Min value: " + QString::number(list->getMin()));
        ui->label_6->setText("Avg value: " + QString::number(list->getAvg()));
    }
}

void MainWindow::on_pushButton_2_clicked()
{
    QMessageBox * message = new QMessageBox();
    message->setText("To which widget?");
    message->addButton("Table", QMessageBox::AcceptRole);
    message->addButton("Text editor", QMessageBox::RejectRole);
    message->addButton("List", QMessageBox::DestructiveRole);
    int button = message->exec();
    delete message;
    switch(button)
    {
    case 0:
        *list>>ui->tableWidget;
        break;
    case 1:
        *list>>ui->textEdit;
        break;
    case 2:
        *list>>ui->listWidget;
    default:
        break;
    }
}
void MainWindow::on_pushButton_5_clicked()
{
    int id = list->getLastId();
    if(id >= 0)
    {
        ui->lineEdit_2->setText(QString::number(id));
    }
}


void MainWindow::on_pushButton_4_clicked()
{
    if(ui->lineEdit_2->text() != "")
    {
        int id = list->getLastId();
        if(id >= 0)
        {
            if(ui->lineEdit_2->text().toInt() > id)
            {
                ui->lineEdit_2->setText(QString::number(id));
            }
            *list - (ui->lineEdit_2->text()).toInt();
            ui->lineEdit_2->clear();
        }
        else
            ui->lineEdit_2->clear();
        ui->label_3->setText("Total elements: " + QString::number(list-
>getElements()));
        ui->label_4->setText("Max value: " + QString::number(list->getMax()));
        ui->label_5->setText("Min value: " + QString::number(list->getMin()));
```

```
        ui->label_6->setText("Avg value: " + QString::number(list->getAvg()));
    }
}



void MainWindow::on_tableWidget_cellChanged(int row, int column)
{
    QTableWidgetItem * item = new QTableWidgetItem;
    item->setText(ui->tableWidget->item(row, column)-
>text().remove(QRegularExpression("[A-Za-z!@#$%^&*\t\n]+")));
    ui->tableWidget->blockSignals(true);
    ui->tableWidget->setItem(row, column, item);
    ui->tableWidget->blockSignals(false);
}


void MainWindow::on_pushButton_6_clicked()
{
    QListWidgetItem *item = new QListWidgetItem("");
    item->setFlags( item->flags() | Qt::ItemIsEditable );
    ui->listWidget->addItem(item);
}


void MainWindow::on_pushButton_7_clicked()
{
    delete ui->listWidget->takeItem(ui->listWidget->count() - 1);
}

void MainWindow::on_listWidget_itemChanged(QListWidgetItem *item)
{
    QString f = item->text();
    f.remove(QRegularExpression("[A-Za-z!@#$%^&*\t\n]+"));
    ui->listWidget->blockSignals(true);
    ui->listWidget->currentItem()->setText(f);
    ui->listWidget->blockSignals(false);
}


void MainWindow::on_pushButton_8_clicked()
{
    if(list->getFirst().error_indicator == false)
    {
        ui->lineEdit_3->setText(QString::number(list->getFirst().value));
    }
    else ui->lineEdit_3->setText("No number");
}


void MainWindow::on_pushButton_9_clicked()
{
    if(list->getLast().error_indicator == false)
    {
        ui->lineEdit_3->setText(QString::number(list->getLast().value));
    }
    else ui->lineEdit_3->setText("No number");
}


void MainWindow::on_pushButton_10_clicked()
{
    list->selectionSort();
}


void MainWindow::on_pushButton_11_clicked()
{
```

```
            list->bubbleSort();
}
```

Висновок: навчився виділяти місце під об’єкти динамічно. Навчився створювати та використовувати конструктор копіювання, перевантажувати оператор присвоєння. Ознайомився з принципами створення та функціонування деструкторів.