Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа
по курсу «Объектно-ориентированное программирование»
III Семестр

Задание 6
Вариант 27
Основы работы с коллекциями: итераторы

Студент:	Чурсина Н. А.
Группа:	М8О-208Б-18
Преподаватель:	Поповкин А. В.
Оценка:	
Дата:	

Москва 2019

1. Код программы на языке C++

Lab6.cpp

```
#include "TRectangle.h"
#include "TBinTree.h"


int main(int argc, char **argv) {
    TBinTree<Figure> binTree;

    int size;
    int menuOpt = 5;
//      int index =0 ;

    std::shared_ptr<Figure> figure;

    while (menuOpt != 0) {
        if (menuOpt >= 0 && menuOpt <= 5) {
            switch (menuOpt) {
                case 0:
                    break;
                case 1:
                    std::cout << "Input coordinates beginning with the left point:\n ";
                    figure = std::make_shared<TRectangle>(std::cin);
                    binTree.Insert(figure);
                    std::cout << "Square Added" << std::endl;
                    break;
                case 2:
                    std::cout << "Enter length of side to delete Square: "; //!
                    std::cin >> size;
                    binTree.DeleteItem(size);
                    break;
                case 3:
                    binTree.Print();
                    break;
                case 4:
                    if (!binTree.Empty()) {
                        for (auto i : binTree) {
                            i->Print();
                        }
                    } else {
                        std::cout << "BinTree is Empty" << std::endl;
                    }
                    break;
                case 5:
                    std::cout << "*****************MENU****************" << std::endl;
                    std::cout << "1. Add a new Square to the BinTree" << std::endl;
                    std::cout << "2. Delete a Square from the BinTree" << std::endl;
                    std::cout << "3. Print the BinTree" << std::endl;
                    std::cout << "4. Print the BinTree with Iterator" << std::endl;
                    std::cout << "5. Print the MENU" << std::endl;
                    std::cout << "0. Exit " << std::endl;
                    std::cout << "*************************************" << std::endl;
                    break;
            }
        } else {
            std::cout << "Input ERROR" << std::endl;
        }
        std::cout << "Enter the Number of an Action: ";
        std::cin >> menuOpt;
    }
    return 0;
}
```

TAllocationBlock.h

```
#ifndef TALLOCATIONBLOCK_H
#define TALLOCATIONBLOCK_H

#include <iostream>
#include <cstdlib>
#include "TList.h"

class TAllocationBlock {
public:
    TAllocationBlock(int32_t size, int32_t count);

    void *Allocate();

    void Deallocate(void *pointer);

    bool Empty();
```

```
    int32_t Size();

    virtual ~TAllocationBlock();

private:
    char *_used_blocks;
    TList _free_blocks; //список для свободных блоков
};

#endif /* TALLOCATIONBLOCK_H */
#ifndef TALLOCATIONBLOCK_H
#define TALLOCATIONBLOCK_H

#include <iostream>
#include <cstdlib>
#include "TList.h"

class TAllocationBlock {
public:
    TAllocationBlock(int32_t size, int32_t count);

    void *Allocate();

    void Deallocate(void *pointer);

    bool Empty();

    int32_t Size();

    virtual ~TAllocationBlock();

private:
    char *_used_blocks;
    TList _free_blocks; //список для свободных блоков
};

#endif /* TALLOCATIONBLOCK_H */
```

TAllocationBlock.cpp

```
#include "TAllocationBlock.h"

TAllocationBlock::TAllocationBlock(int32_t size, int32_t count) { //конструктор класса
    _used_blocks = (char *) malloc(size * count);

    for (int32_t i = 0; i < count; ++i) {
        void *ptr;
        ptr = _used_blocks + i * size;
        _free_blocks.AddLast(ptr);
    }
    std::cout << "Allocator: Constructor" << std::endl;
}

void *TAllocationBlock::Allocate() {
    if (!_free_blocks.Empty()) {
        void *res = _free_blocks.GetBlock();
        int first = 1;
        _free_blocks.DelElement(first);
        std::cout << "Allocator: Allocate" << std::endl;
        return res;
    }

}

void TAllocationBlock::Deallocate(void *ptr) {
    _free_blocks.AddFirst(ptr);
    std::cout << "Allocator: Deallocate" << std::endl;
}

bool TAllocationBlock::Empty() {
    return _free_blocks.Empty();
}

int32_t TAllocationBlock::Size() {  //получение количества выделенных блоков
    return _free_blocks.Length();
}

TAllocationBlock::~TAllocationBlock() {
    while (!_free_blocks.Empty()) {
        int first = 1;
        _free_blocks.DelElement(first);
    }
```

```
    free(_used_blocks);
    //delete _free_blocks;
    //delete _used_blocks;
}
```

Vector.cpp

```
#include <iostream>
#include <cmath>
#include <string.h>
#define DEFAULT_MEMORY_LEAK 8

template<class T>
class Vector {
public :

    explicit Vector() : lengthOfMemory(DEFAULT_MEMORY_LEAK),
                        lengthOfArray(0) {
        vectorArray = new T[lengthOfMemory];

        for (size_t i = 0; i < lengthOfMemory; ++i) {

            vectorArray[i] = 0;
        }

    };


    Vector(const Vector &copyVec) : lengthOfArray(copyVec.lengthOfArray),
                                    lengthOfMemory(copyVec.lengthOfMemory) {
        vectorArray = new T[lengthOfMemory];

        for (size_t i = 0; i < lengthOfArray; ++i) {

            vectorArray[i] = copyVec.vectorArray[i];
        }


    };

    ~Vector() {

        lengthOfArray = 0;

        lengthOfMemory = 0;

        delete vectorArray;

    };


    void addElement(const T &elementToAdd) {

        if (lengthOfArray >= lengthOfMemory - 1) {

            resizeVector();

        }

        vectorArray[lengthOfArray] = elementToAdd;

        ++lengthOfArray;

    };

    T &operator[]
            (const int i) {
        return vectorArray[i];
    };


    bool isPerpendicular() {

        T v11 = 0, v12 = 0, v21 = 0, v22 = 0, v31 = 0, v32 = 0;
```

```
        v11 = vectorArray[2] - vectorArray[0]; // x1
        v12 = vectorArray[3] - vectorArray[1]; //  y1

        // направляющая второго вектора
        v21 = vectorArray[4] - vectorArray[2]; //  x2
        v22 = vectorArray[5] - vectorArray[3]; //  y2

        // направляющая  третьего вектора
        v31 = vectorArray[6] - vectorArray[4]; // x3
        v32 = vectorArray[7] - vectorArray[5]; //  y3

        double cos1 = v11 * v21 + v12 * v22;
        double cos2 = v21 * v31 + v22 * v32;

        return (cos1 == 0) && (cos2 == 0);

    };

    T GetSide() {
        T sideSize = 0;

        sideSize = sqrt((vectorArray[2] - vectorArray[0]) * (vectorArray[2] - vectorArray[0]) +
                        (vectorArray[3] - vectorArray[1]) * (vectorArray[3] - vectorArray[1]));

        return sideSize;
    };

    T Square() {

        T square = 0;

        for (int i = 0; i < lengthOfArray - 4; i += 2) {
            square += vectorArray[i] * vectorArray[i + 3];
        }
        square += vectorArray[lengthOfArray - 2] * vectorArray[1];

        for (int i = 1; i < lengthOfArray - 3; i += 2) {
            square -= vectorArray[i + 1] * vectorArray[i];
        }
        square -= vectorArray[lengthOfArray - 1] * vectorArray[0];

        return abs(square);

    };


    friend std::ostream &operator<<(std::ostream &os, const Vector<T> &obj) {

        os << "[ ";
        for (int i = 0; i < obj.lengthOfArray; ++i) {
            os << obj.vectorArray[i] << " ";
        }
        os << "]";
        return os;
    };

    friend std::istream &operator>>(std::istream &is, Vector<T> &obj) {
        T element = 0;

        is >> element;
        obj.addElement(element);
        return is;
    };


private :

    void resizeVector() {
        T *newVectorArray = new T[lengthOfMemory * 2];
        std::copy(vectorArray, vectorArray + lengthOfArray, newVectorArray);
        delete[]vectorArray;
        vectorArray = newVectorArray;
        lengthOfMemory *= 2;
    };
```

```
    T *vectorArray;
    size_t lengthOfArray;
    size_t lengthOfMemory;

};
```

2. Ссылка на репозиторий на Github

https://github.com/kwk18/OOP/tree/master/oop_exercise_06

3.Набор testcases

```
Allocator: Constructor
BinTree: Created
***************MENU***************
1. Add a new Square to the BinTree
2. Delete a Square from the BinTree
3. Print the BinTree
4. Print the BinTree with Iterator
5. Print the MENU
0. Exit
************************************
Enter the Number of an Action: 1
Input coordinates beginning with the left point:
 0 0 0 1 1 1 1 0
Allocator: Allocate
BinTree Item: Created
Square Added
Enter the Number of an Action: 1
Input coordinates beginning with the left point:
 0 0 0 5 5 5 5 0
Allocator: Allocate
BinTree Item: Created
Square Added
Enter the Number of an Action: 1
Input coordinates beginning with the left point:
 0 0 0 9 9 9 9 0
Allocator: Allocate
BinTree Item: Created
Square Added
Enter the Number of an Action: 3
          \__9(81)
       \__5(25)
   \__1(1)
Enter the Number of an Action: 1
Input coordinates beginning with the left point:
 0 0 0 3 3 3 3 0
Allocator: Allocate
BinTree Item: Created
Square Added
Enter the Number of an Action: 3
          \__9(81)
```

```
        \__5(25)
           \__3(9)
     \__1(1)
Enter the Number of an Action: 2
Enter length of side to delete Square: 3
BinTree Item: Deleted
Allocator: Deallocate
 No Node with this side!
Enter the Number of an Action: 4
[ 0 0 0 1 1 1 1 0 ] 1
[ 0 0 0 5 5 5 5 0 ] 5
[ 0 0 0 9 9 9 9 0 ] 9
Enter the Number of an Action: 0
BinTree: Deleted
BinTree Item: Deleted
BinTree Item: Deleted
BinTree Item: Deleted
Allocator: Deallocate
Allocator: Deallocate
Allocator: Deallocate

Process finished with exit code 0
```

4. Объяснение результатов программы

В файле lab6.cpp реализован интерфейс пользователя. В зависимости от команды выполняется :
0) Завершение программы
1) Инициализация вектора
2) Вывод координат центра по длине стороны
3) Вывод бинарного дерева
4) Прохождение фигур с помощью итератора + вывод длины стороны
5) Печать меню

В файлах **TAllocationBlock.h** и **TAllocationBlock.cpp** реализован аллокатор, использующий для хранения свободных блоков памяти связный список.
В файлах **TList.h, TList.cpp, TListItem.h, TlistItem.cpp** описан связный список для аллокатора.
В **TRectangle.h** и **TRectangle.cpp** описаны методы рассчета координат центра и площади фигуры.
В файлах **TStack.h, TStack.cpp, TStackItem.h, TStackItem.cpp** реализован стек для вершин бинарного дерева поиска.
Vector.cpp - собственная реализация вектора, как структуры данных + расчет площади фигуры и проверка на перпендикулярность.
TIterator.h – в этом файле описана перегрузка операторов.

Вывод: Выполняя данную лабораторную работу, я получила опыт работы с аллокаторами и умными указателями. Узнала о применении аллокаторов и научилась создавать контейнеры.