

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №8 по курсу объектно-ориентированное программирование I семестр, 2021/22 уч. год

Студент Михеева Кристина Олеговна, группа М8О-207Б-20
Преподаватель Дорохов Евгений Павлович

Условие:

Вариант 17: Используя структуру данных, разработанную для лабораторной работы No5, спроектировать и

разработать аллокатор памяти для динамической структуры данных.

Цель построения аллокатора – минимизация вызова операции malloc. Аллокатор должен выделять большие блоки памяти для хранения фигур и при создании новых фигур-объектов выделять место под объекты в этой памяти.

Алокатор должен хранить списки использованных/свободных блоков. Для хранения списка свободных блоков нужно применять динамическую структуру данных (контейнер 2-го уровня, согласно варианту задания).

Описание программы:

Исходный код лежит в 10 файлах:

1. main.cpp: основная программа, взаимодействие с пользователем посредством команд из меню.
2. figure.h: описание абстрактного класса фигуры.
3. point.h: описание класса точки.
4. point.cpp: реализация класса точки.
5. triangle.h: описание класса треугольника, наследующегося от figure.
6. triangle.cpp: реализация класса треугольника, наследующегося от figure.
7. TbinaryTree.cpp: реализация контейнера (бинарное дерево).
8. TBinaryTree.h: описание контейнера (бинарное дерево).
9. TbinaryTreeItem.cpp: реализация элемента бинарного дерева.
10. iTbinaryTreeItem.h: описание элемента бинарного дерева.
11. TAllocatorBlock.h: реализация класса алокатора бинарного дерева

Дневник отладки:

В данной лабораторной работе возникли проблемы с функциями Pop и Push, которые впоследствии были устранены. Также была устранена проблема с утечкой памяти.

Выводы:

В данной лабораторной работе мы познакомились с понятиями аллокатора. Аллокатор умеет выделять и освобождать память в требуемых количествах определенным образом. `std::allocator` -- пример реализации аллокатора из стандартной библиотеки, просто использует `new` и `delete`, которые обычно обращаются к системным вызовам `malloc` и `free`. Этот опыт с выполнением оказался очень полезным ведь аллокаторы нужны, чтобы вставлять свое, нестандартное, выделение памяти в любое место. Большинство стандартных контейнеров их принимают.

Листинг:

figure.h

```
#ifndef FIGURE_H
#define FIGURE_H

#include "point.h"

class Figure {
public:
    virtual size_t VertexesNumber() = 0;
    virtual double Area() = 0;
    virtual void Print (std::ostream &os) = 0;
    virtual ~Figure() {};
};

#endif
```

point.h

```
#ifndef POINT_H
#define POINT_H

#include <iostream>

class Point {
public:
    Point();
    Point(std::istream &is);
    Point(double x, double y);
    friend bool operator == (Point& p1, Point& p2);
    friend class Triangle;
    double getX();
    double getY();
};
```

```

        friend std::istream& operator>>(std::istream& is, Point& p);
        friend std::ostream& operator<<(std::ostream& os, Point& p);

private:
    double x;
    double y;
};

#endif

```

point.cpp

```

#include "point.h"

#include <cmath>

Point::Point() : x(0.0), y(0.0) {}

Point::Point(double x, double y) : x(x), y(y) {}

Point::Point(std::istream &is) {
    is >> x >> y;
}

double Point::getX() {
    return x;
};

double Point::getY() {
    return y;
};

std::istream& operator>>(std::istream& is, Point& p) {
    is >> p.x >> p.y;
    return is;
}

std::ostream& operator<<(std::ostream& os, Point& p) {
    os << "(" << p.x << ", " << p.y << ")";
    return os;
}

bool operator == (Point &p1, Point& p2) {
    return (p1.x == p2.x && p1.y == p2.y);
}

```

triangle.h

```

#include "figure.h"
#include <iostream>
#ifndef TRIANGLE_H
#define TRIANGLE_H
class Triangle : public Figure {

```

```

    public:
        Triangle(std:: istream &is);
        Triangle();
        size_t VertexesNumber();
        double Area();
        double GetArea();
        void Print (std:: ostream &os);
        virtual ~Triangle();
        friend bool operator == (Triangle& t1, Triangle& t2);
        friend std::ostream& operator << (std::ostream& os, Triangle& t);
        double area;

    private:
        Point a;
        Point b;
        Point c;
};
#endif

triangle.cpp
#include "triangle.h"
#include <cmath>
Triangle::Triangle() {}

Triangle::Triangle(std:: istream &is)
{
    is >> a >> b >> c;
    std:: cout << "The triangle was created" << std:: endl;
}

size_t Triangle::VertexesNumber()
{
    return 3;
}

double Triangle::Area() {
    double Square = 0.5 * abs(a.getX() * b.getY() + b.getX() * c.getY() +
c.getX() * a.getY() - a.getY() * b.getX() - b.getY() * c.getX() - c.getY() *
a.getX());
    this->area = Square;
    return Square;
}

void Triangle::Print(std:: ostream &os)
{
    std:: cout << "Triangle: " << a << " " << b << " " << c << std:: endl;
}

Triangle::~~Triangle() {
    std:: cout << "Trianle was deleted" << std:: endl;
}

```

```

}

double Triangle:: GetArea() {
    return area;
}

bool operator == (Triangle& t1, Triangle& t2){
    if(t1.a == t2.a && t1.b == t2.b && t1.c == t2.c){
        return true;
    }
    return false;
}

std::ostream& operator << (std::ostream& os, Triangle& t){
    os << "Triangle: ";
    os << t.a << t.b << t.c;
    os << std::endl;
    return os;
}

```

TBinaryTree.h

```

#ifndef TBINARYTREE_H
#define TBINARYTREE_H
#include "TBinaryTreeItem.h"

class TBinaryTree {
public:
    TBinaryTree();
    TBinaryTree(const TBinaryTree &other);
    void Push(Triangle &triangle);
    TBinaryTreeItem* Pop(TBinaryTreeItem* root, Triangle &triangle);
    Triangle& GetItemNotLess(double area, TBinaryTreeItem* root);
    void Clear();
    bool Empty();
    int Count(double minArea, double maxArea);
    friend std::ostream& operator<<(std::ostream& os, TBinaryTree& tree);
    virtual ~TBinaryTree();
    TBinaryTreeItem *root;
};
#endif

```

TBinaryTree.cpp

```

#include "TBinaryTreeItem.h"

```

```

TBinaryTreeItem::TBinaryTreeItem(const Triangle &triangle) {
    this->triangle = triangle;
    this->left = this->right = NULL;
    this->counter = 1;
}

TBinaryTreeItem::TBinaryTreeItem(const TBinaryTreeItem &other) {
    this->triangle = other.triangle;
    this->left = other.left;
    this->right = other.right;
    this->counter = other.counter;
}

Triangle& TBinaryTreeItem::GetTriangle() {
    return this->triangle;
}

void TBinaryTreeItem::SetTriangle(const Triangle& triangle){
    this->triangle = triangle;
}

TBinaryTreeItem* TBinaryTreeItem::GetLeft(){
    return this->left;
}

TBinaryTreeItem* TBinaryTreeItem::GetRight(){
    return this->right;
}

void TBinaryTreeItem::SetLeft(TBinaryTreeItem* item) {
    if (this != NULL){
        this->left = item;
    }
}

void TBinaryTreeItem::SetRight(TBinaryTreeItem* item) {
    if (this != NULL){
        this->right = item;
    }
}

void TBinaryTreeItem::IncreaseCounter() {
    if (this != NULL){
        counter++;
    }
}

void TBinaryTreeItem::DecreaseCounter() {
    if (this != NULL){
        counter--;
    }
}

int TBinaryTreeItem::ReturnCounter() {
    return this->counter;
}

```

```
TBinaryTreeItem::~TBinaryTreeItem() {
}
```

TBinaryTreeItem.h

```
#ifndef TBINARYTREE_ITEM_H
#define TBINARYTREE_ITEM_H
#include "triangle.h"

class TBinaryTreeItem {
public:
    TBinaryTreeItem(const Triangle& triangle);
    TBinaryTreeItem(const TBinaryTreeItem& other);
    Triangle& GetTriangle();
    void SetTriangle(Triangle& triangle);
    TBinaryTreeItem* GetLeft();
    TBinaryTreeItem* GetRight();
    void SetLeft(TBinaryTreeItem* item);
    void SetRight(TBinaryTreeItem* item);
    void SetTriangle(const Triangle& triangle);
    void IncreaseCounter();
    void DecreaseCounter();
    int ReturnCounter();
    virtual ~TBinaryTreeItem();

private:
    Triangle triangle;
    TBinaryTreeItem *left;
    TBinaryTreeItem *right;
    int counter;
};
#endif
```

TBinaryTreeItem.cpp

```
#include "TBinaryTreeItem.h"
```

```
TBinaryTreeItem::TBinaryTreeItem(const Triangle &triangle) {
    this->triangle = triangle;
    this->left = this->right = NULL;
    this->counter = 1;
```



```

}

TBinaryTreeItem::TBinaryTreeItem(const TBinaryTreeItem &other) {
    this->triangle = other.triangle;
    this->left = other.left;
    this->right = other.right;
    this->counter = other.counter;
}

Triangle& TBinaryTreeItem::GetTriangle() {
    return this->triangle;
}

void TBinaryTreeItem::SetTriangle(const Triangle& triangle){
    this->triangle = triangle;
}

TBinaryTreeItem* TBinaryTreeItem::GetLeft(){
    return this->left;
}

TBinaryTreeItem* TBinaryTreeItem::GetRight(){
    return this->right;
}

void TBinaryTreeItem::SetLeft(TBinaryTreeItem* item) {
    if (this != NULL){
        this->left = item;
    }
}

void TBinaryTreeItem::SetRight(TBinaryTreeItem* item) {
    if (this != NULL){
        this->right = item;
    }
}

void TBinaryTreeItem::IncreaseCounter() {
    if (this != NULL){
        counter++;
    }
}

void TBinaryTreeItem::DecreaseCounter() {
    if (this != NULL){
        counter--;
    }
}

int TBinaryTreeItem::ReturnCounter() {
    return this->counter;
}

TBinaryTreeItem::~TBinaryTreeItem() {
}

```

main.cpp

```
#include "triangle.h"
#include <iostream>
#include "TBinaryTree.h"
#include "TBinaryTreeItem.h"

int main () {
    Triangle a (std:: cin);
    std:: cout << "Area of a triangle:" << " " << a.Area() << std:: endl;

    Triangle b (std:: cin);
    std:: cout << "Area of a triangle:" << " " << b.Area() << std:: endl;

    Triangle c (std:: cin);
    std:: cout << "Area of a triangle:" << " " << c.Area() << std:: endl;

    TBinaryTree tree;
    std:: cout << "Is tree empty? " << tree.Empty() << std:: endl;
    tree.Push(a);
    std:: cout << "And now, is tree empty? " << tree.Empty() << std:: endl;
    tree.Push(b);
    tree.Push(c);
    std:: cout << "The number of figures with area in [minArea, maxArea] is: "
    << tree.Count(0, 100000) << std:: endl;
    std:: cout << "The result of searching the same-figure-counter is: " <<
    tree.root->ReturnCounter() << std:: endl;
    std:: cout << "The result of function named GetItemNotLess is: " <<
    tree.GetItemNotLess(0, tree.root) << std:: endl;
    std:: cout << tree << std:: endl;
    tree.root = tree.Pop(tree.root, a);
    std:: cout << tree << std:: endl;
    system("pause");
}
```

TAllocatorBlock

```
#ifndef TALLOCATORBLOCK_H
#define TALLOCATORBLOCK_H

#include "TLinkedList.h"
```

```

#include <memory>

class TAllocatorBlock {
public:
    TAllocatorBlock(const size_t& size, const size_t count){
        this->size = size;
        for(int i = 0; i < count; ++i){
            unused_blocks.Insert(malloc(size));
        }
    }
    void* Allocate(const size_t& size){
        if(size != this->size){
            std::cout << "Error during allocation\n";
        }
        if(unused_blocks.Length()){
            for(int i = 0; i < 5; ++i){
                unused_blocks.Insert(malloc(size));
            }
        }
        void* tmp = unused_blocks.GetItem(1);
        used_blocks.Insert(unused_blocks.GetItem(1));
        unused_blocks.Remove(0);
        return tmp;
    }
    void Deallocate(void* ptr){
        unused_blocks.Insert(ptr);
    }
    ~TAllocatorBlock(){
        while(used_blocks.size()){
            try{
                free(used_blocks.GetItem(1));
                used_blocks.Remove(0);
            } catch(...){
                used_blocks.Remove(0);
            }
        }
        while(unused_blocks.size()){
            try{
                free(unused_blocks.GetItem(1));
                unused_blocks.Remove(0);
            } catch(...){
                unused_blocks.Remove(0);
            }
        }
    }

private:
    size_t size;
    TLinkedList <void*> used_blocks;
    TLinkedList <void*> unused_blocks;
};

#endif

```


