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

ЛАБОРАТОРНАЯ РАБОТА №8

по курсу объектно-ориентированное программирование I семестр, 2021/22 уч. год

Студент: *Соколов Даниил Витальевич группа М8О-207Б-20*
Преподаватель: *Дорохов Евгений Павлович*

Условие

Задание: Вариант 23: N-арное дерево(Шестиугольник). Используя структуру данных, разработанную для лабораторной работы №5, 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 (контей- нер 2-го уровня, согласно варианту задания). Для вызова аллокатора должны быть переопределены оператор new и delete у классов-фигур.

Нельзя использовать:

- Стандартные контейнеры std.

Программа должна позволять:

- Вводить произвольное количество фигур и добавлять их в контейнер.
- Распечатывать содержимое контейнера.
- Удалять фигуры из контейнера.

Описание программы

Исходный код лежит в этих файлах:

1. TNaryTree.cpp
2. TNaryTree.h
3. TNaryTree_item.cpp
4. TNaryTree_item.h
5. figure.h
6. hexagon.cpp
7. hexagon.h
8. main.cpp
9. point.cpp
10. point.h
11. tallocation_block.cpp

12. tallocation_block.h
13. titerator.h
14. tqueen.cpp
15. tqueen.h
16. tqueen_item.cpp
17. tqueen_item.h

Дневник отладки

Ошибок не было

Недочёты

Недочётов не заметил.

Вывод

В данной лабораторной работе были реализованы аллокаторы классов. Задание не было сложным, так как основной код уже был написан в предыдущих работах. Аллокаторы удобны, когда необходимо придумать свои правила выделения памяти, а также снизить количество системных вызовов.

Исходный код

main.cpp

```
#include "TNaryTree.h"
#include "hexagon.h"
#include "titerator.h"
#include "TNaryTree_item.h"
#include "tallocation_block.h"
#include <string>

int main()
{
    TNaryTree<hexagon> a(4);
    if (a.Empty()) {
        std::cout << "The tree is empty !\n";
    } else {
        std::cout << "The tree is not empty !\n";
    }
    a.Update(std::shared_ptr<hexagon>(new hexagon(Point(1, 4), Point(1, 2), Point(5, 6), Point(2, 8),
    Point(3, 1), Point(2, 6))), ""); // 1
    a.Update(std::shared_ptr<hexagon>(new hexagon(Point(2, 5), Point(1, 5), Point(16, 6), Point(3, 6),
    Point(1, 8), Point(4, 2))), "c"); // 2

    a.Update(std::shared_ptr<hexagon>(new hexagon(Point(3, 5), Point(9, 1), Point(7, 3), Point(1, 8),
    Point(5, 6), Point(4, 8))), "cb"); // 3

    a.Update(std::shared_ptr<hexagon>(new hexagon(Point(8, 5), Point(1, 5), Point(16, 6), Point(3, 6),
    Point(1, 8), Point(4, 2))), "cbc"); // 4


    for (auto i: a) {
        std::cout << *i << std::endl;
    }
    std::cout << a;
    std::cout << a.Area("cb") << "\n";
    TNaryTree<hexagon> b(a);
    std::cout << b;
    std::shared_ptr<hexagon> c = a.GetItem("");
    std::cout << *c;
    a.RemoveSubTree("cbc");
    if (a.Empty()) {
        std::cout << "The tree is empty !\n";
    } else {
        std::cout << "The tree is not empty !\n";
    }
    std::cout << "Allocation test:\n";
    TAllocationBlock block(sizeof(int), 10);
    int* n1;
    int* n2;
```

```
    int* n3;
    n1 = (int*)block.allocate();
    n2 = (int*)block.allocate();
    n3 = (int*)block.allocate();
    *n1 = 10; *n2 = 100; *n3 = 1000;
    std::cout << *n1 << " " << *n2 << " " << *n3 << "\n";
    if (block.has_free_blocks()) {
        std::cout << "Free blocks are avaible !\n";
    } else {
        std::cout << "Free blocks are not avaible!\n";
    }
    return 0;
}
```

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;
            ~Figure() {};
};

#endif
```

hexagon.h

```
#ifndef OCTAGON_H
#define OCTAGON_H

#include "point.h"
#include "figure.h"

class hexagon : figure
{
public:
    hexagon(std::istream& is);
    hexagon();
    ~hexagon();
    hexagon(Point a, Point b, Point c, Point d, Point e, Point f);
```

```
    size_t VertexesNumber();
    double Area();
    void Print(std::ostream& os);

    hexagon& operator=(const hexagon& other);
    bool operator==(hexagon& other);
    friend std::ostream& operator<<(std::ostream& os, hexagon& other);
    friend std::istream& operator>>(std::istream& is, hexagon& other);

private:
    Point a_, b_, c_, d_;
    Point e_, f_;
};

#endif
```

hexagon.cpp

```
#include "hexagon.h"
#include "point.h"

hexagon::hexagon(std::istream& is)
{
    std::cin >> a_ >> b_ >> c_ >> d_;
    std::cin >> e_ >> f_;
}

hexagon::hexagon() : a_(0,0), b_(0,0), c_(0,0), d_(0, 0), e_(0,0), f_(0,0)
{}

hexagon::hexagon(Point a, Point b, Point c, Point d, Point e, Point f)
{
    this->a_ = a; this->b_ = b;
    this->c_ = c; this->d_ = d;
    this->e_ = e; this->f_ = f;
}

size_t hexagon::VertexesNumber()
{
    return (size_t)6;
}

double hexagon::Area()
{
    return 0.5 * abs((a_.getX() * b_.getY() + b_.getX() * c_.getY() + c_.getX() * d_.getY() + d_.getX() * e_.getY()
+ e_.getX() * f_.getY()
    - (b_.getX() * a_.getY() + c_.getX() * b_.getY() +
    d_.getX() * c_.getY() + e_.getX() * d_.getY() + f_.getX() * e_.getY())));
}
```

```
hexagon& hexagon::operator=(const hexagon& other)
{
    this->a_ = other.a_; this->b_ = other.b_;
    this->c_ = other.c_; this->d_ = other.d_;
    this->e_ = other.e_; this->f_ = other.f_;
    return *this;
}

bool hexagon::operator==(hexagon& other)
{
    return this->a_ == other.a_ && this->b_ == other.b_ &&
    this->c_ == other.c_ && this->d_ == other.d_ &&
    this->e_ == other.e_ && this->f_ == other.f_;

}

std::ostream& operator<<(std::ostream& os, hexagon& oct)
{
    os << "Hexagon: " << oct.a_ << " " << oct.b_ << " ";
    os << oct.c_ << " " << oct.d_ << " " << oct.e_ << " ";
    os << oct.f_ << "\n";
    return os;
}

std::istream& operator>>(std::istream& is, hexagon& other)
{
    is >> other.a_ >> other.b_ >> other.c_ >> other.d_;
    is >> other.e_ >> other.f_;
    return is;
}

void hexagon::Print(std::ostream& os)
{
    std::cout << "Octagon: " << a_ << " " << b_ << " ";
    std::cout << c_ << " " << d_ << " " << e_ << " ";
    std::cout << f_ << "\n";
}

hexagon::~hexagon(){}
```

tallocation_block.cpp

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

TAllocationBlock::TAllocationBlock(size_t size, size_t count): _size(size), _count(count)
{
    _used_blocks = (char*)malloc(size * count);
```

```
    for (size_t i = 0; i < count; i++) {
        _free_blocks.Push(_used_blocks + i * size);
    }
    _free_count = count;
    std::cout << "Memory init" << "\n";
}

void* TAllocationBlock::allocate()
{
    void* result = nullptr;
    if (_free_count == 0) {
        std::cout << "No memory exception\n" << "\n";
        return result;
    }
    result = _free_blocks.Top();
    _free_blocks.Pop();
    --_free_count;
    std::cout << "Allocate " << (_count - _free_count) << "\n";
    return result;
}

void TAllocationBlock::deallocate(void* pointer)
{
    _free_blocks.Push(pointer);
    ++_free_count;
    std::cout << "Deallocated block\n";
}

bool TAllocationBlock::has_free_blocks()
{
    return _free_count > 0;
}

TAllocationBlock::~TAllocationBlock()
{
    free(_used_blocks);
}
```

tallocation_block.h

```
#ifndef TALLOCATION_BLOCK_H
#define TALLOCATION_BLOCK_H

#include <cstdlib>
#include "tqueen.h"

class TAllocationBlock
```

```
{
public:
    TAllocationBlock(size_t size, size_t count);
    void* allocate();
    void deallocate(void* pointer);
    bool has_free_blocks();
    virtual ~TAllocationBlock();

private:
    size_t _size;
    size_t _count;
    char* _used_blocks;
    TQueen<void*> _free_blocks;
    size_t _free_count;
};

#endif
```

tqueue.h

```
#ifndef TQUEUE_H
#define TQUEUE_H

#include <iostream>
#include <memory>
#include "tqueen_item.h"

template <class T>
class TQueen
{
public:
    TQueen();
    virtual ~TQueen();
    void Push(const T &item);
    void Pop();
    T &Top();
    bool IsEmpty() const;
    uint32_t GetSize() const;
    template <class A> friend std::ostream& operator<<(std::ostream &os, const TQueen<A>
&stack);

private:
    TQueenItem<T> *head;
    uint32_t count;
};

#endif
```

tqueue.cpp

```
#include <iostream>
#include <memory>
#include "tqueen.h"

template <class T>
TQueen<T>::TQueen()
{
   head = nullptr;
   count = 0;
}

template <class T>
void TQueen<T>::Push(const T &item)
{
   TQueenItem<T> *tmp = new TQueenItem<T>(item, head);
   head = tmp;
   ++count;
}

template <class T>
bool TQueen<T>::IsEmpty() const
{
   return !count;
}

template <class T>
uint32_t TQueen<T>::GetSize() const
{
   return count;
}

template <class T>
void TQueen<T>::Pop()
{
```

```
    if(head) {
        TQueenItem<T> *tmp = &head->GetNext();
        delete head;
        head = tmp;
        --count;
    }
}

template <class T>
T &TQueen<T>::Top()
{
    return head->Pop();
}

template <class T>
TQueen<T>::~TQueen()
{
    for(TQueenItem<T> *tmp = head, *tmp2; tmp; tmp = tmp2) {
        tmp2 = &tmp->GetNext();
        delete tmp;
    }
}

template class
TQueen<void *>;
```