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

ЛАБОРАТОРНАЯ РАБОТА №6 по курсу

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

Студент *Васютинский Вадим Александрович, группа М80-208Б-20*
Преподаватель *Дорохов Евгений Павлович*

Цель работы:

Целью лабораторной работы является:

Закрепление навыков по работе с памятью в C++;
Создание аллокаторов памяти для динамических структур данных.

Задание:

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. Аллокатор должен хранить списки использованных/свободных блоков. Для хранения списка свободных блоков нужно применять динамическую структуру данных (контейнер 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.

Дневник отладки

Данная лабораторная работа превратилась для меня в личный ад. Каждая написанная мной реализация работала в определённых случаях, но в определённый момент вылетала. Итоговый вариант, есть сплав всех неудачных версий.

Недочёты

Из-за недостаточно продолжительной отладки я могу сомневаться в абсолютно безопасной работе программы, но на проведённых мной тестах программа выдавала правильный результат.

Выводы

Лабораторная работа №6 позволила мне реализовать свой класс аллокаторов, полностью прочувствовать процесс выделения памяти на низкоуровневых языках программирования. Лабораторная прошла успешно. В процессе выполнения работы я на практике познакомился с понятием аллокатора. Так как во многих структурах данных используются аллокаторы, то это очень важная тема, которую должен знать каждый программист на C++. Написание собственноручного аллокатора помогает реализовать собственную логику выделения памяти, которая может быть более оправданной в некоторых ситуациях, чем стандартный аллокатор, как для самописных, так и для стандартных структур данных.

Исходный код

figure.h

```
#ifndef OOP5_FIGURE_H
#define OOP5_FIGURE_H

#include <cmath>
#include <iostream>
#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 //OOP5_FIGURE_H
```

main.cpp

```
#include "rectangle.h"
#include "TVector.h"
#include <memory>
#include <string>

int main() {
    std::string command;
    TVector<Rectangle> v;

    while (std::cin >> command) {
        if (command == "print")
            std::cout << v;
        else if (command == "insertlast") {
            Rectangle r;
            std::cin >> r;
            std::shared_ptr<Rectangle> d(new Rectangle(r));
            v.InsertLast(d);
        }
        else if (command == "removelast") {
            v.RemoveLast();
        }
        else if (command == "last") {
            std::cout << v.Last();
        }
        else if (command == "idx") {
            int idx;
            std::cin >> idx;
            std::cout << v[idx];
        }
        else if (command == "clear") {
            v.Clear();
        }
        else if (command == "empty") {
            if (v.Empty()) std::cout << "Yes" << std::endl;
            else std::cout << "No" << std::endl;
        }
        else if (command == "printall") {
```

```
            for (auto rect: v) {
                std::cout << rect;
            }
            std::cout << std::endl;
        }
    }
    return 0;
}
```

rectangle.cpp

```
#include "rectangle.h"

std::istream& operator>>(std::istream& is, Rectangle& r) {
    std::cout << "Enter data: " << std::endl;
    is >> r.a >> r.b >> r.c >> r.d;
    return is;
}

std::ostream& operator<<(std::ostream& os, Rectangle& r) {
    os << "Pentagon: " << r.a << r.b << r.c << r.d;
    return os;
}

Rectangle& Rectangle::operator=(const Rectangle &other) {
    this->a = other.a;
    this->b = other.b;
    this->c = other.c;
    this->d = other.d;
    return *this;
}
```

rectangle.h

```
#ifndef OOP1_RECTANGLE_H
#define OOP1_RECTANGLE_H

#include "figure.h"
class Rectangle : Figure {
public:
    friend std::istream& operator>>(std::istream& is, Rectangle& p);
    friend std::ostream& operator<<(std::ostream& os, Rectangle& p);
    bool operator==(const Rectangle& other);
    Rectangle& operator=(const Rectangle& other);
```

```
    void Print(std::ostream &os) override;
    size_t VertexesNumber() override;
    double Area() override;
    Rectangle();
    Rectangle(Point a, Point b, Point c, Point d);
    Rectangle(std::istream &is);
    Rectangle(const Rectangle &other);
    virtual ~Rectangle();
private:
    Point a, b, c, d;
};
#endif //OOP1_RECTANGLE_H
```

Point.cpp

```
#include "point.h"

#include <cmath>

bool Point::operator==(const Point &other) {
    return (this->x_ == other.x_ && this->y_ == other.y_);
}

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::dist(Point& other) {
    double dx = (other.x_ - x_);
    double dy = (other.y_ - y_);
    return std::sqrt(dx*dx + dy*dy);
}

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;
}
```

Point.h

```
#ifndef OOP5_POINT_H
#define OOP5_POINT_H
#include <iostream>

class Point {
public:
    Point();
    Point(std::istream &is);
    Point(double x, double y);

    double dist(Point& other);

    friend std::istream& operator>>(std::istream& is, Point& p);
    friend std::ostream& operator<<(std::ostream& os, Point& p);
    bool operator==(const Point& other);
private:
    double x_;
    double y_;
};

#endif //OOP5_POINT_H
```

TVector.cpp

```
//
// Created by queens_gambit on 16.01.2022.
//

#include "TVector.h"
#include "rectangle.h"
#include <cassert>

template <typename T>
TVector<T>::TVector()
        : data_(new std::shared_ptr<T>[32]),
          length_(0), capacity_(32) {}

template <typename T>
TVector<T>::TVector(const TVector &vector)
        : data_(new std::shared_ptr<T>[vector.capacity_]),
          length_(vector.length_), capacity_(vector.capacity_) {
    std::copy(vector.data_, vector.data_ + vector.length_, data_);
}

template <typename T>
```

```
TVector<T>::~TVector() {
    delete[] data_;
}

template <typename T>
void TVector<T>::_Resize(const size_t new_capacity) {
    std::shared_ptr<T> *newdata = new std::shared_ptr<T>[new_capacity];
    std::copy(data_, data_ + capacity_, newdata);
    delete[] data_;
    data_ = newdata;
    capacity_ = new_capacity;
}

template <typename T>
void TVector<T>::InsertLast(const std::shared_ptr<T> &item) {
    if (length_ >= capacity_)
        _Resize(capacity_ << 1);
    data_[length_++] = item;
}

template <typename T>
void TVector<T>::EmplaceLast(const T &&item) {
    if (length_ >= capacity_)
        _Resize(capacity_ << 1);
    data_[length_++] = std::make_shared<T>(item);
}

template <typename T>
void TVector<T>::Remove(const size_t index) {
    std::copy(data_ + index + 1, data_ + length_, data_ + index);
    --length_;
}

template <typename T>
void TVector<T>::Clear() {
    delete[] data_;
    data_ = new std::shared_ptr<T>[32];
    length_ = 0;
    capacity_ = 32;
}

template <typename T>
std::ostream &operator<<(std::ostream &os, const TVector<T> &vector) {
    for (size_t i = 0; i < vector.length_; ++i)
        os << (*vector.data_[i]);
    os << std::endl;
    return os;
}
```

```
template class TVector<Rectangle>;
template std::ostream& operator<<(std::ostream& os, const TVector<Rectangle >& arr);
```

TIterator.h

```
#ifndef OOP5_TITERATOR_H
#define OOP5_TITERATOR_H
#include <memory>

#include <memory>

template <typename T>
class TIterator {
public:
    TIterator(std::shared_ptr<T> *iter) : iter_(iter) {}

    T operator*() const {
        return *(*iter_);
    }

    T operator->() const {
        return *(*iter_);
    }

    void operator++() {
        iter_ += 1;
    }

    TIterator operator++(int) {
        TIterator iter(*this);
        ++(*this);
        return iter;
    }

    bool operator==(TIterator const &iterator) const {
        return iter_ == iterator.iter_;
    }

    bool operator!=(TIterator const &iterator) const {
        return iter_ != iterator.iter_;
    }

private:
    std::shared_ptr<T> *iter_;
```

```
};
#endif //OOP5_TITERATOR_H
```

TAllocationBlock.h

```
#ifndef OOP6_TALLOCATIONBLOCK_H
#define OOP6_TALLOCATIONBLOCK_H

#include "TStack.h"
class TAllocationBlock {
public:
    TAllocationBlock(size_t size, size_t count);

    ~TAllocationBlock();

    void *Allocate(size_t size);

    void Free(void *pointer);

    bool FreeBlocksAvailable() const {
        return budget;
    }

private:
    void Resize(size_t new_count);

    size_t size;
    size_t count;
    size_t budget;

    char *used_blocks;
    TStack<void *> free_blocks;
};


#endif //OOP6_TALLOCATIONBLOCK_H
```

TAllocationBlock.cpp

```
#include "TAllocationBlock.h"

#include <iostream>

TAllocationBlock::TAllocationBlock(size_t size, size_t count) : size(size), count(count),
budget(count), used_blocks(new char[size * count]) {
    for (size_t i = 0; i < count; ++i)
        free_blocks.Emplace((void *)(used_blocks + (i * size)));
}
```

```
TAllocationBlock::~TAllocationBlock() {
    delete[] used_blocks;
}

void TAllocationBlock::Resize(size_t new_count) {
    char *newdata = new char[size * new_count];
    std::copy(used_blocks, used_blocks + (size * count), newdata);
    delete[] used_blocks;
    used_blocks = newdata;
    count = new_count;
}

void *TAllocationBlock::Allocate(size_t size) {
    if (size != this->size) {
        std::cerr << "This block allocates " << this->size << "bytes only. "
            << "You tried to allocate " << size << std::endl;
        return 0;
    }

    if (!budget) {
        size_t old_count = count;
        Resize(count << 1);
        budget += (count - old_count);

        for (size_t i = old_count; i < count; ++i) {
            free_blocks.Emplace((void*)(used_blocks + (i * size)));
        }
    }
    --budget;

    return free_blocks.Pop();
}

void TAllocationBlock::Free(void *pointer) {
    free_blocks.Push(std::make_shared<void *>(pointer));
    ++budget;
}
```

TStack.h

```
#include <memory>
#include <cstdlib>

template <typename T>
class TStack {
public:
    TStack(const TStack &);
```

```
    virtual ~TStack();

    size_t Length() const {
        return length;
    }

    bool Empty() const {
        return !length;
    }

    std::shared_ptr<T> &Top() const {
        return data[length];
    }

    void Emplace(const T &&);

    void Push(const std::shared_ptr<T> &);

    T Pop() {
        return *data[--length];
    }

    void Clear();

    template<typename TF>
    friend std::ostream &operator<<(std::ostream &, const TStack<TF> &);

    TStack();

private:
    void Resize(const size_t new_capacity);

    std::shared_ptr<T> *data;

    size_t length, capacity;
};


#endif //OOP6_TSTACK_H
```