**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 6**

Тема: Основы работы с коллекциями: итераторы, аллокаторы.

Студент: Минибаев Айдар

Группа: 80-201Б-18

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2019

1. **Постановка задачи**

Вариант 14:

|  |  |  |  |
| --- | --- | --- | --- |
| 3. | Прямоугольник | Стек | Стек |

1. **Описание программы**

Программа состоит из 5 файлов:

1. main.cpp
2. mystack.h - реализация стека для хранения фигур
3. stack.h - реализация стека для хранения пустых блоков памяти
4. myallocator.h - реализация аллокатора и его основных методов
5. rectangle.h - реализация прямоугольника и его методов

Аллокатор имеет 2 шаблонных параметра: тип хранимых данных и размер памяти в байтах. Аллокатор хранит стек свободных блоков. В случае когда часть занятой памяти освобождают, то ссылка на данный блок памяти хранится в стеке. Фигура добавляется в самый правый незаполненный участок памяти, то есть после хвоста. Если свободного участка памяти после хвоста и до конца участка памяти будет недостаточно, чтобы сохранить фигуру, то в таком случае фигура будет сохранена в свободный блок памяти если он есть в стеке. Иначе программа сообщит, что места недостаточно. Данный аллокатор подходит для фигур, которые имеют одинаковый размер, потому что каждый свободный блок как раз равен размеру фигуры.

1. **Набор тестов**

В двух тестах зададим разный размер памяти при создании стека, а также может добавить фигуру в стек [Add], в верх стека[Push] или после определенного элемента по индексу с помощью итератора [Iter], выводить все фигуры, содержащиеся в стеке[print]. Удалять фигуру в стеке[Rem]: верхнюю фигуру[Pop], либо удалять по указанному индексу с помощью итератора [Item]. Также пользователь может с вызвать функцию [count\_if] и задать площадь, после чего будет указано количество фигур, с площадью меньше заданного числа. Вызов меню[Menu]. Выход и программы [Ex]. В случае если памяти для добавления фигуры будет недостаточно, выйдет исключение.

*test\_01.txt*:

В коде программы зададим размер стека 168 байт, которых хватит для хранения 3 фигур

“Stack<Rectangle<int>,my\_allocator<Rectangle<int>, 168>> t;”

Add Push

0 0 3 0 3 5 0 5

Add Push

-5 -5 0 -5 0 0 -5 0

Add Iter

1 1 3 1 3 6 1 6

1

print

Add Push

0 0 3 0 3 8 0 8

Rem Pop

Add Push

10 10 12 10 12 17 10 17

count\_if 16

print

*test\_02.txt*:

В коде программы зададим размер стека 112 байт, которых хватит для хранения 2 фигур.

“Stack<Rectangle<int>,my\_allocator<Rectangle<int>, 112>> t;”

Add Push

0 0 9 0 9 15 0 15

Add Iter

-5 -5 0 -5 0 7 -5 7

1

Top

count\_if 12

print

count\_if 140

Add Push

12 12 19 12 19 20 12 20

Rem Iter 2

print

Add Push

0 0 -40 50 3 3 4 9

1. **Результаты выполнения тестов.**

***test\_01.txt*:**

The size of one rectangle in the stack 56

Enter a word for action:

1) Add Rectangle to the stack[Add]

2) Remove item [Rem]

3) Show all shapes with std::for\_each [print]

4) Show top stack element [Top]

5) Display on the screen the number of objects whose area is less than the specified with std::count\_if [count\_if]

6) Print Menu[Menu]

7) Exit[Ex]

Add

Add an item to the top of the stack[Push] or to the iterator position[Iter]

Push

cords point A: 0 0

cords point B: 3 0

cords point C: 3 5

cords point D: 0 5

Add

Add an item to the top of the stack[Push] or to the iterator position[Iter]

Push

cords point A: -5 -5

cords point B: 0 -5

cords point C: 0 0

cords point D: -5 0

Add

Add an item to the top of the stack[Push] or to the iterator position[Iter]

Iter

cords point A: 1 1

cords point B: 3 1

cords point C: 3 6

cords point D: 1 6

Enter the index of the item, after which insert the item

1

print

1.

Cords of Rectangle: <-5, -5>, <0, -5>, <0, 0>, <-5, 0>

Rectangle area 25

2.

Cords of Rectangle: <1, 1>, <3, 1>, <3, 6>, <1, 6>

Rectangle area 10

3.

Cords of Rectangle: <0, 0>, <3, 0>, <3, 5>, <0, 5>

Rectangle area 15

Add

Add an item to the top of the stack[Push] or to the iterator position[Iter]

Push

cords point A: 0 0

cords point B: 3 0

cords point C: 3 8

cords point D: 0 8

std::bad\_alloc

Memory limit

You did not select an action or you were mistaken in entering the name of the action

Rem

Delete item on top of stack[Pop] or to the iterator position[Iter]

Pop

Add

Add an item to the top of the stack[Push] or to the iterator position[Iter]

Push

cords point A: 10 10

cords point B: 12 10

cords point C: 12 17

cords point D: 10 17

count\_if

Enter area

16

The number of figures with an area less than a given 3

print

1.

Cords of Rectangle: <10, 10>, <12, 10>, <12, 17>, <10, 17>

Rectangle area 14

2.

Cords of Rectangle: <1, 1>, <3, 1>, <3, 6>, <1, 6>

Rectangle area 10

3.

Cords of Rectangle: <0, 0>, <3, 0>, <3, 5>, <0, 5>

Rectangle area 15

***test\_02.txt*:**

The size of one rectangle in the stack 56

Enter a word for action:

1) Add Rectangle to the stack[Add]

2) Remove item [Rem]

3) Show all shapes with std::for\_each [print]

4) Show top stack element [Top]

5) Display on the screen the number of objects whose area is less than the specified with std::count\_if [count\_if]

6) Print Menu[Menu]

7) Exit[Ex]

Add

Add an item to the top of the stack[Push] or to the iterator position[Iter]

Push

cords point A: 0 0

cords point B: 9 0

cords point C: 9 15

cords point D: 0 15

Add

Add an item to the top of the stack[Push] or to the iterator position[Iter]

Iter

cords point A: -5 -5

cords point B: 0 -5

cords point C: 0 7

cords point D: -5 7

Enter the index of the item, after which insert the item

1

Top

Cords of Rectangle: <0, 0>, <9, 0>, <9, 15>, <0, 15>

count\_if 12

Enter area

The number of figures with an area less than a given 0

print

1.

Cords of Rectangle: <0, 0>, <9, 0>, <9, 15>, <0, 15>

Rectangle area 135

2.

Cords of Rectangle: <-5, -5>, <0, -5>, <0, 7>, <-5, 7>

Rectangle area 60

count\_if 140

Enter area

The number of figures with an area less than a given 2

Add

Add an item to the top of the stack[Push] or to the iterator position[Iter]

Push

cords point A: 12 12

cords point B: 19 12

cords point C: 19 20

cords point D: 12 20

std::bad\_alloc

Memory limit

You did not select an action or you were mistaken in entering the name of the action

Rem

Delete item on top of stack[Pop] or to the iterator position[Iter]

Iter

Enter the index of the item to be deleted

2

print

1.

Cords of Rectangle: <0, 0>, <9, 0>, <9, 15>, <0, 15>

Rectangle area 135

Add

Add an item to the top of the stack[Push] or to the iterator position[Iter]

Push

cords point A: 0 0

cords point B: -40 50

cords point C: 3 3

cords point D: 4 9

libc++abi.dylib: terminating with uncaught exception of type std::overflow\_error: The entered coordinates of the vertices do not belong to the rectangle.

Process finished with exit code 6

1. **Листинг программы**

**main.cpp**

/\*

\* Лабараторная работа 6

\* Минибаев Айдар М8О-201Б

\* 3 вариант: Прямоугольник, стек, стек

\*/

#include <algorithm>

#include <iostream>

#include "stack.h"

#include "rectangle.h"

#include "myallocator.h"

template <class T>

void Rectangle\_read\_cords(std::pair<T, T> &A, std::pair<T, T> &B, std::pair<T, T> &C, std::pair<T, T> &D){

std::cout << "cords point A: ";

std::cin >> A;

std::cout << "cords point B: ";

std::cin >> B;

std::cout << "cords point C: ";

std::cin >> C;

std::cout << "cords point D: ";

std::cin >> D;

}

int main() {

int index, val;

Stack<Rectangle<int>,my\_allocator<Rectangle<int>,112>> t;

std::string option = "Menu";

t.size\_node\_byte();

while(option != "Ex"){

if (option == "Add") {

try{

std::cout << "Add an item to the top of the stack[Push] or to the iterator position[Iter]" << std::endl;

std::cin >> option;

std::pair<int, int> A, B, C, D;

Rectangle\_read\_cords(A, B, C, D);

Rectangle<int> Rec(A, B, C, D);

if (option == "Push") {

t.Push(Rec);

} else if (option == "Iter") {

std::cout << "Enter the index of the item, after which insert the item" << std::endl;

std::cin >> index;

try {

auto it = t.begin();

for (int i = 0; i < index-1; ++i) {

++it;

}

t.insert(it, Rec);

} catch (std::logic\_error &e) {

std::cout << e.what() << std::endl;

continue;

}

}

}catch (std::bad\_alloc& e){

std::cout << e.what() << std::endl;

std::cout << "Memory limit\n";

continue;

}

} else if(option == "Rem") {

std::cout << "Delete item on top of stack[Pop] or to the iterator position[Iter]" << std::endl;

std::cin >> option;

if (option == "Pop") {

try {

t.Pop();

} catch (std::logic\_error &e) {

std::cout << e.what() << std::endl;

continue;

}

} else if (option == "Iter") {

std::cout << "Enter the index of the item to be deleted" << std::endl;

std::cin >> index;

try {

if (index < 0 || index > t.Size) {

throw std::logic\_error("Out of bounds\n");

}

auto it = t.begin();

for (int i = 0; i < index-1; ++i) {

it++;

}

t.erase(it);

} catch (std::logic\_error &e) {

std::cout << e.what() << std::endl;

continue;

}

}

}else if(option == "print") {

int num = 1;

std::for\_each(t.begin(), t.end(), [&num](auto e) {

std::cout << num << ". " << std::endl;

e.Coord\_print(std::cout);

std::cout << "Rectangle area " << e.area\_ << std::endl;

num++;

});

} else if(option == "Top") {

try {

t.Top().Coord\_print(std::cout);

} catch (std::logic\_error &e) {

std::cout << e.what() << std::endl;

continue;

}

} else if(option == "count\_if"){

std::cout << "Enter area" << std::endl;

std::cin >> val;

std::cout << "The number of figures with an area less than a given "<< std::count\_if(t.begin(),t.end(), [val](Rectangle<int> Rec)

{return Rec.area\_ < val;}) << std::endl;

} else if (option == "Menu") {

std::cout << "Enter a word for action:" << std::endl;

std::cout << "1) Add Rectangle to the stack[Add]" <<

std::endl << "2) Remove item [Rem]" <<

std::endl << "3) Show all shapes with std::for\_each [print]" <<

std::endl << "4) Show top stack element [Top]" <<

std::endl << "5) Display on the screen the number of objects whose area is"

" less than the specified with std::count\_if [count\_if]" <<

std::endl << "6) Print Menu[Menu]" <<

std::endl << "7) Exit[Ex]" << std::endl;

} else {

std::cout << "You did not select an action or you were mistaken in entering the name of the action" << std::endl;

}

std::cin >> option;

}

return 0;

}

**mystack.h**

#ifndef UNTITLE06\_STACK\_H

#define UNTITLE06\_STACK\_H

#include <iterator>

#include <iostream>

#include <memory> //for smart ptr

template<class T>

struct Stack {

private:

struct Stack\_node;

public:

size\_t Size = 0;

Stack() = default;

//Stack(const Stack &) = delete; // для копирования

struct forward\_iterator {

using value\_type = T; // using делается для совместимости со станд алг-ми STL

using reference = T &;

using pointer = T \*;

using difference\_type = std::ptrdiff\_t; //значения смещений итераторов относительно друг друга

using iterator\_category = std::forward\_iterator\_tag;

forward\_iterator(Stack\_node \*ptr);

T &operator\*();//разименование

forward\_iterator &operator++(); // ++it

forward\_iterator operator++(int); //it++

bool operator==(const forward\_iterator &o) const;

bool operator!=(const forward\_iterator &o) const;// писат конст желательно для конст объектов и результатов

private:

Stack\_node \*ptr\_; //итератор должен иметь указатель на узел стэка

friend struct Stack;

};

forward\_iterator begin();

forward\_iterator end();

T &Top();

void insert(const forward\_iterator &it, const T &value);//вставка

void erase(const forward\_iterator &it); //удаление

void Push(const T &value);

void Pop();

private:

struct Stack\_node {

T value\_; //потом подправить под фигуру

std::unique\_ptr<Stack\_node> next\_ = nullptr;

forward\_iterator next();

Stack\_node() = default;

Stack\_node(const T &value, std::unique\_ptr<Stack\_node> next) : value\_(value), next\_(std::move(next)) {};

};

std::unique\_ptr<Stack\_node> head\_ = nullptr;

};

template<class T>

typename Stack<T>::forward\_iterator Stack<T>::Stack\_node::next() {

return { next\_.get()};

}

template<class T>

typename Stack<T>::forward\_iterator Stack<T>::begin() {

if (head\_ == nullptr) {

return nullptr;

}

return head\_.get();

}

//typename используем, чтобы компилятор понял, что это

template<class T>

typename Stack<T>::forward\_iterator Stack<T>::end() {

return nullptr;

}

template<class T>

Stack<T>::forward\_iterator::forward\_iterator(Stack\_node \*ptr): ptr\_(ptr) {}

template<class T>

T &Stack<T>::forward\_iterator::operator\*() {

return ptr\_->value\_;

}

template<class T>

typename Stack<T>::forward\_iterator &Stack<T>::forward\_iterator::operator++() {

\*this = ptr\_->next();

return \*this;

}

template<class T>

typename Stack<T>::forward\_iterator Stack<T>::forward\_iterator::operator++(int) {

forward\_iterator old = \*this;

++\*this;

return old;

}

template<class T>

bool Stack<T>::forward\_iterator::operator==(const forward\_iterator &o) const {

return ptr\_ == o.ptr\_;

}

template<class T>

bool Stack<T>::forward\_iterator::operator!=(const forward\_iterator &o) const {

return ptr\_ != o.ptr\_;

}

template<class T>

void Stack<T>::insert(const forward\_iterator &it, const T &value) {

std::unique\_ptr<Stack\_node> new\_node(new Stack\_node(value, nullptr));

if (it.ptr\_ == nullptr && Size != 0) {

throw std::logic\_error("iterator went beyond the bounds of the stack");

} else if (it.ptr\_ == nullptr && Size == 0) {

head\_ = std::unique\_ptr<Stack\_node>(new Stack\_node(value, nullptr));

++Size;

} else {

new\_node->next\_ = std::move(it.ptr\_->next\_);

it.ptr\_->next\_ = std::move(new\_node);

++Size;

}

}

template<class T>

void Stack<T>::erase(const typename Stack<T>::forward\_iterator &it) {

if (it.ptr\_ == nullptr) {

throw std::logic\_error("erasing invalid iterator");

} else if (it == this->begin()) {

head\_ = std::move(it.ptr\_->next\_);

--Size;

} else {

auto it2 = this->begin(); // поиск элемента перед элементом, на который указывает итератор

while (it2.ptr\_->next() != it.ptr\_) {

++it2;

}

it2.ptr\_->next\_ = std::move(it.ptr\_->next\_);

}

}

template<class T>

void Stack<T>::Push(const T &value) {

std::unique\_ptr<Stack\_node> new\_node(new Stack\_node(value, nullptr));

new\_node->next\_ = std::move(head\_);

head\_ = std::move(new\_node);

++Size;

}

template<class T>

T &Stack<T>::Top() {

if (head\_.get()) {

return head\_->value\_;

} else {

throw std::logic\_error("Stack is empty");

}

}

template<class T>

void Stack<T>::Pop() {

if (head\_ == nullptr) {

throw std::logic\_error("Stack is empty");

} else {

head\_ = std::move(head\_->next\_);

--Size;

}

}

#endif //UNTITLE06\_STACK\_H

**stack.h**

#ifndef UNTITLE06\_STACK\_H

#define UNTITLE06\_STACK\_H

#include <iterator>

#include <memory>

template<class T, class Allocator = std::allocator<T>>

struct Stack {

private:

struct Stack\_node;

public:

size\_t Size = 0;

Stack() = default;

//Stack(const Stack &) = delete; // для копирования

struct forward\_iterator {

using value\_type = T; // using делается для совместимости со станд алг-ми STL

using reference = T &;

using pointer = T \*;

using difference\_type = std::ptrdiff\_t; //значения смещений итераторов относительно друг друга

using iterator\_category = std::forward\_iterator\_tag;

forward\_iterator(Stack\_node \*ptr);

T &operator\*();//разименование

forward\_iterator &operator++(); // ++it

forward\_iterator operator++(int); //it++

bool operator==(const forward\_iterator &o) const;

bool operator!=(const forward\_iterator &o) const;// писат конст желательно для конст объектов и результатов

private:

Stack\_node \*ptr\_; //итератор должен иметь указатель на узел стэка

friend struct Stack;

};

void size\_node\_byte() {

std::cout << "The size of one rectangle in the stack " << sizeof(Stack::Stack\_node) << std::endl;

}

forward\_iterator begin();

forward\_iterator end();

T &Top();

void insert(const forward\_iterator &it, const T &value);//вставка

void erase(const forward\_iterator &it); //удаление

void Push(const T &value);

void Pop();

private:

using allocator\_type = typename Allocator::template rebind<Stack\_node>::other;

struct deleter {

deleter(allocator\_type\* allocator): allocator\_(allocator) {}

void operator() (Stack\_node\* ptr) {

if(ptr != nullptr){

std::allocator\_traits<allocator\_type>::destroy(\*allocator\_, ptr);

allocator\_->deallocate(ptr, 1);

}

}

private:

allocator\_type\* allocator\_;

};

using unique\_ptr = std::unique\_ptr<Stack\_node, deleter>;

struct Stack\_node {

T value\_;

unique\_ptr next\_ {nullptr, deleter{&this->allocator\_}};

forward\_iterator next();

Stack\_node() = default;

Stack\_node(const T &value, unique\_ptr next) : value\_(value), next\_(std::move(next)) {};

};

unique\_ptr head\_ {nullptr, deleter{&this->allocator\_}};

allocator\_type allocator\_{}; //конструктор дефолтный для поля

};

template<class T, class Allocator>

typename Stack<T, Allocator>::forward\_iterator Stack<T, Allocator>::Stack\_node::next() {

return { next\_.get()};

}

template<class T, class Allocator>

typename Stack<T, Allocator>::forward\_iterator Stack<T, Allocator>::begin() {

if (head\_ == nullptr) {

return nullptr;

}

return head\_.get();

}

//typename используем, чтобы компилятор понял, что это

template<class T, class Allocator>

typename Stack<T, Allocator>::forward\_iterator Stack<T, Allocator>::end() {

return nullptr;

}

template<class T, class Allocator>

Stack<T, Allocator>::forward\_iterator::forward\_iterator(Stack\_node \*ptr): ptr\_(ptr) {}

template<class T, class Allocator>

T &Stack<T, Allocator>::forward\_iterator::operator\*() {

return ptr\_->value\_;

}

template<class T, class Allocator>

typename Stack<T, Allocator>::forward\_iterator &Stack<T, Allocator>::forward\_iterator::operator++() {

\*this = ptr\_->next();

return \*this;

}

template<class T, class Allocator>

typename Stack<T, Allocator>::forward\_iterator Stack<T, Allocator>::forward\_iterator::operator++(int) {

forward\_iterator old = \*this;

++\*this;

return old;

}

template<class T, class Allocator>

bool Stack<T, Allocator>::forward\_iterator::operator==(const forward\_iterator &o) const {

return ptr\_ == o.ptr\_;

}

template<class T, class Allocator>

bool Stack<T, Allocator>::forward\_iterator::operator!=(const forward\_iterator &o) const {

return ptr\_ != o.ptr\_;

}

template<class T, class Allocator>

void Stack<T, Allocator>::insert(const forward\_iterator &it,const T &value) {

Stack\_node\* newptr = this->allocator\_.allocate(1);

std::allocator\_traits<allocator\_type>::construct(this->allocator\_, newptr, value,

std::unique\_ptr<Stack\_node, deleter>(nullptr, deleter{&this->allocator\_}));

unique\_ptr new\_node(newptr, deleter{&this->allocator\_});

if (it.ptr\_ == nullptr && Size != 0) {

throw std::logic\_error("iterator went beyond the bounds of the stack");

} else if (it.ptr\_ == nullptr && Size == 0) {

new\_node->next\_ = std::move(head\_);

head\_ = std::move(new\_node);

++Size;

} else {

new\_node->next\_ = std::move(it.ptr\_->next\_);

it.ptr\_->next\_ = std::move(new\_node);

++Size;

}

}

template<class T, class Allocator>

void Stack<T, Allocator>::erase(const typename Stack<T, Allocator>::forward\_iterator &it) {

if (it.ptr\_ == nullptr) {

throw std::logic\_error("erasing invalid iterator");

} else if (it == this->begin()) {

head\_ = std::move(it.ptr\_->next\_);

--Size;

} else {

auto it2 = this->begin(); // поиск элемента перед элементом, на который указывает итератор

while (it2.ptr\_->next() != it.ptr\_) {

++it2;

}

it2.ptr\_->next\_ = std::move(it.ptr\_->next\_);

}

}

template<class T, class Allocator>

void Stack<T, Allocator>::Push(const T &value) {

Stack\_node\* newptr = this->allocator\_.allocate(1); // получили ссылку на память, в которую можно положить элемент

std::allocator\_traits<allocator\_type>::construct(this->allocator\_, newptr, value, std::unique\_ptr<Stack\_node, deleter>(

nullptr, deleter{&this->allocator\_})); // к

unique\_ptr new\_node(newptr, deleter{&this->allocator\_});

new\_node->next\_ = std::move(head\_);

head\_ = std::move(new\_node);

++Size;

}

template<class T, class Allocator>

T &Stack<T, Allocator>::Top() {

if (head\_.get()) {

return head\_->value\_;

} else {

throw std::logic\_error("Stack is empty");

}

}

template<class T, class Allocator>

void Stack<T, Allocator>::Pop() {

if (head\_ == nullptr) {

throw std::logic\_error("Stack is empty");

} else {

head\_ = std::move(head\_->next\_);

--Size;

}

}

#endif //UNTITLE06\_STACK\_H

**myallocator.h**

#ifndef UNTITLE06\_MYALLOCATOR\_H

#define UNTITLE06\_MYALLOCATOR\_H

#include <cstdlib>

#include <cstdint>

#include <exception>

#include <iostream>

#include <type\_traits>

#include "mystack.h"

template<class T, size\_t ALLOC\_SIZE>

struct my\_allocator {

using value\_type = T;

using size\_type = std::size\_t;

using difference\_type = std::ptrdiff\_t;

using is\_always\_equal = std::false\_type;

template<class U>

struct rebind {

using other = my\_allocator<U, ALLOC\_SIZE>;

};

my\_allocator():

memory\_pool\_begin\_(new char[ALLOC\_SIZE]),

memory\_pool\_end\_(memory\_pool\_begin\_ + ALLOC\_SIZE),

memory\_pool\_tail\_(memory\_pool\_begin\_)

{}

my\_allocator(const my\_allocator&) = delete;

my\_allocator(my\_allocator&&) = delete;

~my\_allocator() {

delete[] memory\_pool\_begin\_;

}

T\* allocate(std::size\_t n);

void deallocate(T\* ptr, std::size\_t n);

private:

char\* memory\_pool\_begin\_;

char\* memory\_pool\_end\_;

char\* memory\_pool\_tail\_;

Stack<char\*> free\_blocks\_;

};

template<class T, size\_t ALLOC\_SIZE>

T\* my\_allocator<T, ALLOC\_SIZE>::allocate(std::size\_t n){

if(n != 1){

throw std::logic\_error("This allocator can't allocate arrays");

}

if(size\_t(memory\_pool\_end\_ - memory\_pool\_tail\_) < sizeof(T)){

if(free\_blocks\_.Size!=0){

auto it = free\_blocks\_.begin();

char\* ptr = \*it;

free\_blocks\_.Pop();

return reinterpret\_cast<T\*>(ptr);

}

throw std::bad\_alloc();

}

T\* result = reinterpret\_cast<T\*>(memory\_pool\_tail\_);

memory\_pool\_tail\_ += sizeof(T);

return result;

}

template<class T, size\_t ALLOC\_SIZE>

void my\_allocator<T, ALLOC\_SIZE>::deallocate(T\* ptr, std::size\_t n) {

if(n != 1){

throw std::logic\_error("This allocator can't allocate arrays");

}

if(ptr == nullptr){

return;

}

free\_blocks\_.Push(reinterpret\_cast<char\*>(ptr));

}

#endif //UNTITLE06\_MYALLOCATOR\_H

**rectangle.h**

#ifndef UNTITLE06\_RECTANGLE\_H

#define UNTITLE06\_RECTANGLE\_H

#include <string>

#include <type\_traits>

#include <cmath>

#include <stdexcept>

#include <iostream>

template <class T>

std::pair<T, T> operator-(const std::pair<T, T> &p1, const std::pair<T, T> &p2){

return {p1.first-p2.first, p1.second-p2.second};

}

template <class T>

std::ostream& operator<<(std::ostream &o, const std::pair<T, T> &p){

o << "<" << p.first << ", " << p.second << ">";

return o;

}

template <class T>

bool operator==(const std::pair<T, T> &a, const std::pair<T, T> &b){

return (a.first == b.first) && (a.second == b.second);

}

bool isNumber(const std::string& s){

return !s.empty() && s.find\_first\_not\_of("-.0123456789") == std::string::npos; }

template <class T>

std::istream& operator>>(std::istream &is, std::pair<T, T> &p){

std::string checker;

is >> checker;

if(isNumber(checker) == false){

throw std::overflow\_error("Is not a number");

}

p.first = static\_cast<T>(std::stod(checker));

is >> checker;

if(isNumber(checker) == false){

throw std::overflow\_error("Is not a number");

}

p.second = static\_cast<T>(std::stod(checker));

return is;

}

template <class T>

bool collinear(const std::pair<T, T> &a, const std::pair<T, T> &b, const std::pair<T, T> &c, const std::pair<T, T> &d){

return (b.second-a.second)\*(d.first-c.first) - (d.second-c.second)\*(b.first-a.first) <= 1e-9;

}

template <int>

bool collinear(std::pair<int, int> &a, std::pair<int, int> &b, std::pair<int, int> &c, const std::pair<int, int> &d){

return (b.second-a.second)\*(d.first-c.first) == (d.second-c.second)\*(b.first-a.first);

}

template <class T>

bool perpendic(const std::pair<T, T> &a, const std::pair<T, T> &b, const std::pair<T, T> &d){

using vect = std::pair<T, T>;

vect AB = b-a;

vect AD = d-a;

T dotProduct = AB.first\*AD.first + AB.second\*AD.second;

if(dotProduct <= 1e-9 && dotProduct >= -1e-9) return true;

else return false;

}

template <class T>

double dist(const std::pair<T,T> &a, const std::pair<T,T> &b){

return sqrt( ((b.first - a.first) \* (b.first - a.first)) + ((b.second - a.second) \* (b.second - a.second)));

}

template <class T>

class Rectangle{

public:

using type = T;

using point = std::pair<T, T>;

point A, B, C, D;

type area\_;

point center();

void Coord\_print(std::ostream& os);

Rectangle() = default;

Rectangle(point &a, point &b, point &c, point &d){

auto AB = dist(a,b);

auto AD = dist(a,d);

if(a == b || a == c || b == c || a == d ||

!(perpendic(a, b, d)) || !collinear(a, d, c, b) || !collinear(a, b, d, c)){

throw std::overflow\_error("The entered coordinates of the vertices do not belong to the rectangle.");

}else{

A = a;

B = b;

C = c;

D = d;

auto AB = dist(A, B);

auto AD = dist(A, D);

area\_ = AB\*AD;

}

}

type area(){

return area\_;

}

~Rectangle(){};

};

template <class T>

std::pair<T,T> Rectangle<T>::center(){

typename T::type Ox, Oy;

Ox = (A.first + B.first + C.first + D.first)/4;

Oy = (A.second + B.second + C.second + D.second)/4;

return {Ox, Oy};

}

template <class T>

void Rectangle<T>::Coord\_print(std::ostream& os) {

std::cout << "Cords of Rectangle: " << A << ", " << B << ", " << C << ", " << D << std::endl;

}

#endif //UNTITLE06\_RECTANGLE\_H

1. **Выводы:**

Я научился реализовывать аллокатор, с помощью которого можно избежать фрагментации памяти и ускорить работу программы за счет меньшего количества системных вызовов для выделения памяти.

**СПИСОК ЛИТЕРАТУРЫ**

1. Альтернативные аллокаторы памяти [электронный ресурс]. URL: <https://habr.com/ru/post/274827/>
2. Управление памятью в C++ [электронный ресурс]. URL: <https://habr.com/ru/post/148657/>