**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 6**

Тема: Основы работы с коллекциями: аллокаторы

Студент: Почечура Артемий Андреевич

Группа: 80-206

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2021

**1.** **Постановка задачи**

**Цель:**

* Изучение основ работы с контейнерами, знакомство концепцией аллокаторов памяти;

## **Порядок выполнения работы**

1. Ознакомиться с теоретическим материалом.

2. Получить у преподавателя вариант задания.

3. Реализовать задание своего варианта в соответствии с поставленными требованиями.

4. Подготовить тестовые наборы данных.

5. Создать репозиторий на GitHub.

6. Отправить файлы лабораторной работы в репозиторий.

7. Отчитаться по выполненной работе путём демонстрации работающей программы на тестовых наборах данных (как подготовленных самостоятельно, так и предложенных преподавателем) и ответов на вопросы преподавателя (как из числа контрольных, так и по реализации программы).

Вариант 11:

Фигура: прямоугольник;

Структура данных - список;

Аллокатор: - стек.

**2.** **Описание программы**

Реализована структура данных List, которая представляет из себя линейный односторонний список. Аллокатор выделяет память для структуры данных в виде стека. Список может содержать любой тип переменных, в моём случае была выбран прямоугольник (по варианту). Класс прямоугольник Rectangle в данной лабораторной также поддерживает любой тип переменных.

3. **Набор тестов и результаты их исполнения**

Тест 1:

Commands:

add - add element with index.

delete - delete element with index.

print - print list

get - print element with index

end - exit

help

Sides of rectangle (centre of figure - point (0;0)).

Select a function: add

input index: 0

input horisontal side and vertical side: 2 4

Element added

Select a function: add

input index: 1

input horisontal side and vertical side: 4 6

Element added

Select a function: add

input index: 1

input horisontal side and vertical side: 4 8

Element added

Select a function: print

2 4

4 8

4 6

Select a function: end

Тест 2:

Commands:

add - add element with index.

delete - delete element with index.

print - print list

get - print element with index

end - exit

help

Sides of rectangle (centre of figure - point (0;0)).

Select a function: delete

input index: 0

Wrong iterator!

Select a function: add

input index: 1

input horisontal side and vertical side: 2 2

Wrong iterator!

Select a function: add

input index: 0

input horisontal side and vertical side: 2 2

Element added

Select a function: get

input index: 1

Wrong index in List!

Select a function: delete

input index: 1

Wrong iterator!

Select a function: end

Тест 3:

Commands:

add - add element with index.

delete - delete element with index.

print - print list

get - print element with index

end - exit

help

Sides of rectangle (centre of figure - point (0;0)).

Select a function: add

input index: 0

input horisontal side and vertical side: 2 3

Element added

Select a function: add

input index: 1

input horisontal side and vertical side: 3 4

Element added

Select a function: add

input index: 1

input horisontal side and vertical side: 4 5

Element added

Select a function: add

input index: 3

input horisontal side and vertical side: 5 6

Element added

Select a function: print

2 3

4 5

3 4

5 6

Select a function: delete

input index: 2

Element deleted

Select a function: print

2 3

4 5

5 6

Select a function: get

input index: 2

Points: (2.5;3), (2.5;-3), (-2.5;-3), (-2.5;3)

Side 1: 5

Edge: 3.5

Square: 30

Select a function: end

**4.** **Листинг программы**

Файл main.cpp

// Почечура Артемий, гр. М8О-206Б-20

// ЛР №6, вариант 11

// Параметром шаблона должен являться скалярный тип данных задающий тип

данных для оси координат.

// Классы должны иметь публичные поля. Фигура - прямоугольник.

// Для хранения координат фигур необходимо использовать шаблон pair.

// Коллекция - список.

// Аллокатор - стек.

// Создать шаблон динамической коллекцию, согласно варианту задания:

/\* 1. Коллекция должна быть реализована с помощью умных

указателей(shared\_ptr, weak\_ptr).Опционально использование unique\_ptr.

2. В качестве параметра шаблона коллекция должна

принимать тип данных.

3. Коллекция должна содержать метод доступа к элементу по

оператору[].

4. Реализовать аллокатор, который выделяет

фиксированный размер памяти(количество блоков памяти – является параметром

шаблона аллокатора).Внутри аллокатор должен хранить указатель на

используемый блок памяти и динамическую коллекцию указателей на свободные

блоки.

5. Коллекция должна использовать аллокатор для выделения и

освобождения памяти для своих элементов.

6. Аллокатор должен быть совместим с контейнерами map и list.

7. Реализовать программу, которая:

o позволяет вводить с клавиатуры фигуры(с типом int в

качестве параметра шаблона фигуры) и добавлять в коллекцию использующую

аллокатор;

o позволяет удалять элемент из коллекции по номеру элемента;

o выводит на экран введенные фигуры c помощью for\_each. \*/

#include <iostream>

#include <vector>

#include <cmath>

#include <string>

#include <iterator>

#include <algorithm>

#include <memory>

#include <exception>

#include <stdexcept>

#include <stack>

#include "Rectangle.h"

#include "Rectangle.cpp"

using namespace std;

template<class F, class T>

class forward\_iterator : public iterator<input\_iterator\_tag, F> {

public:

F\* ptr;

forward\_iterator(F\* pt) :ptr(pt) {}

T& operator\*() {

return ptr->value;

}

T\* operator->() const {

return &ptr->value;

}

forward\_iterator& operator++() {

if (ptr == nullptr) {

throw runtime\_error("Trying to increment null pointer");

}

if (ptr->next) {

ptr = ptr->next.get();

}

else {

ptr = nullptr;

}

return \*this;

}

forward\_iterator operator++(int) {

auto res = \*this;

operator++();

return res;

}

bool operator==(const forward\_iterator& lval) const {

return ptr == lval.ptr;

}

bool operator!=(const forward\_iterator& lval) const {

return ptr != lval.ptr;

}

operator bool() {

return ptr;

}

};

template <class T>

class my\_allocator : public allocator<T> {

public:

stack<T\*> memory;

T\* allocate(size\_t n) {

for (int i = 0; i < n; i++) {

memory.push\_back(new T);

}

return \*prev(memory.end());

}

void deallocate(T\* pointer, size\_t) {

auto it = find\_if(memory.begin(), memory.end(),

[pointer](const auto ptr) {return ptr == pointer; });

if (it != memory.end()) {

delete (\*it);

memory.erase(it);

}

}

void destroy(T\* p) {

p->~T();

}

};

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

class List {

public:

static Allocator get\_allocator() {

static Allocator allocator;

return allocator;

}

struct list\_item {

T value = {};

unique\_ptr<list\_item> next = nullptr;

list\_item() = default;

list\_item(T val) : value(move(val)) {}

void\* operator new(size\_t size) {

return get\_allocator().allocate(sizeof(T));

}

void operator delete(void\* p) {

// get\_allocator().destroy((T\*)p);

get\_allocator().deallocate((T\*)p, sizeof(T));

}

};

size\_t size\_;

unique\_ptr<list\_item> head;

using iterator = forward\_iterator<list\_item, T>;

List() {

size\_ = 0;

head = nullptr;

}

List(size\_t size) {

size\_ = size;

if (size == 0) {

return;

}

head = make\_unique<list\_item>();

list\_item\* node = head.get();

while (size-- > 1) {

node->next = make\_unique<list\_item>();

node = node->next.get();

}

}

T& operator[](size\_t index) {

if (index >= size() || index < 0) {

throw out\_of\_range("Index is out of range");

}

list\_item\* node = head.get();

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

node = node->next.get();

}

return node->value;

}

size\_t size() {

return size\_;

}

iterator begin() {

return iterator(head.get());

}

iterator end() {

return iterator{ nullptr };

}

iterator insert(iterator it, T val) {

if (it == begin()) {

auto ptr = make\_unique<list\_item>(val);

ptr->next = move(head);

head = move(ptr);

++size\_;

return begin();

}

iterator prev = find\_prev(it);

auto ptr = make\_unique<list\_item>(val);

ptr->next = move(prev.ptr->next);

prev.ptr->next = move(ptr);

++size\_;

return ++prev;

}

iterator erase(iterator it) {

if (size\_ == 0) {

throw runtime\_error("List empty before erase");

}

if (it == begin()) {

head = move(head->next);

size\_--;

return begin();

}

iterator prev = find\_prev(it);

if (it.ptr) {

prev.ptr->next = move(it.ptr->next);

}

else {

throw runtime\_error("Trying to remove element using invalid iterator");

}

iterator res{ prev.ptr };

size\_--;

++res;

return res;

}

T& back() {

if (size\_ == 0) {

throw runtime\_error("List is empty");

}

auto tmp = find\_prev(end());

return \*tmp;

}

T& front() {

if (size\_ == 0) {

throw runtime\_error("List is empty");

}

return \*begin();

}

void pop\_front() {

if (size\_ == 0) {

throw runtime\_error("List empty before pop");

}

erase(begin());

}

void pop\_back() {

if (size\_ == 0) {

throw runtime\_error("List empty before pop");

}

auto tmp = find\_prev(end());

erase(tmp);

}

void push\_front(T value) {

insert(begin(), move(value));

}

void push\_back(T value) {

insert(end(), move(value));

}

iterator find\_prev(iterator cur) {

if (cur == begin()) {

throw runtime\_error("There is no element before the first");

}

iterator prev = begin();

for (iterator it = ++begin(); it != end(); ++it) {

if (it == cur) {

return prev;

}

prev = it;

}

if (cur == end()) {

return prev;

}

throw invalid\_argument("It's not iterator from this List");

}

};

template <class T, class Alloc>

void print(List<T, Alloc>& st) {

for\_each(st.begin(), st.end(), [](auto& el) {cout << el << "\n"; });

}

bool readnum(int& a){

cin >> a;

if (cin.fail()) {

cout <<"Incorrect input"<<'\n';

cin.clear();

cin.ignore(numeric\_limits<int>::max(),'\n');

return false;

}

return true;

}

void help(){

cout << "Commands:\n";

cout << "\tadd - add element with index.\n\tdelete - delete element

with index.\n";

cout << "\tprint - print list";

cout << "\n\tget - print element with index";

cout << "\n\tend - exit\n";

cout << "\thelp\n\n";

cout << "Sides of rectangle (centre of figure - point (0;0)).\n\n";

}

int main(int argc, char\*\* argv) {

List<Rectangle<double>> list;

string act;

help();

while (1){

try{

cout << "Select a function: ";

cin>>act;

if (act=="end"){

break;

}

else if (act == "add"){

int ind;

cout << "input index: ";

if (!readnum(ind)){

continue;

}

cout << "input horisontal side and vertical side: ";

Rectangle<double> a;

if(!(cin>>a)){

cout << "Wrong values!";

continue;

}

auto it = list.begin();

for (int i = 0; i < ind; i++) {

++it;

}

list.insert(it, a);

cout << "Element added\n";

}

else if (act == "delete"){

int ind;

cout << "input index: ";

if (!readnum(ind)){

continue;

}

auto it = list.begin();

for (int i = 0; i < ind; i++) {

++it;

}

list.erase(it);

cout << "Element deleted\n";

}

else if (act == "print"){

print(list);

}

else if (act == "get"){

int ind;

cout << "input index: ";

if (!readnum(ind)){

continue;

}

list[ind].print\_info();

}

else if (act == "help"){

help();

}

else {

cout << "Invalid function. Enter 'help' for help."<<'\n';

}

}

catch (invalid\_argument& exp) {

cout << exp.what() << "\n";

}

catch (runtime\_error& exp) {

cout << exp.what() << "\n";

}

catch (out\_of\_range& exp) {

cout << exp.what() << "\n";

}

}

}

Файл Rectangle.h

#ifndef RECTANGLE\_H

#define RECTANGLE\_H

#include <utility>

using namespace std;

template<typename T>

class Rectangle {

public:

Rectangle(T i,T j);

Rectangle();

Rectangle(const Rectangle<T>& orig);

double square();

void print\_points();

void print\_info();

void print\_sides();

template<typename V> friend std::ostream& operator<<(std::ostream&

os, Rectangle<V>& obj);

template<typename V> friend std::istream& operator>>(std::istream&

is, Rectangle<V>& obj);

bool operator==(Rectangle<T>& copy);

Rectangle<T>& operator=(const Rectangle<T>& copy);

virtual ~Rectangle();

private:

T one;

T two;

};

#endif

Файл Rectangle.cpp

#ifndef RECTANGLE\_H

#define RECTANGLE\_H

#include <utility>

using namespace std;

template<typename T>

class Rectangle {

public:

Rectangle(T i,T j);

Rectangle();

Rectangle(const Rectangle<T>& orig);

double square();

void print\_points();

void print\_info();

void print\_sides();

template<typename V> friend std::ostream& operator<<(std::ostream&

os, Rectangle<V>& obj);

template<typename V> friend std::istream& operator>>(std::istream&

is, Rectangle<V>& obj);

bool operator==(Rectangle<T>& copy);

Rectangle<T>& operator=(const Rectangle<T>& copy);

virtual ~Rectangle();

private:

T one;

T two;

};

#endif

**5.** **Выводы**

В ходе лабораторной работы я реализовал шаблон класса контейнера для хранения объектов шаблона класса класса Rectangle, представляющих собой прямоугольники, координаты которых задаются параметром шаблона. Реализованная программа упростит работу, связанную с хранением и обработкой большого количества вышеописанных объектов. Шаблон класса контейнера реализован таким образом, что данный контейнер позволяет хранить объекты абсолютно любых типов — методы никак не привязаны к конкретным объектам и работают исключительно с указателями. Реализованный класс аллокатора позволит улучшить эффективность работы программы благодаря лучшему контролю выделяемой памяти. Помимо этого, был реализован файл атрибутивной грамматики, описывающий язык общения пользователя с программой — пользователю нужно знать лишь грамматику и синтаксис этого языка, что упрощает его опыт по использованию программы.

ЛИТЕРАТУРА

1. Уроки программирования на языке С++ [Электронный ресурс]. URL:<https://ravesli.com/uroki-cpp> (дата обращения 29.11.2021).

2. The Compiler Generator Coco/R for C++. Documentation [Электронный ресурс]. URL:<https://ssw.jku.at/Research/Projects/Coco/#Docu> (дата обращения 29.11.2021).

3. cpp.reference, справочник по спецификации языка C++ [Электронный ресурс]. URL:<https://en.cppreference.com/w/> (дата обращения 02.12.2021).