Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторные работы

по курсу «Объектно-ориентированное программирование»

III семестр

|  |  |
| --- | --- |
| Студент: | Лисовский О.Р. |
| Группа: | 8О-208Б, №11 |
| Преподаватель: | Поповкин А.В. |
|  |  |
| Оценка: |  |
| Дата: |  |

Москва 2018

Содержание

[Лабораторная работа №1 3](#_Toc533283844)

[Цель работы: 4](#_Toc533283845)

[Задание: 4](#_Toc533283846)

[Код программы: 4](#_Toc533283847)

[Тесты: 8](#_Toc533283848)

[Лабораторная работа №2 9](#_Toc533283849)

[Цель работы: 10](#_Toc533283850)

[Задание: 10](#_Toc533283851)

[Код программы: 10](#_Toc533283852)

[Тесты: 16](#_Toc533283853)

[Лабораторная работа №3 17](#_Toc533283854)

[Цель работы: 18](#_Toc533283855)

[Задание: 18](#_Toc533283856)

[Код программы: 18](#_Toc533283857)

[Тесты: 26](#_Toc533283858)

[Лабораторная работа №4 27](#_Toc533283859)

[Цель работы: 28](#_Toc533283860)

[Задание: 28](#_Toc533283861)

[Код программы: 28](#_Toc533283862)

[Тесты: 30](#_Toc533283863)

[Лабораторная работа №5 31](#_Toc533283864)

[Цель работы: 32](#_Toc533283865)

[Задание: 32](#_Toc533283866)

[Код программы: 32](#_Toc533283867)

[Тесты: 37](#_Toc533283868)

[Лабораторная работа №6 38](#_Toc533283869)

[Цель работы: 39](#_Toc533283870)

[Задание: 39](#_Toc533283871)

[Код программы: 39](#_Toc533283872)

[Тесты: 44](#_Toc533283873)

[Лабораторная работа №7 45](#_Toc533283874)

[Цель работы: 46](#_Toc533283875)

[Задание: 46](#_Toc533283876)

[Код программы: 46](#_Toc533283877)

[Тесты: 50](#_Toc533283878)

[Лабораторная работа №8 51](#_Toc533283879)

[Цель работы: 52](#_Toc533283880)

[Задание: 52](#_Toc533283881)

[Код программы: 52](#_Toc533283882)

[Тесты: 56](#_Toc533283883)

[Заключение 57](#_Toc533283884)

Московский Авиационный Институт

(Национальный Исследовательский Университет)

Кафедра: 806 «Вычислительная математика и программирование»

Факультет: «Прикладная математика и информатика»

Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа №1.

по дисциплине «Объектно-ориентированное программирование»

|  |  |
| --- | --- |
| Студент: | Лисовский О.Р. |
| Группа: | 8О-208Б, №11 |
| Преподаватель: | Поповкин А.В. |
| Вариант | №15 |
| Оценка: |  |
| Дата: |  |

Москва 2018

## Цель работы:

Программирование классов на языке С++; управление памятью в языке С++; изучение базовых понятий ООП; знакомство с классами в C++; знакомство с перегрузкой операторов; знакомство с дружественными функциями; знакомство с операциями ввода-вывода из стандартных библиотек.

## Задание:

Необходимо спроектировать и запрограммировать на языке C++ классы фигур, согласно [вариантов задания](#_Варианты_задания_(фигуры)).

Классы должны удовлетворять следующим правилам:

Должны иметь общий родительский класс Figure; должны иметь общий виртуальный метод Print, печатающий параметры фигуры и ее тип в стандартный поток вывода cout; должный иметь общий виртуальный метод расчета площади фигуры – Square; должны иметь конструктор, считывающий значения основных параметров фигуры из стандартного потока cin; должны быть расположены в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp); программа должна позволять вводить фигуру каждого типа с клавиатуры, выводить параметры фигур на экран и их площадь.

## Код программы:

**figure.h:**

#ifndef FIGURE\_H

#define FIGURE\_H

class Figure

{

public:

virtual double Square() = 0;

virtual void Print() = 0;

virtual ~Figure() {};

};

#endif

**hexagon.h:**

#ifndef HEXAGON\_H

#define HEXAGON\_H

#include <cstdlib>

#include <iostream>

#include "figure.h"

class Hexagon : public Figure

{

public:

Hexagon();

Hexagon(std::istream &is);

Hexagon(size\_t i);

Hexagon(const Hexagon& orig);

double Square() override;

void Print() override;

virtual ~Hexagon();

private:

size\_t side;

};

#endif

**hexagon.cpp:**

#include <iostream>

#include <cmath>

#include "hexagon.h"

Hexagon::Hexagon() : Hexagon(0){

}

Hexagon::Hexagon(size\_t i) : side(i)

{

std::cout << "Hexagon created: " << side << std::endl;

}

Hexagon::Hexagon(std::istream &is)

{

is >> side;

}

Hexagon::Hexagon(const Hexagon& orig)

{

std::cout << "Hexagon copy created" << std::endl;

side = orig.side;

}

double Hexagon::Square()

{

return 3 \* sqrt(3) / 2 \* pow(side, 2);

}

void Hexagon::Print()

{

std::cout << "Hexagon's side = " << side << std::endl;

}

Hexagon::~Hexagon()

{

std::cout << "Hexagon deleted" << std::endl;

}

**octagon.h:**

#ifndef OCTAGON\_H

#define OCTAGON\_H

#include <cstdlib>

#include <iostream>

#include "figure.h"

class Octagon : public Figure

{

public:

Octagon();

Octagon(std::istream &is);

Octagon(size\_t i);

Octagon(const Octagon& orig);

double Square() override;

void Print() override;

virtual ~Octagon();

private:

size\_t side;

};

#endif

**octagon.cpp:**

#include <iostream>

#include <cmath>

#include "octagon.h"

Octagon::Octagon() : Octagon(0){

}

Octagon::Octagon(size\_t i) : side(i)

{

std::cout << "Octagon created: " << side << std::endl;

}

Octagon::Octagon(std::istream &is)

{

is >> side;

}

Octagon::Octagon(const Octagon& orig)

{

std::cout << "Octagon copy created" << std::endl;

side = orig.side;

}

double Octagon::Square()

{

return 2 \* (1 + sqrt(2)) \* pow(side, 2);

}

void Octagon::Print()

{

std::cout << "Octagon's side = " << side << std::endl;

}

Octagon::~Octagon()

{

std::cout << "Octagon deleted" << std::endl;

}

**triangle.h:**

#ifndef TRIANGLE\_H

#define TRIANGLE\_H

#include <cstdlib>

#include <iostream>

#include "figure.h"

class Triangle : public Figure

{

public:

Triangle();

Triangle(std::istream &is);

Triangle(size\_t i);

Triangle(const Triangle& orig);

double Square() override;

void Print() override;

virtual ~Triangle();

private:

size\_t side;

};

#endif

**triangle.cpp:**

#include <iostream>

#include <cmath>

#include "triangle.h"

Triangle::Triangle() : Triangle(0){

}

Triangle::Triangle(size\_t i) : side(i)

{

std::cout << "Triangle created: " << side << std::endl;

}

Triangle::Triangle(std::istream &is)

{

is >> side;

}

Triangle::Triangle(const Triangle& orig)

{

std::cout << "Triangle copy created" << std::endl;

side = orig.side;

}

double Triangle::Square()

{

return sqrt(3) / 4 \* pow(side, 2);

}

void Triangle::Print()

{

std::cout << "side = " << side << std::endl;

}

Triangle::~Triangle()

{

std::cout << "Triangle deleted" << std::endl;

}

**main.cpp:**

#include <cstdlib>

#include "hexagon.h"

#include "octagon.h"

#include "triangle.h"

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

{

std::cout << "Enter Triangle's side" << std::endl;

Figure \*ptr1 = new Triangle(std::cin);

ptr1->Print();

std::cout << ptr1->Square() << std::endl;

delete ptr1;

std::cout << "Enter Octagon's side" << std::endl;

Figure \*ptr2 = new Octagon(std::cin);

ptr2->Print();

std::cout << ptr2->Square() << std::endl;

delete ptr2;

std::cout << "Enter Hexagon's side" << std::endl;

Figure \*ptr3 = new Hexagon(std::cin);

ptr3->Print();

std::cout << ptr3->Square() << std::endl;

delete ptr3;

return 0;

}

**Makefile:**

clean: main.exe

rm \*.o

main.exe: main.o octagon.o hexagon.o triangle.o

g++ -Wall main.o octagon.o hexagon.o triangle.o -o main.exe

main.o: main.cpp octagon.h hexagon.h triangle.h

g++ -c -Wall main.cpp

octagon.o: octagon.cpp octagon.h

g++ -c -Wall octagon.cpp

triangle.o: triangle.cpp triangle.h

g++ -c -Wall triangle.cpp

hexagon.o: hexagon.cpp hexagon.h

g++ -c -Wall hexagon.cpp

## Тесты:

|  |  |
| --- | --- |
| Входные данные | Ожидаемый результат |
| 2 3 4 | 2 1.73205 3 43.4558 4 41.5692 |
| 6 1 12 | 6 15.5885 1 4.82843 12 374.123 |
| 32 120 11 | 32 443.405 120 69529.4 11 314.367 |

Московский Авиационный Институт

(Национальный Исследовательский Университет)

Кафедра: 806 «Вычислительная математика и программирование»

Факультет: «Прикладная математика и информатика»

Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа №2.

по дисциплине «Объектно-ориентированное программирование»

|  |  |
| --- | --- |
| Студент: | Лисовский О.Р. |
| Группа: | 8О-208Б, №11 |
| Преподаватель: | Поповкин А.В. |
| Вариант | №15 |
| Оценка: |  |
| Дата: |  |

Москва 2018

## Цель работы:

Закрепление навыков работы с классами; создание простых динамических структур данных; работа с объектами, передаваемыми «по значению».

## Задание:

Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер [первого уровня](#_Варианты_задания_(структуры), содержащий одну фигуру (колонка фигура 1), согласно [вариантов задания](#_Варианты_задания_(фигуры)) (реализованную в [ЛР1](#_Лабораторная_работа_№1)).

Классы должны удовлетворять следующим правилам:

Требования к классу фигуры аналогичны требованиям из [лабораторной работы 1](#_Задание); классы фигур должны иметь переопределенный оператор вывода в поток std::ostream (<<). Оператор должен распечатывать параметры фигуры (тип фигуры, длины сторон, радиус и т.д.); классы фигур должны иметь переопределенный оператор ввода фигуры из потока std::istream (>>). Оператор должен вводить основные параметры фигуры (длины сторон, радиус и т.д); классы фигур должны иметь операторы копирования (=); классы фигур должны иметь операторы сравнения с такими же фигурами (==); класс-контейнер должен содержать объекты фигур “по значению” (не по ссылке); класс-контейнер должен иметь метод по добавлению фигуры в контейнер; класс-контейнер должен иметь методы по получению фигуры из контейнера (определяется структурой контейнера); класс-контейнер должен иметь метод по удалению фигуры из контейнера (определяется структурой контейнера); класс-контейнер должен иметь перегруженный оператор по выводу контейнера в поток std::ostream (<<); класс-контейнер должен иметь деструктор, удаляющий все элементы контейнера; классы должны быть расположены в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

Нельзя использовать:

Стандартные контейнеры std; шаблоны (template); различные варианты умных указателей (shared\_ptr, weak\_ptr).

Программа должна позволять:

Вводить произвольное количество фигур и добавлять их в контейнер; распечатывать содержимое контейнера; удалять фигуры из контейнера.

## Код программы:

**hexagon.h:**

Класс Hexagon более не является наследником класса Figure

Добавлены прототипы:

Hexagon& operator++();

friend Hexagon operator+(const Hexagon& left, const Hexagon& right);

friend std::ostream& operator<<(std::ostream& os, const Hexagon& obj);

friend std::istream& operator>>(std::istream& is, Hexagon& obj);

Hexagon& operator=(const Hexagon& right);

Убраны прототипы:

Hexagon(size\_t i);

void Print() override;

**hexagon.cpp:**

Добавлены методы:

Hexagon& Hexagon::operator=(const Hexagon& right)

{

if (this == &right)

return \*this;

std::cout << "Hexagon copied" << std::endl;

side = right.side;

return \*this;

}

Hexagon& Hexagon::operator++()

{

side++;

return \*this;

}

Hexagon operator+(const Hexagon& left, const Hexagon& right)

{

return Hexagon(left.side + right.side);

}

std::ostream& operator<<(std::ostream& os, const Hexagon& obj)

{

os << "side = " << obj.side << std::endl;

return os;

}

std::istream& operator>>(std::istream& is, Hexagon& obj)

{

is >> obj.side;

return is;

}

Убраны методы:

void Hexagon::Print()

Hexagon::Hexagon(std::istream &is)

**tree.h:**

#ifndef TTREE\_H

#define TTREE\_H

#include "hexagon.h"

#include "treeitem.h"

class TTree

{

public:

TTree();

TTree(const TTree& orig);

void push(Hexagon hex, int k);

bool empty();

Hexagon pop();

friend std::ostream& operator<<(std::ostream& os, const TTree& tree);

virtual ~TTree();

private:

TTreeItem \*seed;

};

#endif

**tree.cpp:**

#include "tree.h"

TTree::TTree() : seed(nullptr){

}

TTree::TTree(const TTree& orig){

seed = orig.seed;

}

std::ostream& operator<<(std::ostream& os, const TTree& tree)

{

TTreeItem \*item = tree.seed;

if(item == nullptr)

{

return os;

}

item->Show();

return os;

}

void TTree::push(Hexagon hexagon, int k)

{

seed = seed->SetNext(hexagon, k);

}

bool TTree::empty()

{

return seed == nullptr;

}

Hexagon TTree::pop()

{

Hexagon result;

TTreeItem \*oldSeed;

if(seed != nullptr)

{

if(seed->CheckLeft() == true)

{

oldSeed = seed->GetLeft(seed);

result = oldSeed->GetHexagone();

}

else

{

oldSeed = seed;

result = oldSeed->GetHexagone();

}

}

seed = seed->DelItem(result);

return result;

}

TTree::~TTree()

{

delete seed;

}

**treeitem.h:**

#ifndef TTREEITEM\_H

#define TTREEITEM\_H

#include "hexagon.h"

class TTreeItem

{

public:

TTreeItem(const Hexagon& hexagon);

TTreeItem(const TTreeItem& orig);

friend std::ostream& operator<<(std::ostream& os, const TTreeItem& obj);

bool CheckLeft();

void Show();

TTreeItem\* DelItem(Hexagon res);

TTreeItem\* SetNext(Hexagon hex, int k);

TTreeItem\* GetLeft(TTreeItem\* seed);

TTreeItem\* GetRight();

Hexagon GetHexagone() const;

virtual ~TTreeItem();

private:

Hexagon hexagon;

TTreeItem \*left;

TTreeItem \*right;

};

#endif

**treeitem.cpp:**

#include "treeitem.h"

#include <iostream>

TTreeItem::TTreeItem(const Hexagon& hexagon)

{

this->hexagon = hexagon;

this->left = nullptr;

this->right = nullptr;

std::cout << "Tree item: created" << std::endl;

}

TTreeItem::TTreeItem(const TTreeItem& orig)

{

this->hexagon = orig.hexagon;

this->left = orig.left;

this->right = orig.right;

std::cout << "Tree item: copied" << std::endl;

}

bool TTreeItem::CheckLeft()

{

if (this->left == nullptr)

return false;

else

return true;

}

void TTreeItem::Show()

{

if(this->left != nullptr)

{

this->left->Show();

}

std::cout << "Tree item = " << \*this;

if(this->right != nullptr)

{

this->right->Show();

}

return;

}

TTreeItem\* TTreeItem::DelItem(Hexagon res)

{

if(res.Square() < this->hexagon.Square())

{

this->left = this->left->DelItem(res);

return this;

}

else if(res.Square() == this->hexagon.Square())

{

TTreeItem\* saveRight = this->GetRight();

return saveRight;

}

return this;

}

TTreeItem\* TTreeItem::SetNext(Hexagon hex, int k)

{

if(k == 1)

{

TTreeItem \*newTreeItem = new TTreeItem(hex);

return newTreeItem;

}

else if(this->hexagon.Square() > hex.Square())

{

if(this->left == nullptr)

k = 1;

this->left = this->left->SetNext(hex, k);

}

else if(this->hexagon.Square() < hex.Square())

{

if(this->right == nullptr)

k = 1;

this->right = this->right->SetNext(hex, k);

}

return this;

}

Hexagon TTreeItem::GetHexagone() const

{

return this->hexagon;

}

TTreeItem\* TTreeItem::GetLeft(TTreeItem\* seed)

{

TTreeItem \*oldSeed;

if(seed->left != nullptr)

{

oldSeed = seed->GetLeft(seed->left);

}

else

{

oldSeed = seed;

}

return oldSeed;

}

TTreeItem\* TTreeItem::GetRight()

{

return this->right;

}

TTreeItem::~TTreeItem()

{

std::cout << "Tree item: deleted" << std::endl;

delete left;

delete right;

}

std::ostream& operator<<(std::ostream& os, const TTreeItem& obj)

{

os << "Tree's hexagon: "<< std::endl << obj.hexagon;

return os;

}

**main.cpp:**

#include <cstdlib>

#include <iostream>

#include "hexagon.h"

#include "tree.h"

#include "treeitem.h"

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

{

int c, b, k;

Hexagon u,h;

std::cout << "How many hexagons you want to create: ";

std::cin >> c;

b = c;

TTree tree;

std::cout << "Enter hexagon's side: ";

std::cin >> u;

k = 1;

while(c > 0)

{

tree.push(u, k);

c--;

if(c == 0)

break;

k = 0;

std::cout << "Enter hexagon's side: ";

std::cin >> u;

}

std::cout << std::endl << std::endl << tree;

std::cout << std::endl << std::endl;

while(b > 0)

{

h = tree.pop();

std::cout << h;

b--;

}

return 0;

}

**Makefile:**

clean: main.exe

rm \*.o

main.exe: main.o tree.o hexagon.o treeitem.o

g++ -Wall -g main.o tree.o hexagon.o treeitem.o -o main.exe

main.o: main.cpp treeitem.h hexagon.h tree.h

g++ -c -g -Wall main.cpp

tree.o: tree.cpp tree.h

g++ -c -g -Wall tree.cpp

treeitem.o: treeitem.cpp treeitem.h

g++ -c -g -Wall treeitem.cpp

hexagon.o: hexagon.cpp hexagon.h

g++ -c -g -Wall hexagon.cpp

## Тесты:

|  |  |
| --- | --- |
| Входные данные | Ожидаемый результат |
| 4 5 2 7 78 | 2 5 7 78 |
| 6 4 45 21 90 89 1 | 1 4 21 45 89 90 |
| 3 1 9000 2 | 1 2 9000 |

Московский Авиационный Институт

(Национальный Исследовательский Университет)

Кафедра: 806 «Вычислительная математика и программирование»

Факультет: «Прикладная математика и информатика»

Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа №3.

по дисциплине «Объектно-ориентированное программирование»

|  |  |
| --- | --- |
| Студент: | Лисовский О.Р. |
| Группа: | 8О-208Б, №11 |
| Преподаватель: | Поповкин А.В. |
| Вариант | №15 |
| Оценка: |  |
| Дата: |  |

Москва 2018

## Цель работы:

Закрепление навыков работы с классами; знакомство с умными указателями.

## Задание:

Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер [первого уровня](#_Варианты_задания_(структуры), содержащий все три фигуры класса фигуры, согласно [вариантов задания](#_Варианты_задания_(фигуры)) (реализованную в [ЛР1](#_Лабораторная_работа_№1)).

Классы должны удовлетворять следующим правилам:

Требования к классу фигуры аналогичны требованиям из [лабораторной работы 1](#_Задание); класс-контейнер должен содержать объекты используя std:shared\_ptr<…>; класс-контейнер должен иметь метод по добавлению фигуры в контейнер; класс-контейнер должен иметь методы по получению фигуры из контейнера (определяется структурой контейнера); класс-контейнер должен иметь метод по удалению фигуры из контейнера (определяется структурой контейнера); класс-контейнер должен иметь перегруженный оператор по выводу контейнера в поток std::ostream (<<); класс-контейнер должен иметь деструктор, удаляющий все элементы контейнера; классы должны быть расположены в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

Нельзя использовать:

Стандартные контейнеры std; шаблоны (template); объекты «по значению»

Программа должна позволять:

Вводить произвольное количество фигур и добавлять их в контейнер; распечатывать содержимое контейнера; удалять фигуры из контейнера.

## Код программы:

Файлы, не претерпевшие изменений: figure.h

**hexagon.h:**

Класс Hexagon теперь является наследником класса Figure

Добавлен прототип:

void Print();

Убраны прототипы:

Hexagon& operator++();

friend Hexagon operator+(const Hexagon& left, const Hexagon& right);

friend std::istream& operator>>(std::istream& is, Hexagon& obj);

**hexagon.cpp:**

Добавлен метод:

void Hexagon::Print()

{

std::cout << "Hexagone's side = " << side << std::endl;

}

Изменён метод:

Hexagon::Hexagon() : Hexagon(0)

{

std::cout << "Hexagon created: default" << std::endl;

}

Убраны методы:

Hexagon& Hexagon::operator++()

Hexagon operator+(const Hexagon& left, const Hexagon& right)

std::istream& operator>>(std::istream& is, Hexagon& obj)

**octagon.h:**

Добавлены прототипы:

friend std::ostream& operator<<(std::ostream& os, const Octagon& obj);

Octagon& operator=(const Octagon& right);

Убран прототип:

Octagon(std::istream &is);

**octagon.cpp:**

Добавлены методы:

Octagon& Octagon::operator=(const Octagon& right)

{

if(this == &right)

return \*this;

std::cout << "Octagon copied" << std::endl;

side = right.side;

return \*this;

}

std::ostream& operator<<(std::ostream& os, const Octagon& obj)

{

os << "side = " << obj.side;

return os;

}

Изменён метод:

Octagon::Octagon() : Octagon(0)

{

std::cout << "Octagon created: default" << std::endl;

}

Убран метод:

Octagon::Octagon(std::istream &is)

**triangle.h:**

Добавлены прототипы:

friend std::ostream& operator<<(std::ostream& os, const Triangle& obj);

Triangle& operator=(const Triangle& right);

Убран прототип:

Triangle(std::istream &is);

**triangle.cpp:**

Добавлены методы:

Triangle& Triangle::operator=(const Triangle& right)

{

if(this == &right)

return \*this;

std::cout << "Triangle copied" << std::endl;

side = right.side;

return \*this;

}

std::ostream& operator<<(std::ostream& os, const Triangle& obj)

{

os << "side = " << obj.side;

return os;

}

Изменён метод:

Triangle::Triangle() : Triangle(0)

{

std::cout << "Triangle created: default" << std::endl;

}

Убран метод:

Triangle::Triangle(std::istream &is)

**tree.h:**

Добавлены строки:

#include "triangle.h"

#include "octagon.h"

#include <memory>

Изменены прототипы:

void push(std::shared\_ptr<Figure> figure, int k);

std::shared\_ptr<Figure> pop();

Изменён тип private переменной seed:

std::shared\_ptr<TTreeItem> seed;

Убран прототип:

TTree(const TTree& orig);

**tree.cpp:**

Изменён метод:

std::ostream& operator<<(std::ostream& os, const TTree& tree)

{

std::shared\_ptr<TTreeItem> item = tree.seed;

if(item == nullptr)

{

return os;

}

item->Show();

return os;

}

Убран метод:

TTree::TTree(const TTree& orig)

Методы принимавшие или возвращавшие переменную класса Hexagon теперь принимают или возвращают переменную типа std::shared\_ptr<Figure>:

std::shared\_ptr<Figure> TTree::pop()

void TTree::push(std::shared\_ptr<Figure> figure, int k)

**treeitem.h:**

Добавлены строки:

#include "triangle.h"

#include "octagon.h"

#include <memory>

Добавлены прототипы:

TTreeItem(std::shared\_ptr<Figure> figure);

std::shared\_ptr<Figure> GetPtrFigure();

std::shared\_ptr<Figure> GetFigure() const;

Изменены частные переменные класса TTreeItem:

std::shared\_ptr<Figure> figure;

std::shared\_ptr<TTreeItem> left;

std::shared\_ptr<TTreeItem> right;

Изменены прототипы:

TTreeItem();

friend std::ostream& operator<<(std::ostream& os, const TTreeItem obj);

std::shared\_ptr<TTreeItem> Set(std::shared\_ptr<TTreeItem> seed, std::shared\_ptr<Figure> figure, int k);

std::shared\_ptr<TTreeItem> DelItem(std::shared\_ptr<TTreeItem> oldSeed, std::shared\_ptr<TTreeItem> seed);

std::shared\_ptr<TTreeItem> GetLeft(std::shared\_ptr<TTreeItem> seed);

Убран прототип:

TTreeItem(const Hexagon& hexagon);

**treeitem.cpp:**

Добавлены методы:

TTreeItem::TTreeItem(std::shared\_ptr<Figure> figure)

{

this->figure = figure;

this->left = nullptr;

this->right = nullptr;

std::cout << "Tree item: created" << std::endl;

}

std::shared\_ptr<Figure> TTreeItem::GetPtrFigure()

{

return figure;

}

std::shared\_ptr<Figure> TTreeItem::GetFigure() const

{

return this->figure;

}

Изменены методы:

TTreeItem::TTreeItem()

{

this->figure = nullptr;

this->left = nullptr;

this->right = nullptr;

std::cout << "Tree item: created" << std::endl;

}

void TTreeItem::Show()

{

if(this->left != nullptr)

{

this->left->Show();

}

std::cout << "Tree item = ";

this->figure->Print();

if(this->right != nullptr)

{

this->right->Show();

}

return;

}

std::shared\_ptr<TTreeItem> TTreeItem::Set(std::shared\_ptr<TTreeItem> seed, std::shared\_ptr<Figure> figure, int k)

{

if(k == 1)

{

std::shared\_ptr<TTreeItem> newTreeItem = std::make\_shared<TTreeItem>(\*(new TTreeItem(figure)));;

return newTreeItem;

}

else if(seed->figure->Square() > figure->Square())

{

if(seed->left == nullptr)

k = 1;

seed->left = seed->left->Set(seed->left, figure, k);

}

else if(seed->figure->Square() <= figure->Square())

{

if(seed->right == nullptr)

k = 1;

seed->right = seed->right->Set(seed->right, figure, k);

}

return seed;

}

std::shared\_ptr<TTreeItem> TTreeItem::DelItem(std::shared\_ptr<TTreeItem> oldSeed, std::shared\_ptr<TTreeItem> seed)

{

if (seed != oldSeed)

{

seed->left = seed->DelItem(oldSeed, seed->left);

}

else

{

if(seed->right != nullptr)

{

seed = seed->right;

}

else

seed = nullptr;

}

return seed;

}

std::ostream& operator<<(std::ostream& os, TTreeItem& obj)

{

obj.GetPtrFigure()->Print();

return os;

}

В методе TTreeItem::GetLeft все указатели заменены на std::shared\_ptr<TTreeItem>

**main.cpp:**

#include <cstdlib>

#include <iostream>

#include <memory>

#include "figure.h"

#include "hexagon.h"

#include "triangle.h"

#include "octagon.h"

#include "tree.h"

#include "treeitem.h"

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

{

TTree tree;

std::shared\_ptr<Figure> f;

int figure, size, count = 0, key = 1;

std::cout << "What figure you want to create:" << std::endl;

std::cout << "1) triangle" << std::endl << "2) hexagon" << std::endl << "3) octagon" << std::endl;

std::cout << "Input number of figure" << std::endl << "Another number stops creating" << std::endl;

std::cin >> figure;

while(figure >= 1 && figure <= 3)

{

std::cout << "Input figure's size" << std::endl;

std::cin >> size;

if(figure == 1)

{

f = std::make\_shared<Triangle>(Triangle(size));

tree.push(f, key);

}

else if(figure == 2)

{

f = std::make\_shared<Hexagon>(Hexagon(size));

tree.push(f, key);

}

else if(figure == 3)

{

f = std::make\_shared<Octagon>(Octagon(size));

tree.push(f, key);

}

count++;

key = 0;

std::cin >> figure;

}

std::cout << "\n\n" << tree << std::endl << std::endl;

while(count > 0)

{

f = tree.pop();

f->Print();

count--;

}

return 0;

}

**Makefile:**

Добавлены действия:

octagon.o: octagon.cpp octagon.h

g++ -c -g -Wall octagon.cpp

triangle.o: triangle.cpp triangle.h

g++ -c -g -Wall triangle.cpp

Изменены действия:

main.exe: main.o tree.o hexagon.o treeitem.o triangle.o octagon.o

g++ -Wall -g main.o tree.o hexagon.o treeitem.o triangle.o octagon.o -o main.exe

main.o: main.cpp treeitem.h hexagon.h tree.h octagon.h triangle.h figure.h

g++ -c -g -Wall main.cpp

## Тесты:

|  |  |
| --- | --- |
| Входные данные | Ожидаемый результат |
| 1 34 2 45 3 67 3 46 2 13 1 34 5 | Hexagone's side = 13 Triangle's side = 34 Triangle's side = 34 Hexagone's side = 45 Octagon's side = 46 Octagon's side = 67 |
| 2 9000 1 23 3 45 4 | Triangle's side = 23 Octagon's side = 45 Hexagone's side = 9000 |
| 3 1 2 3 1 1 2 34 1 45 3 2 5 | Triangle's side = 1 Octagon's side = 1 Octagon's side = 2 Hexagone's side = 3 Triangle's side = 45 Hexagone's side = 34 |

Московский Авиационный Институт

(Национальный Исследовательский Университет)

Кафедра: 806 «Вычислительная математика и программирование»

Факультет: «Прикладная математика и информатика»

Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа №4.

по дисциплине «Объектно-ориентированное программирование»

|  |  |
| --- | --- |
| Студент: | Лисовский О.Р. |
| Группа: | 8О-208Б, №11 |
| Преподаватель: | Поповкин А.В. |
| Вариант | №15 |
| Оценка: |  |
| Дата: |  |

Москва 2018

## Цель работы:

Знакомство с шаблонами классов; построение шаблонов динамических структур данных.

## Задание:

Необходимо спроектировать и запрограммировать на языке C++ шаблон класса-контейнера [первого уровня](#_Варианты_задания_(структуры), содержащий все три фигуры класса фигуры, согласно [вариантов задания](#_Варианты_задания_(фигуры)) (реализованную в [ЛР1](#_Лабораторная_работа_№1)).

Классы должны удовлетворять следующим правилам:

Требования к классам фигуры аналогичны требованиям из [лабораторной работы 1](#_Задание); шаблон класса-контейнера должен содержать объекты используя std: shared\_ptr<…>; шаблон класса-контейнера должен иметь метод по добавлению фигуры в контейнер; шаблон класса-контейнера должен иметь методы по получению фигуры из контейнера (определяется структурой контейнера); шаблон класса-контейнера должен иметь метод по удалению фигуры из контейнера (определяется структурой контейнера); шаблон класса-контейнера должен иметь перегруженный оператор по выводу контейнера в поток std::ostream (<<); шаблон класса-контейнера должен иметь деструктор, удаляющий все элементы контейнера; классы должны быть расположенны в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

Нельзя использовать:

Стандартные контейнеры std.

Программа должна позволять:

Вводить произвольное количество фигур и добавлять их в контейнер; распечатывать содержимое контейнера; удалять фигуры из контейнера.

## 

## Код программы:

Файлы, не претерпевшие изменений: figure.h, hexagon.h, hexagon.cpp, octagon.h, octagon.cpp, triangle.h, triangle.cpp, Makefile

**tree.h:**

Убран прототип:

bool empty();

Класс TTree шаблонизирован:

template <class F> class TTree

std::shared\_ptr<TTreeItem<F>> seed;

**tree.cpp:**

Убран метод:

bool TTree::empty()

Работа с классом TTree шаблонизирована:

#include "figure.h"

template class TTree<Figure>;

template std::ostream& operator<<(std::ostream& os, const TTree<Figure>& obj);

**treeitem.h:**

Класс TTreeItem шаблонизирован:

template <class F> class TTreeItem

std::shared\_ptr<F> figure;

std::shared\_ptr<TTreeItem<F>> left;

std::shared\_ptr<TTreeItem<F>> right;

**treeitem.cpp:**

Класс TTreeItem шаблонизирован:

#include "figure.h"

template class TTreeItem<Figure>;

template <class Figure> std::ostream& operator<<(std::ostream& os, const TTreeItem<Figure>& obj);

**main.cpp:**

Изменена строка:

TTree<Figure> tree;

## Тесты:

|  |  |
| --- | --- |
| Входные данные | Ожидаемый результат |
| 1 1 2 2 2 2 3 3 2 23 5 | Triangle's side = 1 Hexagone's side = 2 Hexagone's side = 2 Octagon's side = 3 Hexagone's side = 23 |
| 3 2 1 45 2 45 3 643 5 | Octagon's side = 2 Triangle's side = 45 Hexagone's side = 45 Octagon's side = 643 |
| 2 34 3 457 1 24577 3 1 2 68 1 35 3 468 2 457 5 | Octagon's side = 1 Triangle's side = 35 Hexagone's side = 34 Hexagone's side = 68 Hexagone's side = 457 Octagon's side = 457 Octagon's side = 468 Triangle's side = 24577 |

Московский Авиационный Институт

(Национальный Исследовательский Университет)

Кафедра: 806 «Вычислительная математика и программирование»

Факультет: «Прикладная математика и информатика»

Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа №5.

по дисциплине «Объектно-ориентированное программирование»

|  |  |
| --- | --- |
| Студент: | Лисовский О.Р. |
| Группа: | 8О-208Б, №11 |
| Преподаватель: | Поповкин А.В. |
| Вариант | №15 |
| Оценка: |  |
| Дата: |  |

Москва 2018

Цель работы:

Закрепление навыков работы с шаблонами классов; построение итераторов для динамических структур данных.

## Задание:

Используя структуры данных, разработанные для предыдущей лабораторной работы ([ЛР№4](#_Лабораторная_работа_№4)) спроектировать и разработать Итератор для динамической структуры данных. Итератор должен быть разработан в виде шаблона и должен уметь работать со всеми типами фигур, согласно [варианту задания](#_Варианты_задания_(фигуры)_1). Итератор должен позволять использовать структуру данных в операторах типа for.

Нельзя использовать:

Стандартные контейнеры std.

Программа должна позволять:

Вводить произвольное количество фигур и добавлять их в контейнер; распечатывать содержимое контейнера; удалять фигуры из контейнера.

## Код программы:

Файлы, не претерпевшие изменений: hexagon.h, hexagon.cpp, octagon.h, octagon.cpp, triangle.h, triangle.cpp

**figure.h:**

Добавлен прототип:

friend std::ostream& operator <<(std::ostream& os, Figure& obj)

{

obj.Print();

return os;

}

**iterator.h:**

#ifndef ITERATOR\_H

#define ITERATOR\_H

#include <memory>

#include "treeitem.h"

#include "tree.h"

template<class F, class node> class TIterator

{

public:

TIterator(std::shared\_ptr<TTreeItem<F>> seed)

{

tree = seed;

}

std::shared\_ptr<F> operator\*()

{

return tree->GetFigure();

}

std::shared\_ptr<F> operator->()

{

return tree->GetFigure();

}

TIterator<F, node>& operator++()

{

if(tree->CheckRight() != false)

{

tree->Increase();

tree = tree->GetRight();

tree = tree->Min(tree);

}

else

{

tree = tree->GetParent();

}

return \*this;

}

bool operator==(const TIterator<F, node> &inend)

{

return this->tree == inend.tree;

}

bool operator!=(const TIterator<F, node> &inend)

{

return this->tree != inend.tree;

}

private:

std::shared\_ptr<TTreeItem<F>> tree;

};

#endif

**tree.h:**

Добавлена строка:

#include "iterator.h"

Добавлены прототипы:

std::shared\_ptr<TTreeItem<F>> GetSeed();

TIterator<F, TTreeItem<F>> begin();

TIterator<F, TTreeItem<F>> end();

**tree.cpp:**

Добавлены методы:

template <class F> TIterator<F, TTreeItem<F>> TTree<F>::begin()

{

return TIterator<F, TTreeItem<F>>(seed->Min(seed));

}

template <class F> TIterator<F, TTreeItem<F>> TTree<F>::end()

{

return TIterator<F, TTreeItem<F>>(nullptr);

}

template <class F> std::shared\_ptr<TTreeItem<F>> TTree<F>::GetSeed()

{

return this->seed;

}

Изменён метод:

template <class F> void TTree<F>::push(std::shared\_ptr<F> figure, int k)

{

seed = seed->Set(nullptr, seed, figure, k);

}

**treeitem.h:**

Добавлены прототипы:

bool CheckRight();

std::shared\_ptr<TTreeItem<F>> Min(std::shared\_ptr<TTreeItem<F>> seed);

std::shared\_ptr<TTreeItem<F>> Max(std::shared\_ptr<TTreeItem<F>> seed);

std::shared\_ptr<TTreeItem<F>> GetRight();

std::shared\_ptr<TTreeItem<F>> GetParent();

void Increase();

void NullPar();

Изменены прототипы:

std::shared\_ptr<TTreeItem<F>> Set(std::shared\_ptr<TTreeItem<F>> papa, std::shared\_ptr<TTreeItem<F>> seed, std::shared\_ptr<F> figure, int k);

Добавлены новые private переменные к классу TTreeItem:

std::shared\_ptr<TTreeItem<F>> parent;

int go;

**treeitem.cpp:**

Добавлены методы:

template <class F> bool TTreeItem<F>::CheckRight()

{

if (this->right == nullptr)

return false;

else

return true;

}

template <class F> std::shared\_ptr<TTreeItem<F>> TTreeItem<F>::GetRight()

{

return this->right;

}

template <class F> std::shared\_ptr<TTreeItem<F>> TTreeItem<F>::GetParent()

{

return this->parent;

}

template <class F> void TTreeItem<F>::Increase()

{

++go;

return;

}

template <class F> void TTreeItem<F>::NullPar()

{

this->parent = nullptr;

return;

}

template <class F> std::shared\_ptr<TTreeItem<F>> TTreeItem<F>::Min(std::shared\_ptr<TTreeItem<F>> seed)

{

std::shared\_ptr<TTreeItem<F>> min;

if(seed->left != nullptr)

{

min = Min(seed->left);

}

else

{

min = seed;

}

return min;

}

template <class F> std::shared\_ptr<TTreeItem<F>> TTreeItem<F>::Max(std::shared\_ptr<TTreeItem<F>> seed)

{

std::shared\_ptr<TTreeItem<F>> max;

if(seed->right != nullptr)

{

max = Max(seed->right);

}

else

{

max = seed;

}

return max;

}

Изменены методы:

template <class F> TTreeItem<F>::TTreeItem()

{

this->figure = nullptr;

this->left = nullptr;

this->right = nullptr;

this->parent = nullptr;

this->go = 0;

std::cout << "Tree item: created" << std::endl;

}

template <class F> TTreeItem<F>::TTreeItem(std::shared\_ptr<F>& figure)

{

this->figure = figure;

this->left = nullptr;

this->right = nullptr;

this->parent = nullptr;

this->go = 0;

std::cout << "Tree item: created" << std::endl;

}

template <class F> std::shared\_ptr<TTreeItem<F>> TTreeItem<F>::Set(std::shared\_ptr<TTreeItem<F>> papa , std::shared\_ptr<TTreeItem<F>> seed, std::shared\_ptr<F> figure, int k)

{

if(k == 1)

{

std::shared\_ptr<TTreeItem<F>> newTreeItem = std::make\_shared<TTreeItem<F>>(\*(new TTreeItem<F>(figure)));

newTreeItem->parent = papa;

return newTreeItem;

}

else if(seed->figure->Square() > figure->Square())

{

if(seed->left == nullptr)

k = 1;

papa = seed;

seed->left = seed->left->Set(papa, seed->left, figure, k);

}

else if(seed->figure->Square() <= figure->Square())

{

if(seed->right == nullptr)

k = 1;

papa = seed;

seed->right = seed->right->Set(papa, seed->right, figure, k);

}

return seed;

}

**main.cpp:**

Добавлены строки:

std::shared\_ptr<TTreeItem<Figure>> tmp = tree.GetSeed();

tmp = tmp->Max(tmp);

tmp->NullPar();

for(auto i : tree) std::cout << \*i << std::endl;

Изменены строки:

f = std::shared\_ptr<Triangle>(new Triangle(size));

f = std::shared\_ptr<Hexagon>(new Hexagon(size));

f = std::shared\_ptr<Octagon>(new Octagon(size));

Убраны строки:

while(count > 0)

{

f = tree.pop();

f->Print();

count--;

}

**Makefile:**

Изменено действие:

tree.o: tree.cpp tree.h iterator.h

g++ -c -g -Wall tree.cpp

## Тесты:

|  |  |
| --- | --- |
| Входные данные | Ожидаемый результат |
| 1 1 2 2 3 3 2 2 1 1 3 3 5 | Triangle's side = 1 Triangle's side = 1 Hexagone's side = 2 Hexagone's side = 2 Octagon's side = 3 Octagon's side = 3 |
| 3 34 2 647 1 24677 2 32 1 34 4 | Triangle's side = 34 Hexagone's side = 32 Octagon's side = 34 Hexagone's side = 647 Triangle's side = 24677 |
| 3 21 3 2 1 4 2 57 3 56 5 | Triangle's side = 4 Octagon's side = 2 Octagon's side = 21 Hexagone's side = 57 Octagon's side = 56 |

Московский Авиационный Институт

(Национальный Исследовательский Университет)

Кафедра: 806 «Вычислительная математика и программирование»

Факультет: «Прикладная математика и информатика»

Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа №6.

по дисциплине «Объектно-ориентированное программирование»

|  |  |
| --- | --- |
| Студент: | Лисовский О.Р. |
| Группа: | 8О-208Б, №11 |
| Преподаватель: | Поповкин А.В. |
| Вариант | №15 |
| Оценка: |  |
| Дата: |  |

Москва 2018

## Цель работы:

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

## Задание:

Используя структуры данных, разработанные для предыдущей лабораторной работы ([ЛР№5](#_Лабораторная_работа_№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 варианта задания](#_Варианты_задания_(структуры_1)). 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.

## Код программы:

Файлы, не претерпевшие изменений: figure.h, hexagon.h, hexagon.cpp, octagon.h, octagon.cpp, triangle.h, triangle.cpp, iterator.h, tree.h, tree.cpp, main.cpp

**treeitem.h:**

Добавлена строка:

#include "allocation.h"

Добавлены прототипы:

TTreeItem(std::shared\_ptr<F>\* figure);

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

void operator delete(void \*p);

Добавлена новая private переменная:

static Allocator treeitem\_allocator;

**treeitem.cpp:**

Добавлена строка:

template <class F> Allocator TTreeItem<F>::treeitem\_allocator(sizeof(TTreeItem<F>), 127);

Добавлены методы:

template <class F> void \* TTreeItem<F>::operator new(size\_t size)

{

return treeitem\_allocator.allocate();

}

template <class F> void TTreeItem<F>::operator delete(void \*p)

{

treeitem\_allocator.deallocate(p);

}

Изменен метод:

template <class F> std::shared\_ptr<TTreeItem<F>> TTreeItem<F>::DelItem(std::shared\_ptr<TTreeItem<F>> oldSeed, std::shared\_ptr<TTreeItem<F>> seed)

{

if (seed != oldSeed)

{

seed->left = seed->DelItem(oldSeed, seed->left);

}

else

{

if(seed->right != nullptr)

{

treeitem\_allocator.deallocate(seed.get());

seed = seed->right;

}

else

{

treeitem\_allocator.deallocate(seed.get());

seed = nullptr;

}

}

return seed;

}

**allocation.h:**

#ifndef TALLOCATION\_H

#define TALLOCATION\_H

#include <cstdlib>

#include <memory>

#include "alloctree.h"

class Allocator

{

public:

Allocator(size\_t size, size\_t count);

void \*allocate();

void deallocate(void \*pointer);

virtual ~Allocator();

private:

size\_t \_size;

size\_t \_count;

char \*\_used\_blocks;

size\_t \_free\_count;

std::shared\_ptr<Alloctree> \_tree;

};

#endif

**allocation.cpp:**

#include "allocation.h"

#include <iostream>

Allocator::Allocator(size\_t size, size\_t count): \_size(size), \_count(count)

{

\_used\_blocks = (char\*)malloc(\_size \* \_count);

\_free\_count = count;

for(size\_t i = 0; i < \_count; i++)

{

\_tree = \_tree->add(\_used\_blocks + i \* \_size, \_tree);

}

std::cout << "allocation: memory init" << std::endl;

}

void \*Allocator::allocate()

{

char \*result = nullptr;

if(\_free\_count > 0)

{

result = \_tree->push(\_free\_count);

\_free\_count--;

}

else

{

std::cout << "allocation: No memory exception" << std::endl;

}

return result;

}

void Allocator::deallocate(void \*pointer)

{

\_tree->killtree((char\*)pointer, \_free\_count);

std::cout << "allocation: Deallocate block " << std::endl;

\_free\_count++;

}

Allocator::~Allocator()

{

\_tree = nullptr;

if(\_free\_count < \_count)

std::cout << "allocation: memory leak?" << std::endl;

else

std::cout << "allocation: memory freed" << std::endl;

delete \_used\_blocks;

}

**alloctree.h:**

#ifndef TALLOCTREE\_H

#define TALLOCTREE\_H

#include <memory>

class Alloctree

{

public:

Alloctree();

std::shared\_ptr<Alloctree> add(char \*\_used\_blocks, std::shared\_ptr<Alloctree> seed);

void killtree(char \*pointer, size\_t \_count);

char \*push(size\_t \_count);

virtual ~Alloctree();

private:

char \*\_here;

std::shared\_ptr<Alloctree> \_left;

std::shared\_ptr<Alloctree> \_right;

};

#endif

**alloctree.cpp:**

#include "alloctree.h"

#include <iostream>

Alloctree::Alloctree()

{

this->\_here = nullptr;

this->\_left = nullptr;

this->\_right = nullptr;

}

std::shared\_ptr<Alloctree> Alloctree::add(char \*\_used\_blocks, std::shared\_ptr<Alloctree> seed)

{

if(seed == nullptr)

{

std::shared\_ptr<Alloctree> tree(new Alloctree);

tree->\_here = \_used\_blocks;

return tree;

}

else if(seed->\_left == nullptr)

{

seed->\_left = seed->\_left->add(\_used\_blocks, seed->\_left);

return seed;

}

else

{

seed->\_right = seed->\_right->add(\_used\_blocks, seed->\_right);

return seed;

}

}

void Alloctree::killtree(char \*pointer, size\_t \_count)

{

if(\_count > 1)

{

this->\_right->killtree(pointer, \_count - 2);

}

else if(\_count > 0)

{

this->\_left->killtree(pointer, 0);

}

else

{

this->\_here = pointer;

}

}

char\* Alloctree::push(size\_t \_count)

{

if(\_count == 1)

{

return this->\_here;

}

else if(\_count == 2)

{

\_count--;

return this->\_left->push(\_count);

}

else

{

\_count--;

\_count--;

return this->\_right->push(\_count);

}

}

Alloctree::~Alloctree(){

}

**main.cpp:**

Добавлены строки:

#include "allocation.h"

while(count > 0)

{

f = tree.pop();

f->Print();

count--;

}

**Makefile:**

Добавлены действия:

allocation.o: allocation.cpp allocation.h

g++ -c -g -Wall allocation.cpp

alloctree.o: alloctree.cpp alloctree.h

g++ -c -g -Wall alloctree.cpp

Изменены действия:

main.exe: main.o tree.o hexagon.o treeitem.o triangle.o octagon.o allocation.o alloctree.o

g++ -Wall -g main.o tree.o hexagon.o treeitem.o triangle.o octagon.o allocation.o alloctree.o -o main.exe

## Тесты:

|  |  |
| --- | --- |
| Входные данные | Ожидаемый результат |
| 1 23 2 34 3 45 2 34 3 5467 1 35 5 | Triangle's side = 23 Triangle's side = 35 Hexagone's side = 34 Hexagone's side = 34 Octagon's side = 45 Octagon's side = 5467 allocation: memory freed |
| 1 2 1 3 1 4 2 3 4 | Triangle's side = 2 Triangle's side = 3 Triangle's side = 4 Hexagone's side = 3 allocation: memory freed |
| 1 4 2 78 3 78 1 1 4 | Triangle's side = 1 Triangle's side = 4 Hexagone's side = 78 Octagon's side = 78 allocation: memory freed |

Московский Авиационный Институт

(Национальный Исследовательский Университет)

Кафедра: 806 «Вычислительная математика и программирование»

Факультет: «Прикладная математика и информатика»

Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа №7.

по дисциплине «Объектно-ориентированное программирование»

|  |  |
| --- | --- |
| Студент: | Лисовский О.Р. |
| Группа: | 8О-208Б, №11 |
| Преподаватель: | Поповкин А.В. |
| Вариант | №15 |
| Оценка: |  |
| Дата: |  |

Москва 2018

## Цель работы:

Создание сложных динамических структур данных; закрепление принципа OCP.

## Задание:

Необходимо реализовать динамическую структуру данных – «Хранилище объектов» и алгоритм работы с ней. «Хранилище объектов» представляет собой контейнер, одного из следующих видов ([Контейнер 1-го уровня](#_Варианты_задания_(структуры_2)). Каждым элементом контейнера, в свою, является динамической структурой данных одного из следующих видов ([Контейнер 2-го уровня](#_Варианты_задания_(структуры_3)). Таким образом у нас получается контейнер в контейнере. Элементом второго контейнера является объект-фигура, [определенная вариантом задания](#_Варианты_задания_(фигуры)_2). При этом должно выполняться правило, что количество объектов в контейнере второго уровня не больше 5. Т.е. если нужно хранить больше 5 объектов, то создается еще один контейнер второго уровня. Объекты в контейнерах второго уровня должны быть отсортированы по возрастанию площади объекта (в том числе и для деревьев). При удалении объектов должно выполняться правило, что контейнер второго уровня не должен быть пустым. Т.е. если он становится пустым, то он должен удалится.

Нельзя использовать:

Стандартные контейнеры std.

Программа должна позволять:

Вводить произвольное количество фигур и добавлять их в контейнер; распечатывать содержимое контейнера (1-го и 2-го уровня); удалять фигуры из контейнера по критериям (типу и площади).

## Код программы:

Файлы, не претерпевшие изменений: figure.h, hexagon.h, hexagon.cpp, octagon.h, octagon.cpp, triangle.h, triangle.cpp, treeitem.h, treeitem.cpp, allocation.h, allocation.cpp, alloctree.h, alloctree.cpp

**tree.h:**

Добавлены прототипы:

std::shared\_ptr<TTree<F>> GetRight();

void Del(char\*, int);

int GetCount();

Добавлены дополнительные частные переменные:

std::shared\_ptr<TTree<F>> right;

std::shared\_ptr<TTree<F>> left;

int count;

Убраны прототипы:

TIterator<F, TTreeItem<F>> begin();

TIterator<F, TTreeItem<F>> end();

**tree.cpp:**

Добавлены методы:

template <class F>

int TTree<F>::GetCount()

{

return this->count;

}

template <class F>

std::shared\_ptr<TTree<F>> TTree<F>::GetRight()

{

return this->right;

}

template <class F>

void TTree<F>::Del(char\* way, int size)

{

if(way[size] == '0')

{

std::shared\_ptr<TTreeItem<F>> oldSeed;

oldSeed = seed->GetLeft(seed);

seed = seed->DelItem(oldSeed, seed);

oldSeed = nullptr;

count--;

}

else if(way[size] == 'l' && left != nullptr)

{

left->Del(way, size + 1);

if(left->GetCount() == 0)

{

left = nullptr;

}

}

else if(way[size] == 'r' && right != nullptr)

{

right->Del(way, size + 1);

if(right->GetCount() == 0)

{

right = nullptr;

}

}

else

{

std::cout << "Чё-то не то в пути" << std::endl;

}

return;

}

Изменены методы:

template <class F>

TTree<F>::TTree()

{

this->seed = nullptr;

this->left = nullptr;

this->right = nullptr;

this->count = 0;

std::cout<< "Global tree: created" << std::endl;

}

template <class F>

std::ostream& operator<<(std::ostream& os, const TTree<F>& tree)

{

if(tree.left != nullptr)

{

std::cout << \*tree.left;

}

std::shared\_ptr<TTreeItem<F>> item = tree.seed;

if(item == nullptr)

{

return os;

}

item->Show();

std::cout << std::endl;

if(tree.right != nullptr)

{

std::cout << \*tree.right;

}

return os;

}

template <class F>

void TTree<F>::push(std::shared\_ptr<F> figure, int k)

{

if(this->count < 5)

{

seed = seed->Set(nullptr, seed, figure, k);

count++;

}

else if(this->seed->GetFigure()->Square() < figure->Square())

{

if(right == nullptr)

{

k = 1;

std::shared\_ptr<TTree<F>> newtree = std::shared\_ptr<TTree<Figure>>(new TTree<Figure>);

newtree->push(figure, k);

right = newtree;

}

else

{

right->push(figure, k);

}

}

else

{

if(left == nullptr)

{

k = 1;

std::shared\_ptr<TTree<F>> newtree = std::shared\_ptr<TTree<Figure>>(new TTree<Figure>);

newtree->push(figure, k);

left = newtree;

}

else

{

left->push(figure, k);

}

}

}

template <class F>

std::shared\_ptr<F> TTree<F>::pop()

{

int k = 0;

std::shared\_ptr<F> result;

if(left != nullptr && left->GetCount() > 0)

{

result = left->pop();

}

else if(count > 0)

{

std::shared\_ptr<TTreeItem<F>> oldSeed;

if(seed != nullptr)

{

if(seed->CheckLeft())

{

oldSeed = seed->GetLeft(seed);

result = oldSeed->GetFigure();

seed = seed->DelItem(oldSeed, seed);

oldSeed = nullptr;

count--;

k = 1;

}

else

{

oldSeed = seed;

result = seed->GetFigure();

seed = seed->DelItem(oldSeed, seed);

oldSeed = nullptr;

count--;

k = 1;

}

}

}

else

{

result = right->pop();

}

if(k == 0 && left != nullptr &&left->GetCount() == 0)

{

left = left->GetRight();

}

return result;

}

Убраны методы:

template <class F> TIterator<F, TTreeItem<F>> TTree<F>::begin()

template <class F> TIterator<F, TTreeItem<F>> TTree<F>::end()

**main.cpp:**

Добавлены строки:

std::cout << "\n\n" << \*tree << std::endl;

std::cout << "Хотите удолить что-то из конкретного контейнеро? Введите путь к контейнеру состоящий из l, r и 0 - конечноя ветко. e - конец удоления" << std::endl;

char way[20];

scanf("%s", way);

while(way[0] != 'e')

{

tree->Del(way, 0);

count--;

scanf("%s", way);

}

std::cout << "\n\n" << \*tree << std::endl;

Изменены строки:

std::shared\_ptr<TTree<Figure>> tree = std::shared\_ptr<TTree<Figure>>(new TTree<Figure>);

tree->push(f, key);

Убраны строки:

std::shared\_ptr<TTreeItem<Figure>> tmp = tree.GetSeed();

tmp = tmp->Max(tmp);

tmp->NullPar();

for(auto i : tree) std::cout << \*i << std::endl;

**Makefile:**

Изменено действие:

tree.o: tree.cpp tree.h

g++ -c -g -Wall tree.cpp

## Тесты:

|  |  |  |
| --- | --- | --- |
| Входные данные | Ожидаемый результат до удаления нескольких элементов | Ожидаемый результат после удаления нескольких элементов |
| 2 3 1 5 2 6 3 8 2 7 3 56 1 1 6 l l0 r0 e | Triangle's side = 1 Triangle's side = 5 Hexagone's side = 3 Hexagone's side = 6 Hexagone's side = 7 Octagon's side = 8 Octagon's side = 56 | Triangle's side = 5 Hexagone's side = 3 Hexagone's side = 6 Hexagone's side = 7 Octagon's side = 8 |
| 3 3 3 3 3 3 3 3 3 3 1 1 1 1 1 1 1 1 1 1 1 1 2 2 5 l0 e | Triangle's side = 1 Triangle's side = 1 Triangle's side = 1 Triangle's side = 1 Triangle's side = 1 Triangle's side = 1 Hexagone's side = 2 Octagon's side = 3 Octagon's side = 3 Octagon's side = 3 Octagon's side = 3 Octagon's side = 3 | Triangle's side = 1 Triangle's side = 1 Triangle's side = 1 Triangle's side = 1 Triangle's side = 1 Hexagone's side = 2 Octagon's side = 3 Octagon's side = 3 Octagon's side = 3 Octagon's side = 3 Octagon's side = 3 |
| 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 3 2 2 4 e | Triangle's side = 1 Triangle's side = 1 Triangle's side = 1 Triangle's side = 1 Triangle's side = 1 Triangle's side = 1 Hexagone's side = 2 Octagon's side = 3 Octagon's side = 3 Octagon's side = 3 Octagon's side = 3 Octagon's side = 3 | Triangle's side = 1 Triangle's side = 1 Triangle's side = 1 Triangle's side = 1 Triangle's side = 1 Triangle's side = 1 Hexagone's side = 2 Octagon's side = 3 Octagon's side = 3 Octagon's side = 3 Octagon's side = 3 Octagon's side = 3 |

Московский Авиационный Институт

(Национальный Исследовательский Университет)

Кафедра: 806 «Вычислительная математика и программирование»

Факультет: «Прикладная математика и информатика»

Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа №8.

по дисциплине «Объектно-ориентированное программирование»

|  |  |
| --- | --- |
| Студент: | Лисовский О.Р. |
| Группа: | 8О-208Б, №11 |
| Преподаватель: | Поповкин А.В. |
| Вариант | №15 |
| Оценка: |  |
| Дата: |  |

Москва 2018

## Цель работы:

Знакомство с параллельным программированием в C++.

## Задание:

Используя структуры данных, разработанные для [лабораторной работы №6](#_Лабораторная_работа_№6_1) (контейнер первого уровня и классы-фигуры) разработать алгоритм быстрой сортировки для класса-контейнера.

Необходимо разработать два вида алгоритма:

1) Обычный, без параллельных вызовов.

2) С использованием параллельных вызовов. В этом случае, каждый рекурсивный вызов сортировки должен создаваться в отдельном потоке.

Для создания потоков использовать механизмы:

future; packaged\_task/async

Для обеспечения потоко-безопасности структур данных использовать:

mutex; lock\_guard

Нельзя использовать:

Стандартные контейнеры std.

Программа должна позволять:

Вводить произвольное количество фигур и добавлять их в контейнер; распечатывать содержимое контейнера; удалять фигуры из контейнера; проводить сортировку контейнера

## Код программы:

Файлы, не претерпевшие изменений: allocation.cpp, allocation.h, alloctree.cpp, alloctree.h, figure.h, hexagon.cpp, hexagon.h, iterator.h, octagon.cpp, octagon.h, triangle.h, triangle.cpp.

**tree.h:**

Добавлен прототип:

void NewPush(std::shared\_ptr<F>, int);

**tree.cpp:**

Добавлен метод:

template <class F>

void TTree<F>::NewPush(std::shared\_ptr<F> figure, int k)

{

seed = seed->NewSet(nullptr, seed, figure, k);

}

**treeitem.h:**

Добавлен прототип:

std::shared\_ptr<TTreeItem<F>> NewSet(std::shared\_ptr<TTreeItem<F>>, std::shared\_ptr<TTreeItem<F>>, std::shared\_ptr<F>, int);

**treeitem.cpp:**

Добавлен метод:

template <class F>

std::shared\_ptr<TTreeItem<F>> TTreeItem<F>::NewSet(std::shared\_ptr<TTreeItem<F>> papa, std::shared\_ptr<TTreeItem<F>> seed, std::shared\_ptr<F> figure, int k)

{

if(k == 1)

{

std::shared\_ptr<TTreeItem<F>> newTreeItem = std::make\_shared<TTreeItem<F>>(\*(new TTreeItem<F>(figure)));

newTreeItem->parent = papa;

return newTreeItem;

}

else if(go == 0)

{

go++;

if(seed->left == nullptr)

{

seed->left = NewSet(seed, seed->left, figure, 1);

}

else

{

seed->left = NewSet(seed, seed->left, figure, k);

}

}

else if(go == 1)

{

go--;

if(seed->right == nullptr)

{

seed->right = NewSet(seed, seed->right, figure, 1);

}

else

{

seed->right = NewSet(seed, seed->right, figure, k);

}

}

return seed;

}

**Makefile:**

Изменено действие:

main.exe: main.o tree.o hexagon.o treeitem.o triangle.o octagon.o allocation.o alloctree.o

g++ -Wall -g main.o tree.o hexagon.o treeitem.o triangle.o octagon.o allocation.o alloctree.o -o main.exe –lpthread

**main.cpp:**

#include <cstdlib>

#include <iostream>

#include <memory>

#include "figure.h"

#include "hexagon.h"

#include "triangle.h"

#include "octagon.h"

#include "tree.h"

#include "treeitem.h"

#include "allocation.h"

#include <future>

#include <mutex>

#include <thread>

std::mutex stop;

std::shared\_ptr<Figure>\* forsort = nullptr;

int SetMas(int count)

{

int i;

for(i = 0; i < count; i++)

{

forsort[i] = nullptr;

}

return 0;

}

int Sort(int maxCount, int minCount)

{

int i, j;

std::shared\_ptr<Figure> swap;

for(i = minCount; i < maxCount - 1; i++)

{

for(j = i + 1; j < maxCount; j++)

{

if(forsort[j]->Square() < forsort[i]->Square())

{

swap = forsort[j];

forsort[j] = forsort[i];

forsort[i] = swap;

}

}

}

stop.lock();

return 0;

}

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

{

TTree<Figure> tree;

std::shared\_ptr<Figure> f;

int figure, size, count = 0;

std::cout << "What figure you want to create:" << std::endl;

std::cout << "1) triangle" << std::endl << "2) hexagon" << std::endl << "3) octagon" << std::endl;

std::cout << "Input number of figure" << std::endl << "Another number stops creating" << std::endl;

std::cin >> figure;

while(figure >= 1 && figure <= 3)

{

std::cout << "Input figure's size" << std::endl;

std::cin >> size;

if(figure == 1)

{

f = std::shared\_ptr<Triangle>(new Triangle(size));

tree.NewPush(f, count + 1);

}

else if(figure == 2)

{

f = std::shared\_ptr<Hexagon>(new Hexagon(size));

tree.NewPush(f, count + 1);

}

else if(figure == 3)

{

f = std::shared\_ptr<Octagon>(new Octagon(size));

tree.NewPush(f, count + 1);

}

count++;

std::cin >> figure;

}

std::cout << "\n\n" << std::endl;

std::cout << tree << std::endl;

std::cout << "\n\n" << std::endl;

int sort;

std::cout << "Выберете сортировку:\n1 - стандартная однопоточная\n2 - многопоточная\nОстальные варианты отменяют сортировку" << std::endl;

std::cin >> sort;

int kount;

if(sort == 1)

{

TTree<Figure> remake;

f = tree.pop();

remake.push(f, 1);

for(kount = 1; kount < count; kount++)

{

f = tree.pop();

remake.push(f, 0);

}

std::cout << "\n\n" << remake << "\n\n" << std::endl;

while(count > 0)

{

f = remake.pop();

f->Print();

count--;

}

}

else if (sort == 2)

{

std::cout << "Начата параллельная сортировка" << std::endl;

forsort = (std::shared\_ptr<Figure>\*)malloc(count \* sizeof(std::shared\_ptr<Figure>));

std::future<int> prepare = std::async(std::launch::async, SetMas, count);

kount = prepare.get();

for(kount = 0; kount < count; kount++)

{

f = tree.pop();

forsort[kount] = f;

}

int size = count / 4;

std::future<int> first = std::async(std::launch::async, Sort, size \* 2, 0);

std::future<int> second = std::async(std::launch::async, Sort, count, size \* 2);

kount = first.get();

stop.unlock();

kount = second.get();

stop.unlock();

kount = Sort(count, 0);

stop.unlock();

for(kount = 0; kount < count; kount++)

{

forsort[kount]->Print();

}

delete forsort;

}

else

{

while(count > 0)

{

f = tree.pop();

f->Print();

count--;

}

}

return 0;

}

## Тесты:

|  |  |
| --- | --- |
| Входные данные | Ожидаемый результат |
| 2 2 2 2 1 1 3 3 4 2 | Triangle's side = 1 Hexagone's side = 2 Hexagone's side = 2 Octagon's side = 3 |
| 1 2 1 3 2 4 3 1 2 6 2 5 3 6 1 1 2 2 3 3 4 2 | Triangle's side = 1 Triangle's side = 2 Triangle's side = 3 Octagon's side = 1 Hexagone's side = 2 Hexagone's side = 4 Octagon's side = 3 Hexagone's side = 5 Hexagone's side = 6 Octagon's side = 6 |
| 1 2 1 3 2 4 3 1 2 6 2 5 3 6 1 1 2 2 3 3 4 1 | Triangle's side = 1 Triangle's side = 2 Triangle's side = 3 Octagon's side = 1 Hexagone's side = 2 Hexagone's side = 4 Octagon's side = 3 Hexagone's side = 5 Hexagone's side = 6 Octagon's side = 6 |

# Заключение

При выполнении данных лабораторных работ я познакомился с классами и научился программировать их на языке C++, управлять памятью, перегружать операторы, познакомился с дружественными функциями и библиотечными операциями ввода-вывода в C++, изучил базовые понятия ООП. Я научился создавать простые динамические структуры и работать со структурами, передаваемыми «по значению». Я закрепил навыки работы с классами и научился работать с умными указателями. Я научился шаблонизировать классы в C++ и закрепил работу с ними. Я научился создавать итераторы для динамических структур данных. Я научился создавать аллокаторы памяти для динамических структур данных тем самым закрепив работу с памятью в C++. Я научился создавать сложные структуры данных и закрепил принцип OCP. Я освоил параллельное программирование в C++.