Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Отчет по лабораторным работам
по курсу
«Объектно-ориентированное программирование»

Студент: С.С. Лихарев
Преподаватель: А. В. Поповкин
Группа: М80-207Б
Вариант: 14
Дата:
Оценка:
Подпись:

Москва, 2017

Оглавление

Лабораторная работа №1

1.1. Цель работы

- Программирование классов на языке C++.
- Управление памятью в языке C++.
- 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.

1.2. Задание

Необходимо спроектировать и запрограммировать на языке C++ классы фигур, согласно варианту задания.
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.
- Должны 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.

Фигуры: пятиугольник, шестиугольник, восьмиугольник.

1.3. Описание

Чтобы перейти к выполнению всех лабораторных работ по объектно-ориентированному программированию (в дальнейшем будем применять сокращение этого слова: ООП), необходимо понимать, в чем отличие этого подхода от остальных, а также знать важнейшие понятия в ООП, которые часто будут встречаться в следующих работах.

Классы в C++ — это абстракция описывающая методы и свойства ещё не существующих объектов.

Объекты — конкретное представление абстракции, имеющее свои свойства и методы. Созданные объекты на основе одного класса называются экземплярами этого класса. Эти объекты могут иметь различное поведение, свойства, но все равно будут являться объектами одного класса. В ООП существует три основных принципа построения классов:

1. **Инкапсуляция** — это свойство, позволяющее объединить в классе и данные, и методы, работающие с ними и скрыть детали реализации от пользователя.

2. **Наследование** — это свойство, позволяющее создать новый класс-потомок на основе уже существующего, при этом все характеристики класса родителя присваиваются классу-потомку.

3. **Полиморфизм** — свойство классов, позволяющее использовать объекты классов с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

Перегрузка операторов — один из способов реализации полиморфизма. Цель такой перегрузки - позволять использовать одно и то же имя функции для некоторой базовой операции, даже несмотря на то, что она применяется к данным разных типов.

Существуют также два специальных типа функций-членов - конструктор и деструктор. **Конструктор** - это специальная функция-член класса, которая вызывается всякий раз при создании объекта данного класса, **деструктор** - при уничтожении этого же объекта. **Тип** определяет, какие операции, или методы, могут быть применены с использованием этого объекта данных.

Виртуальная функция - это функция-член, которая будет переопределена в производных классах.

Дружественная функция — это функция, которая не является членом класса, но имеет доступ к членам класса, объявленным в полях private или protected.

Частью стандартной библиотеки C++ является библиотека **iostream** – объектно-ориентированная иерархия классов, где используется и множественное, и виртуальное наследование. В ней реализована поддержка для файлового ввода/вывода данных встроенных типов. Кроме того, разработчики классов могут расширять эту библиотеку для чтения и записи новых типов данных. Для использования библиотеки iostream в программе необходимо включить заголовочный файл **#include <iostream>** Операции ввода/вывода выполняются с помощью классов **istream** (потоковый ввод) и **ostream** (потоковый вывод). Третий класс, iostream, является производным от них и поддерживает двунаправленный ввод/вывод. Для удобства в библиотеке определены три стандартных объекта-потока:

1. **cin** – объект класса istream, соответствующий стандартному вводу. В общем случае он позволяет читать данные с терминала пользователя;

2. **cout** – объект класса ostream, соответствующий стандартному выводу. В общем случае он позволяет выводить данные на терминал пользователя;

3. **cerr** – объект класса ostream, соответствующий стандартному выводу для ошибок. В этот поток мы направляем сообщения об ошибках программы.

Вывод осуществляется, как правило, с помощью перегруженного оператора сдвига влево ($<<$), а ввод – с помощью оператора сдвига вправо ($>>$).

1.4. Исходный код

pentagon.cpp	
Pentagon()	Конструктор класса.
Pentagon(std::istream &is);	Конструктор класса из стандартного потока ввода.
Pentagon(const Pentagon& orig);	Конструктор копии класса.
double Square();	Площадь фигуры.
void Print();	Печать фигуры.
~Pentagon();	Деструктор класса.
hexagon.cpp	
Функции hexagon аналогичны функциям pentagon.	
octagon.cpp	
Функции octagon аналогичны функциям pentagon.	

pentagon.h

```
1  class Figure {
2     public:
3        virtual double Square() = 0;
4        virtual void Print() = 0;
5        virtual ~Figure() {};
6  };
7
8  class Pentagon : public Figure {
9     public:
10        Pentagon();
11       Pentagon(std::istream& is);
12       Pentagon(size_t side);
13       Pentagon(const Pentagon& orig);
14       double Square() override;
15       void   Print()  override;
16
17       virtual ~Pentagon();
18    private:
19       size_t side;
20 };
```

pentagon.cpp

```
1  #include "pentagon.h"
2  #include <iostream>
3  #include <cmath>
4
5  Pentagon::Pentagon() : Pentagon(0) {
6  }
7
8  Pentagon::Pentagon(size_t side) {
```

```
 9      std::cout << "Pentagon: created\n";
10  }
11
12  Pentagon::Pentagon(std::istream &is) {
13      long long tmp_side;
14      std::cout << "Pentagon: enter side length: ";
15      while(true){
16          is >> tmp_side;
17          is.clear();
18          is.sync();
19          if(tmp_side <= 0) {
20              std::cerr << "Error: The side of any figure must
                    be > 0. Try again: ";
21          } else {
22              side = tmp_side;
23              break;
24          }
25      }
26      std::cout << "Pentagon: created\n";
27  }
28
29  Pentagon::Pentagon(const Pentagon& orig) {
30      std::cout << "Pentagon: created via copy ctr" << std::
            endl;
31      side = orig.side;
32  }
33
34  double Pentagon::Square() {
35      std::cout << "Pentagon: square: ";
36      return double((5.0 * side * side) / (4.0 * tan(M_PI /
            5.0)));
37  }
38
39  void Pentagon::Print() {
40      std::cout << "Pentagon: side length = " << side << std::
            endl;
41  }
42
43  Pentagon::~Pentagon() {
44      std::cout << "Pentagon: deleted " << std::endl;
45  }
```

Классы и реализации шестиугольника (Hexagon) и восьмиугольника (Octagon) аналогичны классу и реализациям Pentagon.

main.cpp

```
1  #include <iostream>
2  #include <cstdlib>
3  #include <cstring>
4
5  #include "pentagon.h"
6  #include "hexagon.h"
7  #include "octagon.h"
8
9  int main(int argc, char** argv) {
10     std::cout << "Use 'help' or 'h' to get help." << std::
          endl;
11     const int size = 16;
12     char s[size];
13
14     Figure * ptr_fig = nullptr;
15     while (1) {
16         std::cin.getline(s, size);
17         std::cin.clear();
18         std::cin.sync();
19         if (strcmp(s, "create") == 0 || strcmp(s, "cr") == 0)
               {
20             std::cout << "Which figure do you want to create?
                   (pent/hex/oct[agon]): ";
21             std::cin.getline(s, size);
22             std::cin.clear();
23             std::cin.sync();
24
25             if (strcmp(s, "pentagon") == 0 || strcmp(s, "pent
                  ") == 0) {
26                 if (ptr_fig != nullptr) delete ptr_fig;
27                 ptr_fig = new Pentagon(std::cin);
28             } else if (strcmp(s, "hexagon") == 0 || strcmp(s,
                   "hex") == 0) {
29                 if (ptr_fig != nullptr) delete ptr_fig;
30                 ptr_fig = new Hexagon(std::cin);
31             } else if (strcmp(s, "octagon") == 0 || strcmp(s,
                   "oct") == 0) {
32                 if (ptr_fig != nullptr) delete ptr_fig;
33                 ptr_fig = new Octagon(std::cin);
34             } else {
35                 std::cout << "Invalid choice. The figure has
                      not been created! " << std::endl;
36             }
37         } else if (strcmp(s, "print") == 0 || strcmp(s, "pr")
               == 0) {
```

```
38              if(ptr_fig == nullptr) {
39                  std::cout << "The figure doesn't exist." <<
                        std::endl;
40              } else {
41                  ptr_fig->Print();
42              }
43          } else if (strcmp(s, "square") == 0 || strcmp(s, "sq"
               ) == 0) {
44              if(ptr_fig == nullptr) {
45                  std::cout << "The figure doesn't exist." <<
                        std::endl;
46              } else {
47                  std::cout << ptr_fig->Square() << std::endl;
48              }
49          } else if (strcmp(s, "quit") == 0 || strcmp(s, "exit"
               ) == 0 || strcmp(s, "q") == 0) {
50              if (ptr_fig != nullptr) {
51                  delete ptr_fig;
52              }
53              break;
54          } else if (strcmp(s, "help") == 0 || strcmp(s, "h")
               == 0) {
55              std::cout << "\n\ncr[eate]              create new
                    figure";
56                std::cout << "\npr[int]               print <
                     side> of the figure";
57                std::cout << "\nsq[uare]              compute
                     square of the figure";
58                std::cout << "\nq[uit] or exit        exit the
                     program\n\n";
59            }
60      }
61
62      return 0;
63  }
```

Рассмотрим важный момент на строке 14. Это переменная типа указатель на объект Figure, но в него можно записать указатель на объект любого производного класса (например, Pentagon). Благодаря полиморфизму и в частности виртуальных функций, будут вызываться методы именно конкретных фигур, а не Figure (абстрактный родительский класс). В данных работах будем работать с правильными многоугольниками. У таких многоугольников все стороны равны.

1.5. Консоль

```
sergey@svb:~/MAI/OOP/lab1$ valgrind --leak-check=full ./run
==3183== Memcheck, a memory error detector
==3183== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==3183== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==3183== Command: ./run
==3183==
Use 'help' or 'h' to get help.
h


cr[eate]              create new figure
pr[int]               print <side> of the figure
sq[uare]              compute square of the figure
q[uit] or exit        exit the program

cr
Which figure do you want to create? (pent/hex/oct[agon]): pent
Pentagon: enter side length: 4
Pentagon: created
pr
Pentagon: side length = 4
sq
Pentagon: square: 27.5276
cr
Which figure do you want to create? (pent/hex/oct[agon]): oct
Pentagon: deleted
Octagon: enter side length: 4
Octagon: created
sq
Octagon: square: 77.2548
pr
Octagon: side length = 4
cr
Which figure do you want to create? (pent/hex/oct[agon]): hex
Octagon: deleted
Hexagon: enter side length: 5
Hexagon: created
sq
Hexagon: square: 64.9519
q
Hexagon: deleted
==3183==
==3183== HEAP SUMMARY:
==3183==     in use at exit: 72,704 bytes in 1 blocks
```

```
==3183==   total heap usage: 6 allocs, 5 frees, 74,800 bytes allocated
==3183==
==3183== LEAK SUMMARY:
==3183==    definitely lost: 0 bytes in 0 blocks
==3183==    indirectly lost: 0 bytes in 0 blocks
==3183==      possibly lost: 0 bytes in 0 blocks
==3183==    still reachable: 72,704 bytes in 1 blocks
==3183==         suppressed: 0 bytes in 0 blocks
==3183== Reachable blocks (those to which a pointer was found) are not shown.
==3183== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==3183==
==3183== For counts of detected and suppressed errors, rerun with: -v
==3183== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

1.6. Выводы

Данная работа познакомила меня с языком C++, а также с парадигмой ООП в целом. Очень важно знать такие вещи, как виртуальные методы, наследование, полиморфизм и т.д. ООП - это такой стиль программирования, который, в основном, можно применять в любом языке. Познакомился с классами, перегрузками операций. Классы - мощные и сложные средства. Они определяются как тип данных, предназначенный для представления объекта, и также, посредством функций-членов - операций, которые могут выполняться над этими данными.

Лабораторная работа №2

2.1. Цель работы

- Закрепление навыков работы с классами.
- Создание простых динамических структур данных.
- Работа с объектами, передаваемыми «по значению».

2.2. Задание

Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня, содержащий одну фигуру, согласно варианту задания.
Классы должны удовлетворять следующим правилам:

- Требования к классу фигуры аналогичны требованиям из лабораторной работы 1.
- Классы фигур должны иметь переопределенный оператор вывода в поток std::ostream(«). Оператор должен распечатывать параметры фигуры.
- Классы фигур должны иметь переопределенный оператор ввода фигуры из потока std::istream(»). Оператор должен вводить параметры фигуры.
- 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.
- Класс-контейнер должен иметь метод по удалению фигуры из 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).

Программа должна позволять:

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

- Удалять фигуры из контейнера.

Фигура: Пятиугольник.
Контейнер: Бинарное дерево.

2.3. Описание

Существуют такие программы, где мы заранее не знаем, сколько потребуется выделить памяти для решения задач. Становится известно только в процессе работы. В таких случаях применяются динамические структуры данных.

Бинарное дерево - это такая структура данных, которая является конечным множеством узлов. Оно может быть пустым или состоит из корня и двух непересекающихся поддеревьев (левого и правого).

При удалении или поиске конкретной фигуры мы создаем еще заданную фигуру с заданной стороной, затем мы сравниваем (с помощью переопределенных операторов сравнения) её с каждым ключом (фигурой) дерева. Если не обнаружили, то никаких изменений не произойдет и при этом фигура, которая была создана для сравнения с ключами, уничтожается.

В данной работе применяю вложенные классы. Они вводят новую область видимости в пределах класса и позволяют избегать конфликта имен. В бинарное дерево поиска передаются фигуры «по значению», чтобы в них хранился сам объект, а не его копия.

2.4. Исходный код

	pentagon.cpp	
	Pentagon();	Конструктор класса.
	Pentagon(size_t side);	Конструктор класса с параметром - сторона.
	Pentagon(const Pentagon& orig);	Конструктор копии класса.
	double Square();	Площадь фигуры.
	virtual ~Pentagon();	Деструктор класса.
	friend bool operator==(const Pentagon& left, const Pentagon& right); Аналогичные для >, <, !=, <= и >=.	Переопределенные операторы сравнения.
	Pentagon& operator=(const Pentagon& right);	Переопределенный оператор копирования.
	friend std::ostream& operator«(std::ostream& os, const Pentagon& obj);	Переопределенный оператор вывода в поток std::ostream.
	friend std::istream& operator»(std::istream& is, Pentagon& obj);	Переопределенный оператор ввода из потока std::istream.
	Внешний класс TBinaryTree	
public	TBinaryTree();	Конструктор класса.
	TBinaryTree(const TBinaryTree& orig);	Конструктор копирования.
	void Insert(const TItem & val);	Вставка нового ключа в бинарное дерево.
	bool IsEmpty() const;	Проверка, пусто ли дерево.
	TItem* Find(size_t side);	Поиск в бинарном дереве пятиугольника с размером side
	bool Delete(size_t side);	Удаление из дерева пятиугольника с размером side.
	TreeNode * MinValueTreeNode(TreeNode * node);	Поиск наименьшего значения в заданном поддереве.
	TreeNode * deleteTreeNode(TreeNode * _root, TItem & key);	Перестроение дерева в процессе удаления пятиугольника.
	std::ostream& InOrderPrint(std::ostream& os, TreeNode * node, size_t level) const;	Внутренняя рекурсивная функция для печати поддерева в симметрическом обходе.
	friend std::ostream& operator«(std::ostream& os, TBinaryTree& bintree);	Переопределенный оператор вывода в поток std::ostream.
	virtual ~TBinaryTree();	Деструктор класса.
private	**Внутренний класс TreeNode**	
public	TreeNode();	Конструктор класса.
	TreeNode(const TItem & data);	Конструктор копирования.

Классы:

```
1  class Pentagon : public Figure {
2     public:
3          Pentagon();
4         Pentagon(size_t side);
5         Pentagon(const Pentagon& orig);
6         double Square() override;
7
8           friend bool operator==(const Pentagon& left,const
               Pentagon& right);
9           friend bool operator>(const Pentagon& left,const
               Pentagon& right);
10          friend bool operator<(const Pentagon& left,const
               Pentagon& right);
11          friend bool operator!=(const Pentagon& left,const
               Pentagon& right);
12          friend bool operator<=(const Pentagon& left,const
               Pentagon& right);
13          friend bool operator>=(const Pentagon& left,const
               Pentagon& right);
14
15          friend std::ostream& operator<<(std::ostream& os,
               const Pentagon& obj);
16          friend std::istream& operator>>(std::istream& is,
               Pentagon& obj);
17          Pentagon& operator=(const Pentagon& right);
18
19        virtual ~Pentagon();
20    private:
21        int side;
22 };
23
24
25
26 typedef Pentagon TItem;
27 class TBinaryTree {
28     private:
29          class TreeNode {
30              public:
31                  TreeNode();
32              TreeNode(const TItem & data);
33                  TreeNode *left;
34                  TreeNode *right;
35                  TItem data;
36          };
37
```

```
38          TreeNode *root;
39      public:
40          TBinaryTree();
41          TBinaryTree(const TBinaryTree& orig);
42        void Insert(const TItem & val);
43          bool IsEmpty() const;
44          TItem* Find(size_t side);
45          bool Delete(size_t side);
46        TreeNode * MinValueTreeNode(TreeNode * node);
47        TreeNode * deleteTreeNode(TreeNode * _root, TItem & key
             );
48          std::ostream& InOrderPrint(std::ostream& os, TreeNode
               * node, size_t level) const;
49          friend std::ostream& operator<<(std::ostream& os,
              const TBinaryTree& bintree);
50          virtual ~TBinaryTree();
51  };
```

tbinary_tree.cpp

```
1   TBinaryTree::TBinaryTree() {
2       root = nullptr;
3   }
4
5   void TBinaryTree::Insert(const TItem& val){
6      if (root == nullptr)
7         root = new TreeNode(val);
8      else {
9         TreeNode * curNode = root;
10        TreeNode * parNode = nullptr;
11        while (curNode != nullptr) {
12           parNode = curNode;
13           if (val < curNode->data)
14              curNode = curNode->left;
15           else
16              curNode = curNode->right;
17        }
18        if (val < parNode->data)
19           parNode->left = new TreeNode(val);
20        else
21           parNode->right = new TreeNode(val);
22     }
23  }
24
25  TItem * TBinaryTree::Find(size_t side)
26  {
27     if (root == nullptr)
```

```
28          return nullptr;
29       else {
30          Pentagon penta_search(side);
31          TreeNode * curNode = root;
32          while (curNode != nullptr) {
33             if (penta_search < curNode->data)
34                curNode = curNode->left;
35             else if (penta_search > curNode->data)
36                curNode = curNode->right;
37             else
38                return &(curNode->data);
39          }
40       }
41       return nullptr;
42    }
43
44    TBinaryTree::TreeNode * TBinaryTree::MinValueTreeNode(
         TreeNode* node)
45    {
46       TreeNode* current = node;
47
48       while (current->left != nullptr)
49          current = current->left;
50
51       return current;
52    }
53
54    TBinaryTree::TreeNode*  TBinaryTree::deleteTreeNode(TreeNode*
          _root, TItem & key)
55    {
56       if (_root == nullptr) return _root;
57
58       if (key < _root->data)
59          _root->left = deleteTreeNode(_root->left, key);
60       else if (key > _root->data)
61          _root->right = deleteTreeNode(_root->right, key);
62       else {
63          if (_root->left == nullptr) {
64             TreeNode *temp = _root->right;
65             delete _root;
66             return temp;
67          }
68          else if (_root->right == nullptr) {
69             TreeNode *temp = _root->left;
70             delete _root;
71             return temp;
```

```
72        }
73
74        TreeNode* temp = MinValueTreeNode(_root->right);
75        _root->data = temp->data;
76        _root->right = deleteTreeNode(_root->right, temp->data)
              ;
77    }
78    return _root;
79 }
80
81 bool TBinaryTree::Delete(size_t side)
82 {
83    Pentagon penta_search(side);
84    root = deleteTreeNode(root, penta_search);
85    return false;
86 }
87
88 std::ostream& TBinaryTree::InOrderPrint(std::ostream& os,
      TreeNode * node, size_t level) const{
89     if(node != nullptr){
90         InOrderPrint(os, node->left, level + 1);
91         for(size_t i = 0; i < level; i++)
92             os << '\t';
93         os << node->data;
94         InOrderPrint(os, node->right, level + 1);
95     }
96     return os;
97 }
98
99 std::ostream& operator<<(std::ostream& os, const TBinaryTree&
       bintree){
100    if (bintree.IsEmpty())
101       os << "Empty tree\n";
102    else
103       os << "================================================\
             n";
104         bintree.InOrderPrint(os, bintree.root, 0);
105       os << "================================================\
             n\n";
106    return os;
107 }
108
109 bool TBinaryTree::IsEmpty() const{
110    return (root == nullptr);
111 }
112
```

```
113 TBinaryTree::~TBinaryTree() {
114     DeleteNode(root);
115     root = nullptr;
116 }
117
118
119
120 TBinaryTree::TreeNode::TreeNode() {
121     left = nullptr;
122     right = nullptr;
123 }
124
125 TBinaryTree::TreeNode::TreeNode(const TItem& data) {
126     left = nullptr;
127     right = nullptr;
128     this->data = data;
129 }
```

2.5. Консоль

```
sergey@svb:~/MAI/OOP/lab2$ valgrind --leak-check=full ./run
==4563== Memcheck, a memory error detector
==4563== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==4563== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==4563== Command: ./run
==4563==
Use 'help' or 'h' to get help.
Pentagon: created by default ctr
h


ins[ert]             insert <pentagon> in binary tree
p[rint]                print binary tree
f[ind]                 search in the binary tree of a pentagon with the <side>
del[ete]               delete in a binary tree a pentagon with the size <side>
q[uit]                 exit the program

p
Empty tree
================================================


ins
3
Pentagon: created by default ctr
```

```
ins
7
Pentagon: created by default ctr
ins
6
Pentagon: created by default ctr
p
================================================
Pentagon, side = 3
        Pentagon, side = 6
    Pentagon, side = 7
================================================


f
Input side of pentagon to search: 5
Pentagon: created with side
Pentagon deleted
Not found...
f
Input side of pentagon to search: 6
Pentagon: created with side
Pentagon deleted
Found: Pentagon, side = 6

del
Input side of pentagon to delete: 6
Pentagon: created with side
Pentagon deleted
Pentagon deleted
p
================================================
Pentagon, side = 3
    Pentagon, side = 7
================================================


del
Input side of pentagon to delete: 3
Pentagon: created with side
Pentagon deleted
Pentagon deleted
del
Input side of pentagon to delete: 7
Pentagon: created with side
Pentagon deleted
```

```
Pentagon deleted
p
Empty tree
================================================


q
Pentagon deleted
==4563==
==4563== HEAP SUMMARY:
==4563==     in use at exit: 72,704 bytes in 1 blocks
==4563==   total heap usage: 6 allocs, 5 frees, 74,848 bytes allocated
==4563==
==4563== LEAK SUMMARY:
==4563==    definitely lost: 0 bytes in 0 blocks
==4563==    indirectly lost: 0 bytes in 0 blocks
==4563==      possibly lost: 0 bytes in 0 blocks
==4563==    still reachable: 72,704 bytes in 1 blocks
==4563==         suppressed: 0 bytes in 0 blocks
==4563== Reachable blocks (those to which a pointer was found) are not shown.
==4563== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==4563==
==4563== For counts of detected and suppressed errors, rerun with: -v
==4563== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

2.6. Выводы

В данной работе нужно было реализовать свою структуру данных. В моем случае - бинарное дерево. В данной работе было реализовано бинарное дерево поиска. Некоторые структуры существуют в стандартных библиотеках шаблонов, но полезно самостоятельно их реализовать. Многие структуры очень часто применяются, например, в проектах.

Бинарные деревья имеют важное значение в практике программирования. Обычные деревья часто представляются с помощью специальных классов бинарных деревьев, т.к. их проще хранить и обрабатывать.

В бинарное дерево передаются фигуры по значению, а не по ссылке. Иногда это бывает полезно, если мы не хотим, чтобы в нашем хранилище объекты менялись тоже, когда они меняются вне хранилища.

Лабораторная работа №3

3.1. Цель работы

- Закрепление навыков работы с классами.
- Знакомство с умными указателями.

3.2. Задание

Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня, содержащей все три фигуры класса фигуры, согласно варианту задания (реализованную в ЛР 1).
Класы должны удовлетворять следующим правилам:

- Требования к классу фигуры аналогичны требованиям из лабораторной работы 1
- 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).

Программа должна позволять:

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

Фигуры: Пятиугольник, шестиугольник, восьмиугольник.
Контейнер: Бинарное дерево.

3.3. Описание

Умные указатели - это достаточно мощная техника. В отличие от обычных указателей они имеют еще более новую и усовершенствованную функциональность.

Существуют разные виды умных указателей. Приведем основные из них:

unique_ptr - это такой умный указатель, который сменил auto_ptr. В отличие от auto_ptr он запрещает копирование. Большим минусом является то, что объект этого класса теряет права владения ресурсом при копировании. (желательно не применять его по этим причинам).

shared_ptr - самый распространенный и самый широкоиспользуемый умный указатель. Он появился в C++11. Этот указатель уникален тем, что он работает с подсчетом ссылок. То есть, существует некая переменная, которая хранит количество указателей, которые ссылаются на объект. Если эта переменная станет равной нулю, то этот объект уничтожается. Счетчик инкрементируется при каждом выводе либо оператора копирования, либо оператора присванивания.

weak_ptr - умный указатель, у которого некоторые функции совпадают с функциями shared_ptr. Он не позволяет работать с ресурсом напрямую и не участвует в подсчете ссылок (хотя он предоставляет доступ к объекту, который принадлежит одному или нескольким экземплярам shared_ptr). Используется для устранения циклических ссылок std:shared_ptr, а также для отслеживания объекта, и преобразуется в std::shared_ptr для принятия временного владения.

Умные указатели определены в библиотеке <memory>.

3.4. Исходный код

	Внешний класс TBinaryTree	
public	TBinaryTree();	Конструктор класса.
	TBinaryTree(const TBinaryTree& orig);	Конструктор копирования.
	void Insert(std::shared_ptr<Figure> val);	Вставка нового ключа в бинарное дерево.
	bool IsEmpty() const;	Проверка, пусто ли дерево.
	std::shared_ptr<Figure> Find (std::shared_ptr<Figure> key);	Поиск в дереве заданной фигуры (объекта key).
	void DeleteNode(std::shared_ptr<Figure> key);	Удаление из дерева заданной фигуры (объекта key).
	std::shared_ptr<TreeNode> MinValueTreeNode (std::shared_ptr<TreeNode> node);	Поиск наименьшего значения в заданном поддереве.
	std::shared_ptr<TreeNode> deleteTreeNode(std::shared_ptr<TreeNode> _root, std::shared_ptr<Figure> key);	Перестроение дерева в процессе удаления заданной фигуры (используется рекурсивно).
	std::ostream& InOrderPrint(std::ostream& os, std::shared_ptr<TreeNode> node, size_t level) const;	Внутренняя рекурсивная функция для печати поддерева в симметрическом обходе.
	friend std::ostream& operator«(std::ostream& os, TBinaryTree& bintree);	Переопределенный оператор вывода в поток std::ostream.
	virtual ~TBinaryTree();	Деструктор класса.
private	Внутренний класс TreeNode	
public	TreeNode();	Конструктор класса.
	TreeNode(std::shared_ptr<Figure> data);	Конструктор копирования.

tbinary_tree.h

```
1  typedef Pentagon TItem;
2
3  class TBinaryTree {
4      private:
5          class TreeNode {
6              public:
7                  TreeNode();
8              TreeNode(std::shared_ptr<Figure> data);
9                  std::shared_ptr<TreeNode> left;
10                 std::shared_ptr<TreeNode> right;
11                 std::shared_ptr<Figure> data;
12         };
13
```

```
14          std::shared_ptr<TreeNode> root;
15      public:
16          TBinaryTree();
17          TBinaryTree(const TBinaryTree& orig);
18        void Insert(std::shared_ptr<Figure> val);
19          bool IsEmpty() const;
20        void DeleteNode(std::shared_ptr<Figure> key);
21        std::shared_ptr<TreeNode> MinValueTreeNode(std::
             shared_ptr<TreeNode> node);
22        std::shared_ptr<TreeNode> deleteTreeNode(std::
             shared_ptr<TreeNode> _root, std::shared_ptr<Figure>
              key);
23          std::ostream& InOrderPrint(std::ostream& os, std::
               shared_ptr<TreeNode> node, size_t level) const;
24          friend std::ostream& operator<<(std::ostream& os,
               const TBinaryTree &bintree);
25          virtual ~TBinaryTree();
26
27        std::shared_ptr<Figure> Find(std::shared_ptr<Figure>
             key);
28
29  };
```

figure.cpp

```
1  void Figure::print(std::ostream & os) const
2  {
3  }
4
5  bool Figure::TypedEquals(std::shared_ptr<Figure> other)
6  {
7     return (this->Square() == other->Square() && typeid(*this)
          == typeid(*other.get()));
8  }
9
10 bool Figure::SquareLess(std::shared_ptr<Figure> other)
11 {
12    return (this->Square() < other->Square());
13 }
14
15 bool Figure::SquareGreater(std::shared_ptr<Figure> other)
16 {
17    return this->Square() > other->Square();
18 }
```

3.5. Консоль

```
sergey@svb:~/MAI/OOP/lab3$./run
Use 'help' or 'h' to get help.
h


append or add        insert figure in binary tree
f[ind]                 find figure in binary tree
p[rint]                print binary tree
del[ete]               delete in a binary tree a figure with the size <side>
q[uit]                 exit the program

add
Which figure do you want to append? (pent/hex/oct[agon]): pent
Pentagon: enter side length: 5
add
Which figure do you want to append? (pent/hex/oct[agon]): oct
Octagon: enter side length: 3
add
Which figure do you want to append? (pent/hex/oct[agon]): hex
Hexagon: enter side length: 4
add
Which figure do you want to append? (pent/hex/oct[agon]): pent
Pentagon: enter side length: 6
p
================================================
    Octagon, side = 3
        Hexagon, side = 4
Pentagon, side = 5
    Pentagon, side = 6
================================================


f
Which figure do you want to find? (pent/hex/oct[agon]): hex
Hexagon: enter side length: 4
The figure was FOUND in the binary tree
del
Which figure do you want to delete? (pent/hex/oct[agon]): hex
Hexagon: enter side length: 3
p
================================================
    Octagon, side = 3
        Hexagon, side = 4
Pentagon, side = 5
```

```
    Pentagon, side = 6
================================================


del
Which figure do you want to delete? (pent/hex/oct[agon]): hex
Hexagon: enter side length: 4
p
================================================
    Octagon, side = 3
Pentagon, side = 5
    Pentagon, side = 6
================================================


q
```

3.6. Выводы

Умные указатели - важнейшая технология в C++ (появились на стандарте C++11). Это очень удобная и полезная вещь, поэтому даже новые стандарты языка C++ не обошли их. Благодаря им, можно избежать утечек памяти и обеспечить безопасное использование. Также отсутствуют проблемы с ними на уже удаленные объекты.

Лабораторная работа №4

4.1. Цель работы

- Знакомство с шаблонами классов.
- Построение шаблонов динамических структур данных.

4.2. Задание

Необходимо спроектировать и запрограммировать на языке C++ шаблон класса-контейнера первого уровня, содержащий все три фигуры класса фигуры, согласно варианту задания (реализованную в ЛР1).
Классы должны удовлетворять следующим правилам:

- Требования к классам фигуры аналогичны требованиям из лабораторной работы 1.
- 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 расположены в раздельных файлах: отдельно заголовки (.h), отдельно описание методов(.cpp).

Программа должна позволять:

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

Фигуры: Пятиугольник, шестиугольник, восьмиугольник.
Контейнер: Бинарное дерево.

4.3. Описание

Шаблоны (template) - средство языка C++. Они предназначены для кодирования обобщенных алгоритмов, без привязки к типам данных или значениям по умолчанию. Шаблоны позволяют создавать параметризованные классы и функции. Параметром может быть любой типа или значение одного из допустимых типов (целое число, указатель на любой объект с глобально доступным именем, перечисляемы тип, ссылка).

От того, что так устроен механизм шаблонов, их также называют усовершенствованными макросами (что это позволяет не беспокоиться о дублировании функций). И даже иногда рекомендуют вместо макросов и пустых указателей использовать шаблоны.

Из-за того, что я реализовал вложенные классы, приходится применять вложенные параметризованные типы (но с большой осторожностью!). Они настолько плохо читались, а также гораздо сильно расширяли код так, что тяжело дальше было работать с гигантским и плохочитаемым кодом,что мне пришлось разделять на два класса: TBinaryTree и TreeNode (но объекты и классы сами не изменились).

Хотел бы обратить внимание еще на одну особенность языка C++.
На строке 4-5:
template <class T>
using TreeNodePtr = std::shared_ptr<TreeNode<T> >; — на стандарте C++2011 это примерно работает, как typedef (такой инструмент еще применяется в следующих работах).

4.4. Исходный код

	template <class T> class TBinaryTree;	
public	TBinaryTree();	Конструктор класса.
	TBinaryTree(const TBinaryTree<T>& orig);	Конструктор копирования.
	void Insert(std::shared_ptr<T> val);	Вставка нового ключа в бинарное дерево.
	bool IsEmpty() const;	Проверка, пусто ли дерево.
	std::shared_ptr<T> Find(std::shared_ptr key);	Поиск в дереве заданной фигуры (объекта key).
	void Delete(std::shared_ptr<T> key);	Удаление из дерева заданной фигуры (объекта key).
private	TreeNodePtr<T> MinValueTreeNode(TreeNodePtr<T> node);	Внутренняя функция для поиска наименьшего значения в заданном поддереве.
	TreeNodePtr<T> deleteTreeNode(TreeNodePtr<T> _root, std::shared_ptr<T> key);	Внутренняя функция для перестроения дерева в процессе удаления заданной фигуры (используется рекурсивно).
	std::ostream& InOrderPrint(std::ostream& os, TreeNodePtr<T> node, size_t level) const;	Внутренняя рекурсивная функция для печати поддерева в симметрическом обходе.
public	template <class A> friend std::ostream& operator«(std::ostream& os, TBinaryTree<A>& bintree);	Переопределенный оператор вывода в поток std::ostream.
	virtual ~TBinaryTree();	Деструктор класса.
	template <class T> class TreeNode;	
public	TreeNode();	Конструктор класса.
	TreeNode(std::shared_ptr<T> data);	Конструктор копирования.

tbinary_tree.h

```
1  template <class T>
2  class TreeNode;
3
4  template <class T>
5  using TreeNodePtr = std::shared_ptr<TreeNode<T> >;
6
7  template<class T>
8  class TreeNode {
9  public:
10     TreeNode();
11     TreeNode(std::shared_ptr<T> data);
12     TreeNodePtr<T> left;
13     TreeNodePtr<T> right;
```

```
14      std::shared_ptr<T> data;
15  };
16
17  template <class T>
18  class TBinaryTree {
19      private:
20          TreeNodePtr<T> root;
21        TreeNodePtr<T> MinValueTreeNode(TreeNodePtr<T> node);
22        TreeNodePtr<T> deleteTreeNode(TreeNodePtr<T> _root, std
              ::shared_ptr<T> key);
23        std::ostream& InOrderPrint(std::ostream& os,
              TreeNodePtr<T> node, size_t level) const;
24      public:
25          TBinaryTree();
26          TBinaryTree(const TBinaryTree<T>& orig);
27        void Insert(std::shared_ptr<T> figure);
28          bool IsEmpty() const;
29          void Delete(std::shared_ptr<T> key);
30        std::shared_ptr<T> Find(std::shared_ptr<T> key);
31
32          template <class A> friend std::ostream& operator<<(
                std::ostream& os, TBinaryTree<A> & bintree);
33          virtual ~TBinaryTree();
34  };
35
36  template <class T>
37  TBinaryTree<T>::TBinaryTree() {
38      root = nullptr;
39  }
40
41  template <class T>
42  void TBinaryTree<T>::Insert(std::shared_ptr<T> val) {
43      if (root == nullptr)
44          root = TreeNodePtr<T>(new TreeNode<T>(val));
45      else {
46          TreeNodePtr<T> curNode = root;
47          TreeNodePtr<T> parNode = nullptr;
48          while (curNode != nullptr) {
49              parNode = curNode;
50              if (val->SquareLess(curNode->data))
51                  curNode = curNode->left;
52              else
53                  curNode = curNode->right;
54          }
55          if (val->SquareLess(parNode->data))
56              parNode->left = TreeNodePtr<T>(new TreeNode<T>(val))
```

```
            ;
57          else
58              parNode->right = TreeNodePtr<T>(new TreeNode<T>(val)
                    );
59      }
60  }
61
62
63  template <class T>
64  std::shared_ptr<T> TBinaryTree<T>::Find(std::shared_ptr<T>
        key)
65  {
66      if (root == nullptr)
67          return nullptr;
68
69      TreeNodePtr<T> curNode = root;
70      TreeNodePtr<T> parNode = nullptr;
71      while (curNode != nullptr) {
72          parNode = curNode;
73          if (key->TypedEquals(curNode->data))
74              return curNode->data;
75          if (key->SquareLess(curNode->data))
76              curNode = curNode->left;
77          else
78              curNode = curNode->right;
79      }
80      return nullptr;
81  }
82
83
84  template<class T>
85  TreeNodePtr<T> TBinaryTree<T>::MinValueTreeNode(TreeNodePtr<T
        > node)
86  {
87      TreeNodePtr<T> current = node;
88
89      while (current->left != nullptr)
90          current = current->left;
91
92      return current;
93  }
94
95  template<class T>
96  TreeNodePtr<T> TBinaryTree<T>::deleteTreeNode(TreeNodePtr<T>
        _root, std::shared_ptr<T> key)
97  {
```

```
98     if (_root == nullptr) return _root;
99     if (key->TypedEquals(_root->data)) {
100        if (_root->left == nullptr) {
101           TreeNodePtr<T> temp = _root->right;
102           return temp;
103        }
104        else if (_root->right == nullptr) {
105           TreeNodePtr<T> temp = _root->left;
106           return temp;
107        }
108
109        TreeNodePtr<T> temp = MinValueTreeNode(_root->right);
110        _root->data = temp->data;
111        _root->right = deleteTreeNode(_root->right, temp->data)
              ;
112    }
113    else if (key->SquareLess(_root->data))
114        _root->left = deleteTreeNode(_root->left, key);
115    else
116        _root->right = deleteTreeNode(_root->right, key);
117
118    return _root;
119 }
120
121 template<class T>
122 std::ostream & TBinaryTree<T>::InOrderPrint(std::ostream & os
       , TreeNodePtr<T> node, size_t level) const
123 {
124    if (node != nullptr) {
125        InOrderPrint(os, node->left, level + 1);
126        for (size_t i = 0; i < level; i++)
127           os << '\t';
128        node->data->print(os);
129        InOrderPrint(os, node->right, level + 1);
130    }
131    return os;
132 }
133
134 template <class T>
135 void TBinaryTree<T>::Delete(std::shared_ptr<T> key)
136 {
137    if (key == nullptr)
138        return;
139    root = deleteTreeNode(root, key);
140 }
141
```

```
142 template <class A>
143 std::ostream& operator<<(std::ostream& os, TBinaryTree<A> &
       bintree) {
144    if (bintree.IsEmpty())
145       os << "Empty tree\n";
146    else
147       os << "================================================\
          n";
148    bintree.InOrderPrint(os, bintree.root, 0);
149    os << "================================================\n\n
       ";
150    return os;
151 }
152
153 template <class T>
154 bool TBinaryTree<T>::IsEmpty() const {
155    return (root == nullptr);
156 }
157
158 template <class T>
159 TBinaryTree<T>::TBinaryTree(const TBinaryTree<T> & orig) {
160    std::cout << "not impl\n";
161 }
162
163 template <class T>
164 TBinaryTree<T>::~TBinaryTree() {
165    root = nullptr;
166 }
167
168 template <class T>
169 TreeNode<T>::TreeNode() {
170    left = nullptr;
171    right = nullptr;
172 }
173
174 template<class T>
175 TreeNode<T>::TreeNode(std::shared_ptr<T> data)
176 {
177    left = nullptr;
178    right = nullptr;
179    this->data = data;
180 }
```

4.5. Консоль

```
sergey@svb:~/MAI/OOP/lab4$./run
Use 'help' or 'h' to get help.
h


append or add        insert figure in binary tree
f[ind]                 find figure in binary tree
p[rint]                print binary tree
del[ete]               delete in a binary tree a figure with the size <side>
q[uit]                 exit the program

p
Empty tree
================================================


add
Which figure do you want to append? (pent/hex/oct[agon]): pent
Pentagon: enter side length: 5
add
Which figure do you want to append? (pent/hex/oct[agon]): pent
Pentagon: enter side length: 2
add
Which figure do you want to append? (pent/hex/oct[agon]): hex
Hexagon: enter side length: 4
add
Which figure do you want to append? (pent/hex/oct[agon]): oct
Octagon: enter side length: 7
p
================================================
    Pentagon, side = 2
        Hexagon, side = 4
Pentagon, side = 5
    Octagon, side = 7
================================================


del
Which figure do you want to delete? (pent/hex/oct[agon]): hex
Hexagon: enter side length: 4
p
================================================
    Pentagon, side = 2
Pentagon, side = 5
```

```
    Octagon, side = 7
================================================


f
Which figure do you want to find? (pent/hex/oct[agon]): pent
Pentagon: enter side length: 1
The figure was NOT FOUND in the binary tree
f
Which figure do you want to find? (pent/hex/oct[agon]): pent
Pentagon: enter side length: 5
The figure was FOUND in the binary tree
q
```

4.6. Выводы

В данной работе познакомился с важнейшим инструментом - шаблоны. Они часто применяются в проектах, приложениях и т.д, а также нередко используются для контейнерных классов, поскольку идея параметров типа хорошо согласуется с необходимостью создания общего плана хранения для разных типов. Шаблоны упрощают программирование и делают программы надежнее, поскольку они предоставляют программисту возможность повторного использованного протестированного кода, не занимаясь его копированием вручную. А также такие инструменты сумели решить проблему «одинаковых классов».

Лабораторная работа №5

5.1. Цель работы

- Знакомство навыков работы с шаблонами классов.
- Построение итераторов для динамических структур данных.

5.2. Задание

Используя структуры данных, разработанные для предыдущей лабораторной работы (ЛР4) 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.

Итератор должен позволять использовать структуру данных в операторах типа for. Например: for (auto i : stack) std::cout « *i « std::endl;

Программа должна позволять:

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

Фигуры: Пятиугольник, шестиугольник, восьмиугольник.
Контейнер: Бинарное дерево.

5.3. Описание

Для доступа к элементам некоторого множества элементов используют специальные объекты, называемые итераторами. Итератор предоставляет доступ к отдельным элементам. Все контейнеры STL предоставляют итераторы, чтобы алгоритмы могли получить доступ к своим элементам стандартным способом, независимо от типа контейнера, в котором сохранены элементы. В контейнерных типах stl они доступны через методы класса (например, begin() в шаблоне класса vector). Функциональные возможности указателей и итераторов близки. Их можно назначать и копировать. В данной работе для использования итератора бинарного дерева применяется алгоритм называемый DFS(Depth First Search).

Существует пять категорий итераторов (в порядке возрастания силы):

- **Итератор ввода (input iterator).** Итератор ввода X может выполнить итерацию последовательности с помощью оператора ++ и прочитать элемент любое количество раз с помощью оператора *. Можно сравнить итераторы ввода с помощью операторов ++ и !=. Используется потоками ввода.

- **Итератор вывода (output iterator).** Итератор вывода X может выполнить итерацию последовательности с помощью оператора ++ и один раз записать элемент с помощью оператора *. Используется потоками вывода.
- **Однонаправленный (forward iterator).** Однонаправленный итератор X может выполнять итерацию последовательности с помощью оператора ++ и прочитать любой элемент или записать неконстантные элементы любое количество раз с помощью оператора *. Получение доступа к членам элементов осуществляется с помощью оператора -> и однонаправленные итераторы можно сравнить с помощью операторов == и !=. Применяется для прохода по элементам в одном направлении.
- **Двунаправленный (bidirectional iterator)** Способен пройтись по элементам в любом направлении. Получить доступ к членам элементов и сравнить двунаправленные итераторы можно так же, как и однонаправленные итераторы. Такие итераторы реализованы в некоторых контейнерных типах stl (например, list, map).
- **Произвольный доступ(random access).** Через них можно иметь доступ к любому элементу. Такие итераторы реализованы в некоторых контейнерных типах stl (например, vector, array)

5.4. Исходный код

common.h

```
1 template <class T>
2 class TreeNode;
3
4 template <class T>
5 class TBinaryTree;
6
7 template <class T>
8 using TreeNodePtr = std::shared_ptr<TreeNode<T> >;
```

Классы контейнера первого уровня и фигуры остаются неизменными. Только класс итератора дерева является дружественным к этим классам.
Рассмотрим работу с итератором бинарного дерева.

iterator.h

```
1 template <class T>
2 class TBinaryTreeIterator {
3 public:
4    TBinaryTreeIterator(TreeNodePtr<T> n) {
5       treeNodePtr = n;
6       if (!treeNodePtr)
7          return;
8
```

```
 9          while (treeNodePtr->left)
10              treeNodePtr = treeNodePtr->left;
11      }
12
13      std::shared_ptr<T> operator * () {
14          return treeNodePtr->GetPtr();
15      }
16      std::shared_ptr<T> operator -> () {
17          return treeNodePtr->GetPtr();
18      }
19      void operator ++ () {
20          TreeNodePtr<T> res = treeNodePtr;
21          if (res->right) {
22              res = res->right;
23              while (res->left)
24                  res = res->left;
25          }
26          else
27          {
28              while (true) {
29                  if (!res->parent) {
30                      res = nullptr;
31                      break;
32                  }
33                  if (res->parent->left == res) {
34                      res = res->parent;
35                      break;
36                  }
37                  res = res->parent;
38              }
39          }
40          treeNodePtr = res;
41      }
42
43      TBinaryTreeIterator operator ++ (int) {
44          TBinaryTreeIterator iter(*this);
45          ++(*this);
46          return iter;
47      }
48      bool operator == (TBinaryTreeIterator const& i) {
49          return treeNodePtr == i.treeNodePtr;
50      }
51      bool operator != (TBinaryTreeIterator const& i) {
52          return !(*this == i);
53      }
54  private:
```

```
55      TreeNodePtr<T> treeNodePtr;
56  };
```

5.5. Консоль

```
sergey@svb:~/MAI/OOP/lab5$ ./run
Use 'help' or 'h' to get help.
h


add                     insert figure in binary tree
f[ind]                  find figure in binary tree
p[rint]                 print binary tree
it[erate]               print iterator of binary tree
del[ete]                delete in a binary tree a figure with the size <side>
q[uit]                  exit the program

add
Which figure do you want to append? (pent/hex/oct[agon]): pent
Pentagon: enter side length: 5
add
Which figure do you want to append? (pent/hex/oct[agon]): hex
Hexagon: enter side length: 3
add
Which figure do you want to append? (pent/hex/oct[agon]): oct
Octagon: enter side length: 4
add
Which figure do you want to append? (pent/hex/oct[agon]): pent
Pentagon: enter side length: 1
add
Which figure do you want to append? (pent/hex/oct[agon]): pent
Pentagon: enter side length: 2
add
Which figure do you want to append? (pent/hex/oct[agon]): hex
Hexagon: enter side length: 7
add
Which figure do you want to append? (pent/hex/oct[agon]): oct
Octagon: enter side length: 6
add
Which figure do you want to append? (pent/hex/oct[agon]): oct
Octagon: enter side length: 9
add
Which figure do you want to append? (pent/hex/oct[agon]): hex
Hexagon: enter side length: 8
add
```

```
Which figure do you want to append? (pent/hex/oct[agon]): pent
Pentagon: enter side length: 12
p
================================================
Pentagon, side = 1
Pentagon, side = 2
Hexagon, side = 3
Octagon, side = 4
Pentagon, side = 5
Octagon, side = 6
Hexagon, side = 7
Hexagon, side = 8
Octagon, side = 9
Pentagon, side = 12
================================================


it
Pentagon, side = 1

Pentagon, side = 2

Hexagon, side = 3

Octagon, side = 4

Pentagon, side = 5

Octagon, side = 6

Hexagon, side = 7

Hexagon, side = 8

Octagon, side = 9

Pentagon, side = 12

del
Which figure do you want to delete? (pent/hex/oct[agon]): pent
Pentagon: enter side length: 5
it
Pentagon, side = 1

Pentagon, side = 2
```

```
Hexagon, side = 3

Octagon, side = 4

Octagon, side = 6

Hexagon, side = 7

Hexagon, side = 8

Octagon, side = 9

Pentagon, side = 12

q
```

5.6. Выводы

В данной работе я применил итератор для бинарного дерева. Честно говоря, я встречал итераторы для списка, для очереди и т.д. много раз, но только не для бинарного дерева (и тем более для n-дерева!!). Такая ситуация не удивляет меня, поскольку доступ к элементам моего контейнера сложнее осуществлять, чем остальным структурам данных, кроме n-дерева. Тем не менее полезно уметь применять его для каждого контейнера, ведь благодаря ему мы можем сослаться не на весь контейнер, а на какое-то место в этом наборе элементов (каждый контейнер имеет свой набор элементов). Это позволяет нам именно с этого места или в нём осуществлять какие-либо операции (например, изменить элемент, вставить или удалить элемент). Созданные итераторы я активно применял в следующих лабораторных работах.

Лабораторная работа №6

6.1. Цель работы

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

6.2. Задание

Используя структуры данных, разработанные для предыдущей лабораторной работы (ЛР5) 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 объемы 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).

Для вызова аллокатора должны быть переопределены оператор **new** и **delete** у классов-фигур.

Программа должна позволять:

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

Фигуры: Пятиугольник, шестиугольник, восьмиугольник.
Контейнер 1-го уровня: Бинарное дерево.
Контейнер 2-го уровня: Связанный список.

6.3. Описание

Каждый контейнер имеет определенный для него аллокатор (allocator). Аллокатор управляет выделением памяти для контейнера. Аллокатором по умолчанию является объект класса allocator, но можно определять свои собственные аллокаторы, если это необходимо для специализированных приложений. Аллокатор позволяет выделять и освобождать память в требуемых количествах определённым образом. Из стандартной библиотеки можно взять пример реализации аллокатора. Он просто использует new и delete, которые обычно обращаются к системным вызовам malloc и free.

В данной работе мне необходимо было реализовать аллокаторы памяти для списка. Я выбрал односвязный список, поскольку для меня удобнее было бы

работать с ним так и в создании, так и в удалении. Для емкости бинарного дерева (MAX_TREE_CAPACITY) я выбрал 100.

Также стоит рассмотреть эти две строки.

template <class T> TAllocationBlock
TreeNode<T>::allocator(sizeof(TreeNode<T>), MAX_TREE_CAPACITY);

Это инициализация, то есть присваивание начального значения статическому полю класса TreeNode. Она происходит только один раз, так как это единое статическое поле для всех объектов класса.

6.4. Исходный код

	Класс TAllocationBlock	
public	TAllocationBlock(size_t size, size_t count);	Конструктор класса.
	void *Allocate();	Выделение памяти.
	void Deallocate(void *ptr);	Освобождение памяти.
	bool Empty();	Проверка, пуст ли аллокатор.
	size_t Size();	Получение количества выделенных блоков.
	virtual ~TAllocationBlock();	Деструктор класса.

Описание класса контейнера первого уровня и фигуры остается неизмененным.

TAllocationBlock.h

```
1  typedef unsigned char Byte;
2  typedef void * VoidPtr;
3
4  template<class T>
5  class TList;
6
7  class TAllocationBlock
8  {
9  public:
10    TAllocationBlock(size_t size, size_t count);
11    void *Allocate();
12    void Deallocate(void *ptr);
13    bool Empty();
14    size_t Size();
15
16    virtual ~TAllocationBlock();
17
18 private:
19    Byte *_used_blocks;
20    TList<VoidPtr> _free_blocks;
```

```
21 };
```

TAllocationBlock.cpp

```
1  #include "TAllocationBlock.h"
2  #include <iostream>
3
4  TAllocationBlock::TAllocationBlock(size_t size, size_t count)
5  {
6     _used_blocks = (Byte *)malloc(size * count);
7
8      void * ptr;
9     for (size_t i = 0; i < count; ++i) {
10       ptr = _used_blocks + i * size;
11       _free_blocks.Push(ptr);
12    }
13 }
14
15 void *TAllocationBlock::Allocate()
16 {
17    if (!_free_blocks.IsEmpty()) {
18       void * res = _free_blocks.Top();
19       _free_blocks.Pop();
20       return res;
21    }
22    else {
23       throw std::bad_alloc();
24    }
25 }
26
27 void TAllocationBlock::Deallocate(void *ptr)
28 {
29    _free_blocks.Push(ptr);
30 }
31
32 bool TAllocationBlock::Empty()
33 {
34    return _free_blocks.IsEmpty();
35 }
36
37 size_t TAllocationBlock::Size()
38 {
39    return _free_blocks.GetLength();
40 }
41
42 TAllocationBlock::~TAllocationBlock()
43 {
```

```
44      free(_used_blocks);
45      std::cout << "allocator finished it\'s work\n";
46  }
```

6.5. Консоль

```
sergey@svb:~/MAI/OOP/lab6$ valgrind --leak-check=full ./run
==4251== Memcheck, a memory error detector
==4251== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==4251== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==4251== Command: ./run
==4251==
Use 'help' or 'h' to get help.
add
Which figure do you want to append? (pent/hex/oct[agon]): pent
Pentagon: enter side length: 4
p
================================================
Pentagon, side = 4
================================================


f
Which figure do you want to find? (pent/hex/oct[agon]): pent
Pentagon: enter side length: 4
The figure was FOUND in the binary tree
del
Which figure do you want to delete? (pent/hex/oct[agon]): pent
Pentagon: enter side length: 4
q
allocator finished it's work
==4251==
==4251== HEAP SUMMARY:
==4251==     in use at exit: 72,704 bytes in 1 blocks
==4251==   total heap usage: 213 allocs, 212 frees, 84,528 bytes allocated
==4251==
==4251== LEAK SUMMARY:
==4251==    definitely lost: 0 bytes in 0 blocks
==4251==    indirectly lost: 0 bytes in 0 blocks
==4251==      possibly lost: 0 bytes in 0 blocks
==4251==    still reachable: 72,704 bytes in 1 blocks
==4251==         suppressed: 0 bytes in 0 blocks
==4251== Reachable blocks (those to which a pointer was found) are not shown.
==4251== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==4251==
```

```
==4251== For counts of detected and suppressed errors, rerun with: -v
==4251== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
sergey@svb:~/MAI/OOP/lab6$ valgrind --leak-check=full ./run
==2818== Memcheck, a memory error detector
==2818== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==2818== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==2818== Command: ./run
==2818==
Use 'help' or 'h' to get help.
p
Empty tree
================================================


q
allocator finished it's work
==2818==
==2818== HEAP SUMMARY:
==2818==     in use at exit: 72,704 bytes in 1 blocks
==2818==   total heap usage: 204 allocs, 203 frees, 84,352 bytes allocated
==2818==
==2818== LEAK SUMMARY:
==2818==    definitely lost: 0 bytes in 0 blocks
==2818==    indirectly lost: 0 bytes in 0 blocks
==2818==      possibly lost: 0 bytes in 0 blocks
==2818==    still reachable: 72,704 bytes in 1 blocks
==2818==         suppressed: 0 bytes in 0 blocks
==2818== Reachable blocks (those to which a pointer was found) are not shown.
==2818== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==2818==
==2818== For counts of detected and suppressed errors, rerun with: -v
==2818== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

6.6. Выводы

Аллокаторы бывают полезными для специализированных приложений. Они позволяют обобщенным контейнерам отделить управление памятью от фактических данных. Программисты должны учитывать последствия динамического выделения памяти и дважды обдумать использование функции malloc или оператора new. Проекты, где управление и распределение памяти не продумано надлежащим образом, часто страдают от случайных сбоев после длительной игровой сессии из-за нехватки памяти (которые, кстати, практически невозможно воспроизвести) и стоят сотни часов работы программистов, пытающихся освободить память и реорганизовать её выделение.

Лабораторная работа №7

7.1. Цель работы

- Создание сложных динамических структур данных.
- Закрепление принципа OCP.

7.2. Задание

Необходимо реализовать динамическую структуру данных - «Хранилище объектов» и алгоритм работы с ней. «Хранилище объектов» представляет собой контейнер: связанный список. Каждым элементом контейнера является динамическая структура данных: бинарное дерево.Таким образом у нас получается контейнер в контейнере.
При этом должно выполняться правило, что количество объектов в контейнере второго небольше 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 удалиться.
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:
  - По типу (например, все квадраты).
  - По площади (например, все объекты с площадью меньше чем заданная).

Фигуры: Пятиугольник, шестиугольник, восьмиугольник.
Контейнер 1-го уровня: Бинарное дерево.
Контейнер 2-го уровня: Связанный список.

7.3. Описание

Контейнер — это объект, который может содержать в себе другие объекты. Существует несколько разных типов контейнеров. Например, класс vector определяет динамический массив, deque создает двунаправленную очередь, a list представляет связный список. Эти контейнеры называются последовательными контейнерами (sequence containers), потому что в терминологии STL последовательность — это линейный список.

В этой лабораторной мне требовалось создать контейнер на основе двух контейнеров, вложенных друг в друга.

IRemoveCriteria<T> — это интерфейс, а по сути функтор, то есть класс в котором переопределен оператор() — оператор вызова функции, подобные классы применяются в STL. Также у меня определены два класса, реализующие этот интерфейс: **RemoveCriteriaByMaxSquare** и **RemoveCriteriaByFigureType**, которые будут передаваться в объект **Storage**, как параметр функции удаления фигур. Фактически это предикаты, которые определяют, удовлетворяет ли фигура критерию. Зачем вообще использовать эти классы-функторы, а не просто указатели на функции? Потому что нужно задать дополнительные параметры для критериев, например, максимальную площадь фигуры. Эти параметры как раз хранятся в классах, названных выше

7.4. Исходный код

	template <class T> class IRemoveCriteria;	
public	virtual bool operator()(std::shared_ptr value) = 0;	Абстрактный метод интерфейса, который будет перегружен.
	Класс RemoveCriteriaByMaxSquare : public IRemoveCriteria<Figure>	
public	RemoveCriteriaByMaxSquare(double value);	Конструктор принимающий параметр максимальной площади.
	bool operator()(std::shared_ptr value);	Перегруженный оператор вызова функции.
	Класс RemoveCriteriaByFigureType : public IRemoveCriteria	
public	RemoveCriteriaByFigureType(const char * value);	Конструктор принимающий параметр тип фигуры.
	bool operator()(std::shared_ptr value);	Перегруженный оператор вызова функции.
	~RemoveCriteriaByFigureType();	Деструктор класса.
	template <class T> class TStorage;	
public	TStorage();	Конструктор класса.
	~TStorage();	Деструктор класса.
	void Insert(std::shared_ptr<T> item);	Вставка элемента в хранилище объектов.
	void DeleteByCriteria(IRemoveCriteria<T> &crit);	Удаление элементов по критериям.
	friend std::ostream & operator«(std::ostream & os, TStorage<T> & stor);	Переопределенный оператор вывода в поток std::ostream.

crit.h

```
1  template <class T>
2  class IRemoveCriteria
3  {
4  public:
5     virtual bool operator()(std::shared_ptr<T> value) = 0;
6  };
7
8
9  class RemoveCriteriaByMaxSquare : public IRemoveCriteria<
       Figure>
10 {
11 public:
12    RemoveCriteriaByMaxSquare(double value)
13    {
14       _MaxSquareValue = value;
15    }
16
17    bool operator()(std::shared_ptr<Figure> value) override
18    {
19       return value->Square() < _MaxSquareValue;
20    }
21
22 private:
23    double _MaxSquareValue;
24 };
25
26
27 class RemoveCriteriaByFigureType : public IRemoveCriteria<
       Figure>
28 {
29 public:
30    RemoveCriteriaByFigureType(const char * value)
31    {
32       _TypeName = new char[strlen(value) + 1];
33       strcpy(_TypeName, value);
34    }
35
36    bool operator()(std::shared_ptr<Figure> value) override
37    {
38       return strcmp(typeid(*value).name()+6, _TypeName)==0;
39    }
40
41    ~RemoveCriteriaByFigureType()
42    {
43       delete _TypeName;
```

```
44      }
45
46  private:
47      char * _TypeName;
48  };
```

storage.h

```
1
2   template <class T>
3   class TStorage
4   {
5   private:
6       TList<TBinaryTree<T>> storage;
7   public:
8       TStorage() {
9       }
10
11      ~TStorage() {
12      }
13
14      void Insert(std::shared_ptr<T> item) {
15          if (storage.IsEmpty()) {
16              TBinaryTree<T> tree;
17              tree.Insert(item);
18              storage.Push(tree);
19          } else {
20              TBinaryTree<T> & top = storage.Top();
21              if (top.GetCount() < 5) {
22                  top.Insert(item);
23              } else {
24                  TBinaryTree<T> tree;
25                  tree.Insert(item);
26                  storage.Push(tree);
27              }
28          }
29          TBinaryTree<T> & top = storage.Top();
30          std::cout << "Object was added with index " << storage.
                GetLength() - 1 << "." << top.GetCount() - 1 << "\n"
                ;
31      }
32
33      void DeleteByCriteria(IRemoveCriteria<T> &crit) {
34          auto it_list = storage.begin();
35          while(it_list != storage.end()) {
36              TList< std::shared_ptr<T> > figuresToDelete;
37              for (auto it_tree = (*it_list)->begin(); it_tree !=
```

```
            (*it_list)->end(); it_tree++) {
38              if (crit(*it_tree))
39                  figuresToDelete.Push(*it_tree);
40          }
41          bool needToDeleteTree = (figuresToDelete.GetLength()
                == (*it_list)->GetCount());
42          for (auto it_figlist = figuresToDelete.begin();
               it_figlist != figuresToDelete.end(); it_figlist
               ++) {
43              (*it_list)->Delete(**it_figlist);
44          }
45          if (needToDeleteTree) {
46              auto it_tmp = it_list;
47              it_list++;
48              storage.Delete(it_tmp);
49          } else
50              it_list++;
51      }
52  }
53
54  friend std::ostream & operator<<(std::ostream & os,
       TStorage<T>& stor) {
55      size_t i = stor.storage.GetLength() -1;
56      if (i == -1) {
57          std::cout << "================ EMPTY STORAGE
               ================ ";
58      }
59      for (auto it_list = stor.storage.begin(); it_list !=
           stor.storage.end(); it_list++) {
60          std::cout << "=================== TREE " << i-- << "
                ==================== " << std::endl;
61          std::cout << **it_list;
62      }
63      return os;
64  }
65 };
```

7.5. Консоль

```
sergey@svb:~/MAI/OOP/lab7$ ./run
Use 'help' or 'h' to get help.
add
Which figure do you want to append? (pent/hex/oct[agon]): pent
Pentagon: enter side length: 4
Object was added with index 0.0
```

```
add
Which figure do you want to append? (pent/hex/oct[agon]): oct
Octagon: enter side length: 3
Object was added with index 0.1
add
Which figure do you want to append? (pent/hex/oct[agon]): pent
Pentagon: enter side length: 7
Object was added with index 0.2
add
Which figure do you want to append? (pent/hex/oct[agon]): hex
Hexagon: enter side length: 3
Object was added with index 0.3
add
Which figure do you want to append? (pent/hex/oct[agon]): pent
Pentagon: enter side length: 5
Object was added with index 0.4
add
Which figure do you want to append? (pent/hex/oct[agon]): oct
Octagon: enter side length: 6
Object was added with index 1.0
p
================== TREE 1 ==================
Octagon         side = 6        square = 173.82
================== TREE 0 ==================
Hexagon         side = 3        square = 23.38
Pentagon        side = 4        square = 27.53
Pentagon        side = 5        square = 43.01
Octagon         side = 3        square = 43.46
Pentagon        side = 7        square = 84.30

del
Which criterion do you want to use? (a[rea]/t[ype]): a
Enter the square threshold under which figures will be deleted: 50
p
================== TREE 1 ==================
Octagon         side = 6        square = 173.82
================== TREE 0 ==================
Pentagon        side = 7        square = 84.30

del
Which criterion do you want to use? (a[rea]/t[ype]): t
Which type of figure do you want to delete? (pent/hex/oct[agon]): oct
p
================== TREE 0 ==================
Pentagon        side = 7        square = 84.30
```

q

7.6. Выводы

В этой лабораторной работе мне удалось создать контейнер на основе двух контейнеров, вложенных друг в друга. Также познакомился с принципом ООП - ОСП (Open/Closed Principle). Он помогает делать базовые алгоритмы неизменными (независящими от вносимых изменений). Благодаря принципу, уменьшается количество проверок, которые нужно сделать для новых тестов. ОСП устанавливает такое положение: «программные сущности (классы, модули, функции и т.д.) должны быть открыты для расширения, но закрыты для изменения».

Лабораторная работа №8

8.1. Цель работы

- Знакомство с параллельным программированием в C++.

8.2. Задание

Используя структуры данных, разработанных для лабораторной работы 6 (контейнер первого уровня и класс-фигуры) разработать алгоритм быстрой сортировки для класса-контейнера.
Необходимо разработать два вида алгоритма:

- Обычный, без параллельных вызовов.
- 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

Программа должна позволять:

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

Фигуры: Пятиугольник, шестиугольник, восьмиугольник.
Контейнер: Бинарное дерево.

8.3. Описание

Поскольку в ранее сделанных работах я работал с бинарным деревом поиска, а как мы знаем, что его ключи расположены так, что значения этих ключей находятся в отсортированном порядке. Поэтому в качестве контейнера, для которого применяется быстрая сортировка, взял связанный список, чтобы убедиться в том,

что в параллельном программировании сортировка верно работала. В работе применяются некоторые инструменты, такие как, например, std::future. Это такой метод, который позволяет получить результат работы функции, которую мы передаем. Также существует такой шаблон класса std::promise, который предоставляет средство для хранения значения или исключения. Можно вкратце сказать, что std::promise является «производителем» асинхронной операции, а std::future - «потребителем» асинхронной операции.

8.4. Исходный код

tlist.h

```
1  template <class T>
2  class TListItem;
3
4  template <class T>
5  using TListItemPtr = std::shared_ptr<TListItem<T> >;
6
7  template <class T> class TList {
8     public:
9        TList()
10       {
11          head = nullptr;
12       }
13
14
15       void Push(T* item)
16       {
17          TListItemPtr<T> other(new TListItem<T>(item));
18          other->SetNext(head);
19          head = other;
20       }
21
22       void Push(std::shared_ptr<T> item)
23       {
24          TListItemPtr<T> other(new TListItem<T>(item));
25          other->SetNext(head);
26          head = other;
27       }
28
29       bool IsEmpty() const
30       {
31          return head == nullptr;
32       }
33
34       size_t Size()
```

```
35          {
36              size_t result = 0;
37              for (auto i : *this)
38                  result++;
39              return result;
40          }
41
42          TListIterator<T> begin()
43          {
44              return TListIterator<T>(head);
45          }
46
47          TListIterator<T> end()
48          {
49              return TListIterator<T>(nullptr);
50          }
51
52          void Sort()
53          {
54              if (Size() > 1) {
55                  std::shared_ptr<T> middle = Pop();
56                  TList<T> left, right;
57                  while (!IsEmpty()) {
58                      std::shared_ptr<T> item = Pop();
59                      if (!item->SquareLess(middle)) {
60                          left.Push(item);
61                      } else {
62                          right.Push(item);
63                      }
64                  }
65                  left.Sort();
66                  right.Sort();
67                  while (!left.IsEmpty()) {
68                      Push(left.PopLast());
69                  }
70                  Push(middle);
71                  while (!right.IsEmpty()) {
72                      Push(right.PopLast());
73                  }
74              }
75          }
76
77          void SortParallel()
78          {
79              if (Size() > 1) {
80                  std::shared_ptr<T> middle = PopLast();
```

```
81              TList<T> left, right;
82              while (!IsEmpty()) {
83                  std::shared_ptr<T> item = PopLast();
84                  if (!item->SquareLess(middle)) {
85                      left.Push(item);
86                  } else {
87                      right.Push(item);
88                  }
89              }
90              std::future<void> left_res = left.BackgroundSort
                    ();
91              std::future<void> right_res = right.
                    BackgroundSort();
92
93
94              left_res.get();
95
96
97              while (!left.IsEmpty()) {
98                  Push(left.PopLast());
99              }
100             Push(middle);
101             right_res.get();
102             while (!right.IsEmpty()) {
103                 Push(right.PopLast());
104             }
105         }
106     }
107
108     std::shared_ptr<T> Pop()
109     {
110         std::shared_ptr<T> result;
111         if (head != nullptr) {
112             result = head->GetValue();
113             head = head->GetNext();
114         }
115         return result;
116     }
117
118     std::shared_ptr<T> PopLast()
119     {
120         std::shared_ptr<T> result;
121         if (head != nullptr) {
122             TListItemPtr <T> element = head;
123             TListItemPtr <T> prev = nullptr;
124             while (element->GetNext() != nullptr) {
```

```
125                 prev = element;
126                 element = element->GetNext();
127             }
128             if (prev != nullptr) {
129                 prev->SetNext(nullptr);
130                 result = element->GetValue();
131             } else {
132                 result = element->GetValue();
133                 head = nullptr;
134             }
135         }
136         return result;
137     }
138 
139     void Delete(std::shared_ptr<T> key)
140     {
141         bool found = false;
142         if (head != nullptr) {
143             TListItemPtr <T> element = head;
144             TListItemPtr <T> prev = nullptr;
145             while (element != nullptr) {
146                 if (element->GetValue()->TypedEquals(key)) {
147                     found = true;
148                     break;
149                 }
150                 prev = element;
151                 element = element->GetNext();
152             }
153             if (found) {
154                 if (prev != nullptr) {
155                     prev->SetNext(element->GetNext());
156                 }
157                 else {
158                     head = element->GetNext();
159                 }
160             }
161         }
162     }
163 
164     template <class A>
165     friend std::ostream& operator<<(std::ostream& os, const
            TList<A>& list)
166     {
167         TListItemPtr<A> item = list.head;
168         if (list.IsEmpty())
169             os << "List is empty\n";
```

```
170             while (item != nullptr) {
171                 os << *item;
172                 item = item->GetNext();
173             }
174             return os;
175         }
176         virtual ~TList() {/*nothing need to be done,
                shared_ptrs make all the work!*/}
177     private:
178         std::future<void> BackgroundSort()
179         {
180             std::packaged_task<void(void) > task(std::bind(std::
                    mem_fn(&TList<T>::SortParallel), this));
181             std::future<void> res(task.get_future());
182             std::thread thr(std::move(task));
183             thr.detach();
184             return res;
185         }
186         TListItemPtr<T> head;
187 };
188 #endif
```

8.5. Консоль

```
sergey@svb:~/MAI/OOP/lab8$./run
Use 'help' or 'h' to get help.
h


append or add        insert figure in list
p[rint]                print list
del[ete]               delete figure with the size <side> from the list
s[ort]                 sort list
ps[ort]                sort list in parallel
q[uit]                 exit the program

add
Which figure do you want to append? (pent/hex/oct[agon]): pent
Pentagon: enter side length: 4
add
Which figure do you want to append? (pent/hex/oct[agon]): oct
Octagon: enter side length: 3
add
Which figure do you want to append? (pent/hex/oct[agon]): pent
Pentagon: enter side length: 7
```

```
add
Which figure do you want to append? (pent/hex/oct[agon]): hex
Hexagon: enter side length: 3
add
Which figure do you want to append? (pent/hex/oct[agon]): pent
Pentagon: enter side length: 5
add
Which figure do you want to append? (pent/hex/oct[agon]): oct
Octagon: enter side length: 6
p
Octagon         side = 6        square = 173.82
Pentagon        side = 5        square = 43.01
Hexagon         side = 3        square = 23.38
Pentagon        side = 7        square = 84.30
Octagon         side = 3        square = 43.46
Pentagon        side = 4        square = 27.53

sort
p
Hexagon         side = 3        square = 23.38
Pentagon        side = 4        square = 27.53
Pentagon        side = 5        square = 43.01
Octagon         side = 3        square = 43.46
Pentagon        side = 7        square = 84.30
Octagon         side = 6        square = 173.82

q
sergey@svb:~/MAI/OOP/lab8$./run
Use 'help' or 'h' to get help.
add
Which figure do you want to append? (pent/hex/oct[agon]): pent
Pentagon: enter side length: 4
add
Which figure do you want to append? (pent/hex/oct[agon]): oct
Octagon: enter side length: 3
add
Which figure do you want to append? (pent/hex/oct[agon]): pent
Pentagon: enter side length: 7
add
Which figure do you want to append? (pent/hex/oct[agon]): hex
Hexagon: enter side length: 3
add
Which figure do you want to append? (pent/hex/oct[agon]): pent
Pentagon: enter side length: 5
add
Which figure do you want to append? (pent/hex/oct[agon]): oct
```

```
Octagon: enter side length: 6
p
Octagon         side = 6        square = 173.82
Pentagon        side = 5        square = 43.01
Hexagon         side = 3        square = 23.38
Pentagon        side = 7        square = 84.30
Octagon         side = 3        square = 43.46
Pentagon        side = 4        square = 27.53

psort
p
Hexagon         side = 3        square = 23.38
Pentagon        side = 4        square = 27.53
Pentagon        side = 5        square = 43.01
Octagon         side = 3        square = 43.46
Pentagon        side = 7        square = 84.30
Octagon         side = 6        square = 173.82
q
```

8.6. Выводы

Если выбирать между обычной и многопоточной реализацией, то я бы выбрал второй вариант, поскольку с потоками обычно программы быстрее работают. И вообще параллелизм применяется для разделения обязанностей (это почти всегда приветствуется при разработке программ). И даже имеет смысл распараллелить для разделения функционально не связанных между собой частей программы, если мы хотим, чтобы они, которые относятся к разным частям операции, выполнялись одновременно. Иначе без явного распараллеливания нам пришлось бы обращаться к коду из посторонней части программы во время выполнения операции. Также применение параллелизма повышает производительность программы. Но мы должны понимать, что в каких-то случаях нежелательно его применять. Например, могут быть такие ситуации, когда затраты перевешивают выигрыш. Также потоки являются ограниченными ресурсами. Если одновременно работает слишком много потоков, то ресурсы ОС истощаются, что может привести к замедлению работы всей системы.

Лабораторная работа №9

9.1. Цель работы

- Знакомство с лямбда - выражениями.

9.2. Задание

Используя структуры данных, разработанные для лабораторной работы 6 (контейнер первого уровня и классы-фигуры) необходимо разработать:

- Контейнер второго уровня с использованным шаблонов.
- Реализовать с помощью лямбда-выражений набор команд, совершающих операции над контейнерами 1-го уровня:
  - Генерация фигур со случайным значением параметров;
  - Печать контейнера на экран;
  - Удаление элементов со значением площади меньше определенного числа;
- В контейнер второго уровня поместить цепочку команд.
- Реализовать цикл, который проходит по всем командам в контейнере второго уровня и выполняет их, применяя к контейнеру первого уровня.

Для создания потоков использовать механизмы:

- future
- packaged_task/async

Для обеспечения потоков - безопасности структур данных использовать:

- mutex
- lock_guard

Фигуры: Пятиугольник, шестиугольник, восьмиугольник.
Контейнер 1-го уровня: Бинарное дерево.
Контейнер 2-го уровня: Связанный список.

9.3. Описание

Лямбда-выражение (или просто лямбда) в C++11 — это удобный способ определения анонимного объекта-функции непосредственно в месте его вызова или передачи в функцию в качестве аргумента. Обычно лямбда-выражения используются для инкапсуляции нескольких строк кода, передаваемых алгоритмам или асинхронным методам.

Структура лямбда-функции такова: [capture] (params) body, где «квадратные скобки» - такая часть, которая позволяет привязывать переменные, доступные в текущей области видимости, а «круглые скобки» - другая часть, указывающая список принимаемых параметров лямбда-функции и, наконец, третья часть («фигурные скобки») содержит тело лямбда-функции.

В данной работе применяются некоторые инструменты, например mutable. В классе TBinaryTree<T> рассмотрим строку:
mutable std::recursive_mutex tree_mutex; - здесь модификатор mutable применяется для того, чтобы можно было изменять эту переменную в const - функциях.

9.4. Исходный код

```
1  #define FIGURE_COUNT 10
2
3  int main(int argc, char** argv) {
4      const int size = 16;
5      char s[size];
6
7
8     TBinaryTree<Figure> tree;
9
10    typedef std::function<void(void) > TCommand;
11    TList<TCommand> commandsList;
12
13
14    TCommand cmdInsert = [&]() {
15       size_t seed = std::chrono::system_clock::now().
            time_since_epoch().count();
16
17       std::default_random_engine randEngine(seed);
18       std::uniform_int_distribution<int> generatorFigureType
            (1, 3);
19       std::uniform_int_distribution<int> generatorFigureSide
            (1, 20);
20       for (int i = 0; i < FIGURE_COUNT; ++i) {
21          std::cout << "Command: Inserting ";
22          std::shared_ptr<Figure> ptr_fig;
23          size_t side = generatorFigureSide(randEngine);
24          switch (generatorFigureType(randEngine)) {
25          case 1: {
26             ptr_fig = std::make_shared<Pentagon>(side);
27             break;
28          }
29          case 2: {
30             ptr_fig = std::make_shared<Hexagon>(side);
```

```
31              break;
32          }
33          case 3: {
34              ptr_fig = std::make_shared<Octagon>(side);
35              break;
36          }
37          }
38          tree.Insert(ptr_fig);
39          std::cout << *ptr_fig << std::endl;
40      }
41  };
42
43  TCommand cmdRemove = [&]() {
44
45      if (tree.IsEmpty()) {
46          std::cout << "Command: removing failed! Tree is
                empty" << std::endl;
47      }
48      else {
49          size_t seed = std::chrono::system_clock::now().
                time_since_epoch().count();
50
51          std::default_random_engine randEngine(seed);
52          std::uniform_int_distribution<int>
                generatorFigureSquare(30, 150);
53          double sqr = generatorFigureSquare(randEngine);
54          std::cout << "Command: removing figures with square
                less than " << sqr << std::endl;
55
56          TList<Figure> figuresToDelete;
57          for (auto it_tree : tree) {
58              if (it_tree->Square() < sqr)
59                  figuresToDelete.Push(it_tree);
60          }
61          for (auto it_figlist : figuresToDelete) {
62              tree.Delete(it_figlist);
63          }
64      }
65  };
66
67  TCommand cmdPrint = [&]() {
68      std::cout << "Command: Printing tree" << std::endl;
69      std::cout << tree;
70  };
71
72
```

```
73
74      commandsList.Push(std::shared_ptr<TCommand> (&cmdPrint,
           [](TCommand*) {})); // using custom deleter
75      commandsList.Push(std::shared_ptr<TCommand>(&cmdRemove,
           [](TCommand*) {})); // using custom deleter
76      commandsList.Push(std::shared_ptr<TCommand> (&cmdPrint,
           [](TCommand*) {})); // using custom deleter
77      commandsList.Push(std::shared_ptr<TCommand> (&cmdInsert,
           [](TCommand*) {})); // using custom deleter
78      commandsList.Push(std::shared_ptr<TCommand>(&cmdPrint, [](
           TCommand*) {})); // using custom deleter
79
80
81      while (!commandsList.IsEmpty()) {
82          std::shared_ptr<TCommand> cmd = commandsList.Pop();
83          std::future<void> fut = std::async(*cmd);
84          fut.get();
85          //std::thread(*cmd).detach();
86      }
87
88       return 0;
89  }
```

9.5. Консоль

```
sergey@svb:~/MAI/OOP/lab9$./run
Command: Printing tree
Empty tree
================================================

Command: Inserting Octagon      side = 17       square = 1395.42

Command: Inserting Pentagon     side = 1        square = 1.72

Command: Inserting Pentagon     side = 6        square = 61.94

Command: Inserting Hexagon      side = 4        square = 41.57

Command: Inserting Hexagon      side = 10       square = 259.81

Command: Inserting Hexagon      side = 16       square = 665.11

Command: Inserting Hexagon      side = 1        square = 2.60

Command: Inserting Hexagon      side = 7        square = 127.31
```

```
Command: Inserting Pentagon     side = 12       square = 247.75

Command: Inserting Hexagon      side = 7        square = 127.31

Command: Printing tree
================================================
  Pentagon     side = 1     square = 1.72
                            Hexagon      side = 1     square = 2.60
                   Hexagon      side = 4     square = 41.57
           Pentagon     side = 6     square = 61.94
                            Hexagon      side = 7     square = 127.31
                                                Hexagon      side = 7     square = 127.31
                                       Pentagon     side = 12    square = 247.75
                   Hexagon      side = 10    square = 259.81
                            Hexagon      side = 16    square = 665.11
Octagon      side = 17    square = 1395.42
================================================

Command: removing figures with square less than 134.00
Command: Printing tree
================================================
  Pentagon     side = 12    square = 247.75
           Hexagon      side = 10    square = 259.81
                    Hexagon      side = 16    square = 665.11
Octagon      side = 17    square = 1395.42
================================================
```

9.6. Выводы

В данной работе я познакомился с техникой программирования, которая сочетает в себе преимущества указателей на функции и функциональных объектов (другими словами, обыкновенные объекты с перегруженным «()» оператором) и позволяет избежать неудобств. Это лямбда-выражения. Как и функциональные объекты, они позволяют хранить состояния, но их компактный синтаксис в отличие от функциональных объектов не требует объявления класса, а также помогают нам писать более компактный код и избежать ошибок, нежели используя функциональные объекты.

Исходные файлы лежат на GitHub
(ссылка: https://github.com/SergLih/study_MAI/tree/master/year_2/OOP)