Московский авиационный институт (национальный исследовательский университет)

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

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

Студентка: А. Довженко
Преподаватель: А. В. Поповкин
Группа: 08-207

Вариант: 12
Дата:
Оценка:
Подпись:

Москва, 2017

Лабораторная работа №1

Задача: 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).

Фигуры: трапеция, ромб, пятиугольник.

1 Описание

Основная идея ООП — **объект**. Объект есть сущность, одновременно содержащая данные и поведение. Они являются строительными блоками объектно-ориентированных программ. Та или иная программа, которая задействует объектно-ориентированную технологию, по сути является набором объектов. Перед тем как создать объект C++, необходимо определить его общую форму, используя ключевое слово class. Класс определяет новый пользовательский тип данных, который соединяет в себе код и данные.

Классы в программировании состоят из свойств (атрибутов) и методов. **Свойства** — это любые данные, которыми можно характеризовать объект класса. **Методы** — это функции, которые могут выполнять какие-либо действия над данными (свойствами) класса. Компоненты, использованные при определении класса, называются его членами. Как правило, класс имеет интерфейс и реализацию. Интерфейс — это часть объявления класса, к которой его пользователь имеет прямой доступ. Реализация — это часть объявления класса, доступ к которой пользователь может получить только косвенно, через интерфейс. Открытый интерфейс индентифицируется меткой **public:**, а реализация меткой **private:**.

В ООП существует 3 основных принципа построение классов:

- **Инкапсуляция** — это свойство, позволяющее объединить в классе и данные, и методы, работающие с ним и скрыть детали реализации от пользователя.
- **Наследование** — это свойство, позволяющее создать новый класс-потомок на основе уже существующего, при этом все характеристики класса-родителя присваиваются классу-потомку.
- **Полиморфизм** — свойство классов, позволяющее использовать объекты классов с одинаковым интерфейсом без информации о типе и внутреннй структуре объекта.

Проблема поддержания правильного состояния переменных актуальна и для самого первого момента выставления начальных значений. Для этого в класса предусмотрены специальные методы/функции, называемые конструкторами. **Конструктор** является специальной функцией-членом класса, которая должна определяться с тем же именем, что и класс, чтобы компилятор мог отличить его от других функций класса. Важным отличием конструкторов от других функций является то, что конструкторы не возвращают значений, так что они не могут специфицировать возвращаемый тип. C++ требует вызова конструктора для каждого создаваемого объекта, что позволяет гарантировать корректную инициализацию объекта перед тем, как он будет использоваться в программе — когда объект создается, вызов конструктора происходит автоматически. Любому классу, который не определяет конструктор явным

образом, компилятор предоставляет конструктор по умолчанию, т.е. конструктор без параметров. **Деструктор** класса вызывется при уничтожении объекта. Имя деструктора аналогично имени конструктора, только в начале ставится знак тильды. Деструктор не имеет входных параметров.

Виртуальная функция — это функция-член, которая, как предполагается, будет переопределена в производных классах. При обращении к объекту производного класса, используя указатель или ссылку на базовый класс, можно вызвать виртуальные функции объекта и выполнить переопределенную в производном классе версию функции. Виртуальные функции гарантируют, что выбрана верная версия функции для объекта, независимо от выражения, используемого для вызова функции.

Операции ввода/вывода выполняются с помощью классов istream (потоковые 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: cin — объект класса istream, соответствующий стандартному вводу; cout — объект класса ostream, соответствующий стандартному выводу; cerr — объект класса ostream, соответствующий стандартному выводу для ошибок. Вывод осуществляется, как правило, с помощью перегруженного оператора сдвига влево, а ввод — с помощью оператора сдвига вправо.

2 Исходный код

trapeze.cpp	
Trapeze();	Конструктор класса
Trapeze(std::istream &is);	Конструктор класса из стандартного потока
Trapeze(const Trapeze& orig);	Конструктор копии класса
double Square();	Площадь фигуры
void Print();	Печать фигуры
~Trapeze();	Деконструктор класса
rhomb.cpp	
Rhomb();	Конструктор класса
Rhomb(std::istream &is);	Конструктор класса из стандартного потока
Rhomb(const Rhomb& orig);	Конструктор копии класса
double Square();	Площадь фигуры
void Print();	Печать фигуры
~Rhomb();	Деконструктор класса
pentagon.cpp	
Pentagon();	Конструктор класса
Pentagon(std::istream &is);	Конструктор класса из стандартного потока
Pentagon(const Pentagon& orig);	Конструктор копии класса
double Square();	Площадь фигуры
void Print();	Печать фигуры
~Pentagon();	Деконструктор класса

```
1
2  class Figure
3  {
4  public:
5      virtual double Square() = 0;
6      virtual void Print() = 0;
7      virtual ~Figure() {};
8  };
9
10 class Trapeze : public Figure
11 {
12 public:
13     Trapeze();
14     Trapeze(std::istream &is);
15     Trapeze(int32_t small_base, int32_t big_base, int32_t l_side, int32_t r_side);
16     Trapeze(const Trapeze& orig);
```

```
17      double Square() override;
18      void Print() override;
19      virtual ~Trapeze();
20
21  private:
22      int32_t small_base;
23      int32_t big_base;
24      int32_t l_side;
25      int32_t r_side;
26  };
27
28
29  class Rhomb : public Figure
30  {
31  public:
32      Rhomb();
33      Rhomb(std::istream &is);
34      Rhomb(int32_t side, int32_t smaller_angle);
35      Rhomb(const Rhomb& orig);
36      double Square() override;
37      void Print() override;
38      virtual ~Rhomb();
39
40  private:
41      int32_t side;
42      int32_t smaller_angle;
43  };
44
45  class Pentagon : public Figure
46  {
47  public:
48      Pentagon();
49      Pentagon(std::istream& is);
50      Pentagon(int32_t side);
51      Pentagon(const Pentagon& orig);
52      double Square() override;
53      void Print() override;
54      virtual ~Pentagon();
55
56  private:
57      int32_t side;
58  };
```

3 Консоль

```
karma@karma:~/mai_study/OOP/lab1$ valgrind --leak-check=full ./run
==5388== Memcheck,a memory error detector
==5388== Copyright (C) 2002-2015,and GNU GPL'd,by Julian Seward et al.
==5388== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==5388== Command: ./run
==5388==
Use 'help'or 'h'to get help.
h
Commands 'create_trapeze'and 'cr_tr'create new trapeze with your parameters.
Commands 'create_rhomb'and 'cr_rh'create new rhomb with your parameters.
Commands 'create_pentagon'and 'cr_pen'create new pentagon with your parameters.
Commands 'print_trapeze'and 'pr_tr'output parameters of trapeze.
Commands 'print_rhomb'and 'pr_rh'output parameters of rhomb.
Commands 'print_pentagon'and 'pr_pen'output parameters of pentagon.
Commands 'square_trapeze'and 'sq_tr'output square of trapeze.
Commands 'square_rhomb'and 'sq_rh'output square of rhomb.
Commands 'square_pentagon'and 'sq_pen'output square of pentagon.
Commands 'quit'and 'q'exit the program.
cr_tr
Enter smaller base,bigger base,left side and right side.
2 7 4 4
pr_tr
Smaller base = 2,bigger base = 7,left side = 4,right side = 4
cr_pen
Enter side.
10000
sq_pen
Square: 1.72048e+08
cr_pen
Enter side.
9
pr_pen
Sides =  9
cr_rh
Enter side and smaller angle.
15 30
pr_rh
Side = 15,smaller_angle = 30
pr_tr
```

```
Smaller base = 2,bigger base = 7,left side = 4,right side = 4
pr_pen
Sides =  9
sq_tr
Square: 16
sq_rh
Square: 112.5
sq_pen
Square: 139.359
q
Trapeze deleted
Rhomb deleted
Pentagon deleted
==5388==
==5388== HEAP SUMMARY:
==5388==     in use at exit: 72,704 bytes in 1 blocks
==5388==   total heap usage: 7 allocs,6 frees,74,824 bytes allocated
==5388==
==5388== LEAK SUMMARY:
==5388==    definitely lost: 0 bytes in 0 blocks
==5388==    indirectly lost: 0 bytes in 0 blocks
==5388==      possibly lost: 0 bytes in 0 blocks
==5388==    still reachable: 72,704 bytes in 1 blocks
==5388==         suppressed: 0 bytes in 0 blocks
==5388== Reachable blocks (those to which a pointer was found) are not shown.
==5388== To see them,rerun with: --leak-check=full --show-leak-kinds=all
==5388==
==5388== For counts of detected and suppressed errors,rerun with: -v
==5388== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Лабораторная работа №2

Задача: Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровная, 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).

Фигура: трапеция.
Контейнер: связный список.

1 Описание

Динамические структуры данных используются в тех случаях, когда мы заранее не знаем, сколько памяти необходимо выделить для нашей программы – это выясняется только в процессе работы. В общем случае эта структура представляет собо отдельные элементы, связанные между собой с помощью ссылок. Каждый элемент состоит из двух областей памяти: поля данных и ссылок. Ссылки – это адреса других узлов того же типа, с которыми данный элемент логически связан. При добавлении нового элемента в такую структуру выделяется новый блок памяти и устанавливаются связи этого элемента с уже существующими.

Структура данных список является простейшим типом данных динамической структуры, состоящей из узлов. Каждый узел включает в себя в классическом варианте два поля: данные и указатель на следующий узел в списке. Элементы связного списка можно вставлять и удалять произвольным образом. Доступ к списку осуществляется через указатель, который содержит ядрес первого элемента списка, называемого головой списка.

Параметры в функцию могут передаваться одним из следующих способов: по значению и по ссылке. При передаче аргументов по значению компилятор создает временную копию объекта, который должен быть передан, и размещает его в области стековой памяти, предназначенной для хранения локальных объектов. Вызываемая функция оперирует именно с этой копией, не оказывая влияния на оригинал объекта. Прототипы функций, принимающих аргументы по значению, предусматривают в качестве параметров указание типа объекта,а не его адреса. Если же необходимо, чтобы функция модифицировала оригинал объекта, используется передача параметров по ссылке. При этом в функцию передается не сам объект, а только его адрес. Таким образом, все модификации в теле функции переданных ей по ссылке аргументов воздействуют на объект. Использование передачи адреса объекта весьма эффективный способ работы с большим числом данных. Кроме того, так как передается адрес, а не сам объект, существенно экономится стековая память.

2 Исходный код

trapeze.cpp	
Trapeze();	Конструктор класса
Trapeze(std::istream &is);	Конструктор класса из стандартного потока

Trapeze(const Trapeze& orig);	Конструктор копии класса
double Square();	Площадь фигуры
void Print();	Печать фигуры
~Trapeze();	Деконструктор класса
bool operator ==(const Trapeze &obj) const;	Переопределенный оператор сравнения
Trapeze& operator =(const Trapeze &obj);	Переопределенный оператор копирования
friend std::ostream& operator «(std::ostream &os, const Trapeze &obj);	Переопределенный оператор вывода в поток std::ostream
friend std::istream& operator »(std::istream &is, Trapeze &obj);	Переопределенный оператор ввода из потока std::istream
TListItem.cpp	
TListItem(const Trapeze &obj);	Конструктор класса
Trapeze GetFigure() const;	Получение фигуры из узла
TListItem* GetNext();	Получение ссылки на следующий узел
TListItem* GetPrev();	Получение ссылки на предыдущий узел
void SetNext(TListItem *item);	Установка ссылки на следующий узел
void SetPrev(TListItem *item);	Установка ссылки на предыдущий узел
friend std::ostream& operator«(std::ostream &os, const TListItem &obj);	Переопределенный оператор вывода в поток std::ostream
virtual ~TListItem();	Деконструктор класса
TList.cpp	
TList();	Конструктор класса
void Push(Trapeze &obj);	Добавление фигуры в список
Trapeze Pop();	Получение фигуры из списка
const bool IsEmpty() const;	Проверка, пуст ли список
uint32t GetLength();	Получение длины списка
friend std::ostream& operator«(std::ostream &os, const TList &list);	Переопределенный оператор вывода в поток std::ostream
virtual ~TList();	Деконструктор класса

```
1
2  class TList
3  {
4  public:
5      TList();
6      void Push(Trapeze &obj);
```

```
7     const bool IsEmpty() const;
8     uint32_t GetLength();
9     Trapeze Pop();
10    friend std::ostream& operator<<(std::ostream &os, const TList &list);
11    virtual ~TList();
12
13  private:
14    uint32_t length;
15    TListItem *head;
16
17    void PushFirst(Trapeze &obj);
18    void PushLast(Trapeze &obj);
19    void PushAtIndex(Trapeze &obj, int32_t ind);
20    Trapeze PopFirst();
21    Trapeze PopLast();
22    Trapeze PopAtIndex(int32_t ind);
23  };
24
25  class TListItem
26  {
27  public:
28    TListItem(const Trapeze &obj);
29
30    Trapeze GetFigure() const;
31    TListItem* GetNext();
32    TListItem* GetPrev();
33    void SetNext(TListItem *item);
34    void SetPrev(TListItem *item);
35    friend std::ostream& operator<<(std::ostream &os, const TListItem &obj);
36
37    virtual ~TListItem(){};
38
39  private:
40    Trapeze item;
41    TListItem *next;
42    TListItem *prev;
43  };
44
45  class Trapeze : public Figure
46  {
47  public:
48    Trapeze();
49    Trapeze(std::istream &is);
50    Trapeze(int32_t small_base, int32_t big_base, int32_t l_side, int32_t r_side);
51    Trapeze(const Trapeze &orig);
52
53    bool operator ==(const Trapeze &obj) const;
54    Trapeze& operator =(const Trapeze &obj);
55    friend std::ostream& operator <<(std::ostream &os, const Trapeze &obj);
```

```
56      friend std::istream& operator >>(std::istream &is, Trapeze &obj);
57
58      double Square() override;
59      void Print() override;
60      virtual ~Trapeze();
61
62  private:
63      int32_t small_base;
64      int32_t big_base;
65      int32_t l_side;
66      int32_t r_side;
67  };
```

3 Консоль

```
karma@karma:~/mai_study/OOP/lab2$ valgrind --leak-check=full ./run
==4059== Memcheck,a memory error detector
==4059== Copyright (C) 2002-2015,and GNU GPL'd,by Julian Seward et al.
==4059== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==4059== Command: ./run
==4059==
Choose an operation:
1) Add trapeze
2) Delete trapeze from list
3) Print list
0) Exit
3
The list is empty.

Choose an operation:
1) Add trapeze
2) Delete trapeze from list
3) Print list
0) Exit
1
Enter bigger base: 5
Enter smaller base: 2
Enter left side: 1
Enter right side: 1
Enter index = 0
Choose an operation:
1) Add trapeze
2) Delete trapeze from list
```

```
3) Print list
0) Exit
1
Enter bigger base:
4
Enter smaller base: 3
Enter left side: 3
Enter right side: 3
Enter index = 0
Choose an operation:
1) Add trapeze
2) Delete trapeze from list
3) Print list
0) Exit
1
Enter bigger base: 5
Enter smaller base: 5
Enter left side: 5
Enter right side: 5
Enter index = 1
Choose an operation:
1) Add trapeze
2) Delete trapeze from list
3) Print list
0) Exit
1
Enter bigger base: 2
Enter smaller base: 2
Enter left side: 2
Enter right side: 2
Enter index = 2
Choose an operation:
1) Add trapeze
2) Delete trapeze from list
3) Print list
0) Exit
3
idx: 0   (3 4 3 3)

idx: 1   (2 5 1 1)
```

```
idx: 2   (5 5 5 5)

idx: 3   (2 2 2 2)


Choose an operation:
1) Add trapeze
2) Delete trapeze from list
3) Print list
0) Exit
2
Enter index = 1
Choose an operation:
1) Add trapeze
2) Delete trapeze from list
3) Print list
0) Exit
2
Enter index = 1
Choose an operation:
1) Add trapeze
2) Delete trapeze from list
3) Print list
0) Exit
2
Enter index = 1
Choose an operation:
1) Add trapeze
2) Delete trapeze from list
3) Print list
0) Exit
3
idx: 0   (3 4 3 3)


Choose an operation:
1) Add trapeze
2) Delete trapeze from list
3) Print list
0) Exit
1
```

```
Enter bigger base: 3
Enter smaller base: 3
Enter left side: 3
Enter right side: 3
Enter index = 0
Choose an operation:
1) Add trapeze
2) Delete trapeze from list
3) Print list
0) Exit
1
Enter bigger base: 2
Enter smaller base: 2
Enter left side: 2
Enter right side: 2
Enter index = 1
Choose an operation:
1) Add trapeze
2) Delete trapeze from list
3) Print list
0) Exit
3
idx: 0   (3 3 3 3)

idx: 1   (3 4 3 3)

idx: 2   (2 2 2 2)


Choose an operation:
1) Add trapeze
2) Delete trapeze from list
3) Print list
0) Exit
0
==4059==
==4059== HEAP SUMMARY:
==4059==     in use at exit: 72,704 bytes in 1 blocks
==4059==   total heap usage: 9 allocs,8 frees,75,040 bytes allocated
==4059==
==4059== LEAK SUMMARY:
```

```
==4059==    definitely lost: 0 bytes in 0 blocks
==4059==    indirectly lost: 0 bytes in 0 blocks
==4059==      possibly lost: 0 bytes in 0 blocks
==4059==    still reachable: 72,704 bytes in 1 blocks
==4059==         suppressed: 0 bytes in 0 blocks
==4059== Reachable blocks (those to which a pointer was found) are not shown.
==4059== To see them,rerun with: --leak-check=full --show-leak-kinds=all
==4059==
==4059== For counts of detected and suppressed errors,rerun with: -v
==4059== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Лабораторная работа №3

Задача: Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня, содержащий все три фигуры, согласно варианту задания Классы 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 добалению 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 ostream.
- Класс-контейнер должен иметь деструктор, удаляющий все элементы контейнера.
- Классы должны быть расположены в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

Фигуры: трапеция, ромб, пятиугольник
Контейнер: связный список.

1 Описание

Умный указатель – класс (обычно шаблонный), имитирующий интерфейс обычного указателя и добавляющий некую новую функциональность, например, проверку границ при доступе или очистку памяти.
Существует 3 вида умных указателей стандартной библиотеки C++:

- unique ptr – обеспечивает, чтобы у базового указателя был только один владелец. Может быть передан новому владельцу, но не может быть скопирован или сделан общим. Заменяет auto ptr, использовать который не рекомендуется.
- shared ptr – умный указатель с подсчитанными ссылками. Используется, когда необходимо присвоить один необработанный указатель нескольким владельцам, например, когда копия указателя возвращается из контейнера, но требуется сохранить оригинал. Необработанный указатель не будет удален до тех пор, пока все владельцы shared ptr не выйдут из области или не откажутся от владения.
- weak ptr – умный указатель для особых случаев использования с shared ptr. weak ptr предоставляет доступ к объекту, который принадлежит одному или нескольким экземплярам shared ptr, но не участвует в подсчете ссылок. Используется, когда требуется отслеживать объект, но не требуется, чтобы он оставался в активном состоянии.

2 Исходный код

TListItem.cpp	
TListItem(const std::sharedptr<Figure> &obj);	Конструктор класса
std::sharedptr<Figure> GetFigure() const;	Получение фигуры из узла
std::sharedptr<TListItem> GetNext();	Получение ссылки на следующий узел
std::sharedptr<TListItem> GetPrev();	Получение ссылки на предыдущий узел
void SetNext(std::sharedptr<TListItem> item);	Установка ссылки на следующий узел
void SetPrev(std::sharedptr<TListItem> item);	Установка ссылки на предыдущий узел
friend std::ostream& operator«(std::ostream &os, const TListItem &obj);	Переопределенный оператор вывода в поток std::ostream
virtual ~TListItem();	Деконструктор класса

TList.cpp	
TList();	Конструктор класса
void Push(std::sharedptr<Figure> &obj);	Добавление фигуры в список
std::sharedptr<Figure> Pop();	Получение фигуры из списка
const bool IsEmpty() const;	Проверка, пуст ли список
uint32t GetLength();	Получение длины списка
friend std::ostream& operator«(std::ostream &os, const TList &list);	Переопределенный оператор вывода в поток std::ostream
virtual ~TList();	Деконструктор класса

```
1
2  class TList
3  {
4  public:
5      TList();
6      void Push(std::shared_ptr<Figure> &obj);
7      const bool IsEmpty() const;
8      uint32_t GetLength();
9      std::shared_ptr<Figure> Pop();
10     friend std::ostream& operator<<(std::ostream &os, const TList &list);
11     virtual ~TList();
12
13 private:
14     uint32_t length;
15     std::shared_ptr<TListItem> head;
16
17     void PushFirst(std::shared_ptr<Figure> &obj);
18     void PushLast(std::shared_ptr<Figure> &obj);
19     void PushAtIndex(std::shared_ptr<Figure> &obj, int32_t ind);
20     std::shared_ptr<Figure> PopFirst();
21     std::shared_ptr<Figure> PopLast();
22     std::shared_ptr<Figure> PopAtIndex(int32_t ind);
23 };
24
25 class TListItem
26 {
27 public:
28     TListItem(const std::shared_ptr<Figure> &obj);
29
30     std::shared_ptr<Figure> GetFigure() const;
31     std::shared_ptr<TListItem> GetNext();
32     std::shared_ptr<TListItem> GetPrev();
33     void SetNext(std::shared_ptr<TListItem> item);
34     void SetPrev(std::shared_ptr<TListItem> item);
35     friend std::ostream& operator<<(std::ostream &os, const TListItem &obj);
```

```
36
37      virtual ~TListItem(){};
38
39  private:
40      std::shared_ptr<Figure> item;
41      std::shared_ptr<TListItem> next;
42      std::shared_ptr<TListItem> prev;
43  };
```

3 Консоль

```
karma@karma:~/mai_study/OOP/lab3$ ./run
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
0) Exit
1
Enter bigger base: 5
Enter smaller base: 2
Enter left side: 2
Enter right side: 2
Enter index = 0
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
0) Exit
3
Enter side: 5
Enter index = 0
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
0) Exit
```

```
3
Enter side: 10
Enter index = 1
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
0) Exit
2
Enter side: 6
Enter smaller angle: 40
Enter index = 1
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
0) Exit
5
idx: 0   Sides =  5,type: pentagon

idx: 1   Side = 6,smaller_angle = 40,type: rhomb

idx: 2   Smaller base = 2,bigger base = 5,left side = 2,right side = 2,type:
trapeze

idx: 3   Sides =  10,type: pentagon


Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
0) Exit
4
Enter index = 2
```

```
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
0) Exit
4
Enter index = 0
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
0) Exit
5
idx: 0   Side = 6,smaller_angle = 40,type: rhomb

idx: 1   Sides =  10,type: pentagon


Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
0) Exit
0
```

Лабораторная работа №4

Задача: Необходимо спроектировать и запрограммировать на языке C++ шаблон класса-контейнера первого уровня, содержащий все три фигуры, согласно варианту задания

Классы должны удовлетворять следующим правилам:

- Требования к классу фигуры аналогичны требованиям из лабораторной работы 1.
- Шаблон класса-контейнера должен содержать объекты, используя std::shared ptr<...>.
- Шаблон класса-контейнера должен иметь метод по добалению фигуры в контейнер.
- Шаблон класса-контейнера должен иметь методы по получению фигуры из контейнера.
- Шаблон класса-контейнера должен иметь метод по удалению фигуры из контейнера.
- Шаблон класса-контейнера должен иметь перегруженный оператор по выводу контейнера в поток 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).

Фигуры: трапеция, ромб, пятиугольник
Контейнер: связный список.

1 Описание

Шаблоны (template) предназначены для кодирования обобщенных алгоритмов, без привязки к некоторым параметрам (например, типам данных, размерам буферов, значениям по умолчанию). В C++ возможно создание шаблонов функций и классов. Шаблоны позволяют создавать параметризованные классы и функции. Параметром может быть любой типа или значение одного из допустимых типов (целое число, перечисляемый тип, указатель на любой объект с глобально доступным именем, ссылка).

Шаблоны используются в случаях дублирования одного и того же кода для нескольких типов. Например, можно использовать шаблоны функций для создания набора функций, которые применяют один и тот же алгоритм к различным типам данных. Кроме того, шаблоны классов можно использовать для разработки набора типобезопасных классов. Иногда рекомендуется использовать шаблоны вместо макросов C и пустых указателей. Шаблоны особенно полезны при работы с коллекциями и умными указателями.

2 Исходный код

TListItem.cpp	
TListItem(const std::sharedptr<T>&obj);	Конструктор класса
std::sharedptr<T> GetFigure() const;	Получение фигуры из узла
std::sharedptr<TListItem<T» GetNext();	Получение ссылки на следующий узел
std::sharedptr<TListItem<T» GetPrev();	Получение ссылки на предыдущий узел
void SetNext(std::sharedptr<TListItem<T» item);	Установка ссылки на следующий узел
void SetPrev(std::sharedptr<TListItem<T» item);	Установка ссылки на предыдущий узел
friend std::ostream& operator«(std::ostream &os, const TListItem<A> &obj);	Переопределенный оператор вывода в поток std::ostream
virtual ~TListItem();	Деконструктор класса
TList.cpp	
TList();	Конструктор класса
void Push(std::sharedptr<T> &obj);	Добавление фигуры в список

std::sharedptr<T> Pop();	Получение фигуры из списка
const bool IsEmpty() const;	Проверка, пуст ли список
uint32t GetLength();	Получение длины списка
friend std::ostream& operator«(std::ostream &os, const TList<A> &list);	Переопределенный оператор вывода в поток std::ostream
virtual ~TList();	Деконструктор класса

```
1
2  template <class T>
3  class TList
4  {
5  public:
6      TList();
7      void Push(std::shared_ptr<T> &obj);
8      const bool IsEmpty() const;
9      uint32_t GetLength();
10     std::shared_ptr<T> Pop();
11     template <class A> friend std::ostream& operator<<(std::ostream &os, const TList<A>
           &list);
12     void Del();
13     virtual ~TList();
14
15 private:
16     uint32_t length;
17     std::shared_ptr<TListItem<T>> head;
18
19     void PushFirst(std::shared_ptr<T> &obj);
20     void PushLast(std::shared_ptr<T> &obj);
21     void PushAtIndex(std::shared_ptr<T> &obj, int32_t ind);
22     std::shared_ptr<T> PopFirst();
23     std::shared_ptr<T> PopLast();
24     std::shared_ptr<T> PopAtIndex(int32_t ind);
25 };
26
27 template <class T>
28 class TListItem
29 {
30 public:
31     TListItem(const std::shared_ptr<T> &obj);
32
33     std::shared_ptr<T> GetFigure() const;
34     std::shared_ptr<TListItem<T>> GetNext();
35     std::shared_ptr<TListItem<T>> GetPrev();
36     void SetNext(std::shared_ptr<TListItem<T>> item);
37     void SetPrev(std::shared_ptr<TListItem<T>> item);
38     template <class A> friend std::ostream& operator<<(std::ostream &os, const
```

```
        TListItem<A> &obj);
39
40      virtual ~TListItem(){};
41
42  private:
43      std::shared_ptr<T> item;
44      std::shared_ptr<TListItem<T>> next;
45      std::shared_ptr<TListItem<T>> prev;
46  };
```

3 Консоль

```
karma@karma:~/mai_study/OOP/lab4$ ./run
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
0) Exit
1
Enter bigger base: 10
Enter smaller base: 5
Enter left side: 5
Enter right side: 5
Enter index = 0
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
0) Exit
1
Enter bigger base: 15
Enter smaller base: 10
Enter left side: 5
Enter right side: 5
Enter index = 0
Choose an operation:
1) Add trapeze
2) Add rhomb
```

```
3) Add pentagon
4) Delete figure from list
5) Print list
0) Exit
2
Enter side: 10
Enter smaller angle: 60
Enter index = 1
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
0) Exit
3
Enter side: 3
Enter index = 2
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
0) Exit
5
idx: 0   Smaller base = 10,bigger base = 15,left side = 5,right side = 5,type:
trapeze

idx: 1   Smaller base = 5,bigger base = 10,left side = 5,right side = 5,type:
trapeze

idx: 2   Side = 10,smaller_angle = 60,type: rhomb

idx: 3   Sides =  3,type: pentagon


Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
```

```
4) Delete figure from list
5) Print list
0) Exit
4
Enter index = 0
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
0) Exit
4
Enter index = 1
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
0) Exit
4
Enter index = 1
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
0) Exit
4
Enter index = 0
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
0) Exit
5
The list is empty.
```

```
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
0) Exit
0
```

Лабораторная работа №5

Задача: Используя структуры данных, разработанные для предыдущей лабораторной работы (ЛР №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.

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

Фигуры: трапеция, ромб, пятиугольник.
Контейнер: связный список.

1 Описание

Для доступа к элементам некоторого множества элементов используют специальные объекты, называемые итераторами. В контейнерных типах stl они доступны через методы класса (например, begin() в шаблоне класса vector). Функциональные возможности указателей и итераторов близки, так что обычный указатель тоже может использоваться как итератор.

Категории итераторов:

- Итератор ввода (input iterator) – используется потоками ввода.
- Итератор вывода (output iterator) – используется потоками вывода.
- Однонаправленный итератор (forward iterator) – для прохода по элементам в одном направлении.
- Двунаправленный итератор (bidirectional iterator) – способен пройти по элементам в любом направлении. Такие итераторы реализованы в некоторых контейнерных типах stl (list, set, multiset, map, multimap).
- Итераторы произвольного доступа (random access) – через них можно иметь доступ к любому элементу. Такие итераторы реализованы в некоторых контейнерных типах stl (vector, deque, string, array).

2 Исходный код

Описание классов фигур и класса-контейнера остается неизменным.

```
1
2  template <class N, class T>
3  class TIterator
4  {
5  public:
6      TIterator(std::shared_ptr<N> n) {
7          cur = n;
8      }
9
10     std::shared_ptr<T> operator* () {
11         return cur->GetFigure();
12     }
13
14     std::shared_ptr<T> operator-> () {
15         return cur->GetFigure();
16     }
```

```
17
18      void operator++() {
19          cur = cur->GetNext();
20      }
21
22      TIterator operator++ (int) {
23          TIterator cur(*this);
24          ++(*this);
25          return cur;
26      }
27
28      void operator--() {
29          cur = cur->GetPrev();
30      }
31
32      TIterator operator-- (int) {
33          TIterator cur(*this);
34          --(*this);
35          return cur;
36      }
37
38      bool operator== (const TIterator &i) {
39          return (cur == i.cur);
40      }
41
42      bool operator!= (const TIterator &i) {
43          return (cur != i.cur);
44      }
45
46  private:
47      std::shared_ptr<N> cur;
48  };
```

3 Консоль

```
karma@karma:~/mai_study/OOP/lab5$ ./run
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
6) Print list with iterator
0) Exit
2
```

```
Enter side: 10
Enter smaller angle: 10
Enter index = 0
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
6) Print list with iterator
0) Exit
2
Enter side: 9
Enter smaller angle: 20
Enter index = 0
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
6) Print list with iterator
0) Exit
1
Enter bigger base: 9
Enter smaller base: 8
Enter left side: 7
Enter right side: 6
Enter index = 1
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
6) Print list with iterator
0) Exit
3
Enter side: 5
Enter index = 0
Choose an operation:
```

```
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
6) Print list with iterator
0) Exit
5
idx: 0   Sides =  5,type: pentagon

idx: 1   Side = 9,smaller_angle = 20,type: rhomb

idx: 2   Side = 10,smaller_angle = 10,type: rhomb

idx: 3   Smaller base = 8,bigger base = 9,left side = 7,right side = 6,type:
trapeze


Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
6) Print list with iterator
0) Exit
4
Enter index = 2
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
6) Print list with iterator
0) Exit
6
Sides =  5,type: pentagon
Side = 9,smaller_angle = 20,type: rhomb
Smaller base = 8,bigger base = 9,left side = 7,right side = 6,type: trapeze
Choose an operation:
```

```
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
6) Print list with iterator
0) Exit
0
```

Лабораторная работа №6

Задача: Используя структуры данных, разработанные для предыдущей лабораторной работы (ЛР №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-ого уровня, согласно варианту задания).
Для вызова аллокатора должны быть переопределены операторы new и delete у классов-фигур.

Фигуры: трапеция, ромб, пятиугольник.
Контейнер 1-ого уровня: связный список.
Контейнер 2-ого уровня: стек.

1 Описание

Аллокатор памяти – часть программы (как прикладной, так и операционной системы), обрабатывающая запросы на выделение и освобождение оперативной памяти или запросы на включение заданной области памяти в адресное пространство процессора.
Основное назначение аллокатора памяти в первом смысле – реализация динамической памяти. В языке C динамическое выделение памяти производится через функцию malloc.
Программисты должны учитывать последствия динамического выделения памяти и дважды обдумать использование функции malloc или оператора new. Легко убедить себя, что вы не делаете так уж много аллокаций, а значит большого значения это не имеет, но такой тип мышления распространяется лавиной по всей команде и приводит к медленной смерти. Фрагментация и потери в производительности, связанные с использование динамической памяти, не будучи пресеченными в зародыше, могут иметь катастрофические трудноразрешаемые последствия в вашем дальнейшем цикле разработки. Проекты, где управление и распределение памяти не продумано надлежащим образом, часто страдают от случайных сбоев после длительной сессии из-за нехватки памяти и стоят сотни часов работы программистов, пытающихся освободить память и реорганизовать ее выделение.

2 Исходный код

Описание классов фигур и класса-контейнера остается неизменным.

TAllocationBlock.cpp	
TAllocationBlock(int32t size, int32t count);	Конструктор класса
void *Allocate();	Выделение памяти
void Deallocate(void *ptr);	Освобождение памяти
bool Empty();	Проверка, пуст ли аллокатор
int32t Size();	Получение количества выделенных блоков
virtual ~TAllocationBlock();	Деконструктор класса

```
1
2 class TAllocationBlock
3 {
4 public:
```

```
5       TAllocationBlock(int32_t size, int32_t count);
6       void *Allocate();
7       void Deallocate(void *ptr);
8       bool Empty();
9       int32_t Size();
10
11      virtual ~TAllocationBlock();
12
13  private:
14      Byte *_used_blocks;
15      TStack<void *>_free_blocks;
16  };
```

3 Консоль

```
karma@karma:~/mai_study/OOP/lab6$ ./run
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
6) Print list with iterator
0) Exit
1
Enter bigger base: 10
Enter smaller base: 10
Enter left side: 10
Enter right side: 10
Enter index = 0
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
6) Print list with iterator
0) Exit
3
Enter side: 10
Enter index = 0
Choose an operation:
```

```
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
6) Print list with iterator
0) Exit
2
Enter side: 10
Enter smaller angle: 10
Enter index = 1
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
6) Print list with iterator
0) Exit
5
idx: 0   Sides =  10,type: pentagon

idx: 1   Smaller base = 10,bigger base = 10,left side = 10,right side = 10,type:
trapeze

idx: 2   Side = 10,smaller_angle = 10,type: rhomb


Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
6) Print list with iterator
0) Exit
4
Enter index = 0
Choose an operation:
1) Add trapeze
2) Add rhomb
```

```
3) Add pentagon
4) Delete figure from list
5) Print list
6) Print list with iterator
0) Exit
4
Enter index = 0
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
6) Print list with iterator
0) Exit
6
Side = 10,smaller_angle = 10,type: rhomb
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
6) Print list with iterator
0) Exit
1
Enter bigger base: 10
Enter smaller base: 10
Enter left side: 10
Enter right side: 10
Enter index = 0
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
6) Print list with iterator
0) Exit
3
Enter side: 10
```

```
Enter index = 1
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
6) Print list with iterator
0) Exit
5
idx: 0   Smaller base = 10,bigger base = 10,left side = 10,right side = 10,type:
trapeze

idx: 1   Side = 10,smaller_angle = 10,type: rhomb

idx: 2   Sides =  10,type: pentagon


Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Print list
6) Print list with iterator
0) Exit
0
```

Лабораторная работа №7

Задача: Необходимо реализовать динамическую структуру данных – "Хранилище объектов"и алгоритм работы с ней. "Хранилище объектов"представляет собой контейнер стек. Каждым элементом контейнера является динамическая структура список. Таким образом, у нас получается контейнер в контейнере. Элементов второго контейнера является объект-фигура, определенная вариантом задания.
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. При удалении объектов должно выполянться правило, что контейнер второго уровня не должен быть пустым. Т.е. если он становится пустым, то он должен удалиться.

Фигуры: трапеция, ромб, пятиугольник.
Контейнер 1-ого уровня: связный список.
Контейнер 2-ого уровня: стек.

1 Описание

Принцип открытости/закрытости (OCP) – принцип ООП, устанавливающий следующее положение: "программные сущности (классы, модули, функции и т.п.) должны быть открыты для расширения, но закрыты для изменения".
Контейнер в программировании – структура (АТД), позволяющая инкапсулировать в себе объекты любого типа. Объектами (переменными) контейнеров являются коллекции, которые уже могут содержать в себе объекты определенного типа.
Например, в языке C++, std::list (шаблонный класс) является контейнером, а объект его класса-конкретизации, как например, std::list<int> mylist является коллекцией.
Среди "широких масс"программистов наиболее известны контейнеры, построенные на основе шаблонов, однако, существуют и реализации в виде библиотек (наиболее широко известна библиотека GLib). Кроме того, применяются и узкоспециализированные решения. Примерами контейнеров в C++ являются контейнеры из стандартной библиотеки (STL) – map, vector и т.д. В контейнерах часто встречаются реализации алгоритмов для них. В ряде языков программирования (особенно в скриптовых типа Perl или PHP) контейнеры и работа с ними встроена в язык.
Стек – тип или структура данных в виде набора элементов, которые расположены по принципу LIFO, т.е. "последний пришел, первый вышел". Доступ к элементам осуществляет через обращение к головному элементу (тот, который был добавлен последним).

2 Исходный код

Описание классов фигур и класса-контейнера списка остается неизменным.

TStack.hpp	
TStack();	Конструктор класса
void Push(const O&);	Добавление элемента в стек
void Print();	Печать стека
void RemoveByType(const int&);	Удаление из стека элементов одного типа
void RemoveLesser(const double&);	Удаление из стека элементов, площадь которых меньше, чем заданная
virtual ~TStack();	Деконструктор класса

```
1
2 template <typename Q, typename O> class TStack
3 {
```

```
4  private:
5      class Node {
6      public:
7          Q data;
8          std::shared_ptr<Node> next;
9          Node();
10         Node(const Q&);
11         int itemsInNode;
12     };
13
14     std::shared_ptr<Node> head;
15     int count;
16 public:
17     TStack();
18
19     void Push(const Q&);
20     void Print();
21     void RemoveByType(const int&);
22     void RemoveLesser(const double&);
23
24     virtual ~TStack();
25 };
26
27 template <typename T> class TList
28 {
29 private:
30     class TNode {
31     public:
32         TNode();
33         TNode(const std::shared_ptr<T>&);
34         auto GetNext() const;
35         auto GetItem() const;
36         std::shared_ptr<T> item;
37         std::shared_ptr<TNode> next;
38
39         void* operator new(size_t);
40         void operator delete(void*);
41         static TAllocator nodeAllocator;
42     };
43
44     template <typename N, typename M>
45     class TIterator {
46     private:
47         N nodePtr;
48     public:
49         TIterator(const N&);
50         std::shared_ptr<M> operator* ();
51         std::shared_ptr<M> operator-> ();
52         void operator ++ ();
```

```
53          bool operator == (const TIterator&);
54          bool operator != (const TIterator&);
55      };
56
57      int length;
58
59      std::shared_ptr<TNode> head;
60      auto psort(std::shared_ptr<TNode>&);
61      auto pparsort(std::shared_ptr<TNode>& head);
62      auto partition(std::shared_ptr<TNode>&);
63
64  public:
65      TList();
66      virtual ~TList();
67      bool PushFront(const std::shared_ptr<T>&);
68      bool Push(const std::shared_ptr<T>&, const int);
69      bool PopFront();
70      bool Pop(const int);
71      bool IsEmpty() const;
72      int GetLength() const;
73      auto& getHead();
74      auto&& getTail();
75      void sort();
76      void parSort();
77
78      TIterator<std::shared_ptr<TNode>, T> begin() {return TIterator<std::shared_ptr<
            TNode>, T>(head->next);};
79      TIterator<std::shared_ptr<TNode>, T> end() {return TIterator<std::shared_ptr<TNode
            >, T>(nullptr);};
80
81      template <typename A> friend std::ostream& operator<< (std::ostream&, TList<A>&);
82  };
```

3 Консоль

```
karma@karma:~/mai_study/OOP/lab7$ ./run
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete by criteria
5) Print
0) Exit
1
Enter bigger base: 2
```

```
Enter smaller base: 1
Enter left side: 1
Enter right side: 1
Item was added
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete by criteria
5) Print
0) Exit
2
Enter side: 3
Enter smaller angle: 10
Item was added
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete by criteria
5) Print
0) Exit
3
Enter side: 3
Item was added
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete by criteria
5) Print
0) Exit
3
Enter side: 4
Item was added
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete by criteria
5) Print
```

```
0) Exit
3
Enter side: 5
Item was added
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete by criteria
5) Print
0) Exit
5
Side = 3,smaller_angle = 10,square = 1.56283,type: rhomb
Smaller base = 1,bigger base = 2,left side = 1,right side = 1,square = 1.86603,type:
trapeze
Sides =  3,square = 15.4843,type: pentagon
Sides =  4,square = 27.5276,type: pentagon
Sides =  5,square = 43.0119,type: pentagon

Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete by criteria
5) Print
0) Exit
1
Enter bigger base: 4
Enter smaller base: 3
Enter left side: 3
Enter right side: 3
Item was added
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete by criteria
5) Print
0) Exit
2
Enter side: 4
```

```
Enter smaller angle: 90
Item was added
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete by criteria
5) Print
0) Exit
3
Enter side: 8
Item was added
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete by criteria
5) Print
0) Exit
5
Smaller base = 3,bigger base = 4,left side = 3,right side = 3,square = 5.95804,type:
trapeze
Side = 4,smaller_angle = 90,square = 16,type: rhomb
Sides =  8,square = 110.111,type: pentagon

Side = 3,smaller_angle = 10,square = 1.56283,type: rhomb
Smaller base = 1,bigger base = 2,left side = 1,right side = 1,square = 1.86603,type:
trapeze
Sides =  3,square = 15.4843,type: pentagon
Sides =  4,square = 27.5276,type: pentagon
Sides =  5,square = 43.0119,type: pentagon

Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete by criteria
5) Print
0) Exit
4
Enter criteria
```

```
1) by type
2) lesser than square
1
1) trapeze
2) rhomb
3) pentagon
Enter type
1
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete by criteria
5) Print
0) Exit
5
Side = 4,smaller_angle = 90,square = 16,type: rhomb
Sides =  8,square = 110.111,type: pentagon

Side = 3,smaller_angle = 10,square = 1.56283,type: rhomb
Sides =  3,square = 15.4843,type: pentagon
Sides =  4,square = 27.5276,type: pentagon
Sides =  5,square = 43.0119,type: pentagon

Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete by criteria
5) Print
0) Exit
4
Enter criteria
1) by type
2) lesser than square
2
Enter square
20
Choose an operation:
1) Add trapeze
2) Add rhomb
```

```
3) Add pentagon
4) Delete by criteria
5) Print
0) Exit
5
Sides =  8,square = 110.111,type: pentagon

Sides =  4,square = 27.5276,type: pentagon
Sides =  5,square = 43.0119,type: pentagon

Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete by criteria
5) Print
0) Exit
0
```

Лабораторная работа №8

Задача: Используя структуры данных, разработанные для лабораторной работы №6 (контейнер 1-ого уровня и классы-фигуры) разработать алгоритм быстрой сортировки для класс-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 ипользовать механизмы:

- future
- packaged task/async

Для обеспечения потокобезопасности структур использовать механизмы:

- mutex
- lock quard

Программа должна позволять:

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

Фигуры: трапеция, ромб, пятиугольник.
Контейнер: связный список.

1 Описание

Параллельное программирование – это техника программирования, которая использует преимущества многоядерных или многопроцесорных компьютеров и является подмножеством более широкого понятия многопоточности (multithreading).
Параллельное программирование может быть сложным, но его легче понять, если считать его не “трудным“ , а просто “немного иным“. Оно включает в себя все черты более традиционного, последовательного программирования, но в параллельном прогаммировании имеются три дополнительных, четко определенных этапа:

- Определение параллелизма: анализ задачи с целью выделить подзадачи, которые могут выполняться одновременно.
- Выявление параллелизма: изменение структуры задачи таким образом, чтобы можно было эффективно выполнять подзадачи. Для этого часто требуется найти зависимости между подзадачами и организовать исходный код так, чтобы ими можно было эффективно управлять.
- Выражение параллелизма: реализация параллельного алгоритма в исходном коде с помощью системы обозначений параллельного программирования.

2 Исходный код

Описание классов фигур и методов класса-контейнера, определенных раннее, остается неизменным.

```
1
2  template <class T>
3  std::shared_ptr<TListItem<T>> TList<T>::PSort(std::shared_ptr<TListItem<T>> &head)
4  {
5      if (head == nullptr || head->GetNext() == nullptr) {
6          return head;
7      }
8
9      std::shared_ptr<TListItem<T>> partitionedEl = Partition(head);
10     std::shared_ptr<TListItem<T>> leftPartition = partitionedEl->GetNext();
11     std::shared_ptr<TListItem<T>> rightPartition = head;
12
13     partitionedEl->SetNext(nullptr);
14
15     if (leftPartition == nullptr) {
16         leftPartition = head;
17         rightPartition = head->GetNext();
18         head->SetNext(nullptr);
```

```
19      }
20
21      rightPartition = PSort(rightPartition);
22      leftPartition = PSort(leftPartition);
23      std::shared_ptr<TListItem<T>> iter = leftPartition;
24      while (iter->GetNext() != nullptr) {
25          iter = iter->GetNext();
26      }
27
28      iter->SetNext(rightPartition);
29
30      return leftPartition;
31  }
32
33  template <class T>
34  std::shared_ptr<TListItem<T>> TList<T>::Partition(std::shared_ptr<TListItem<T>> &head)
35  {
36      std::lock_guard<std::mutex> lock(mutex);
37      if (head->GetNext()->GetNext() = nullptr) {
38          if (head->GetNext()->GetFigure()->Square() > head->GetFigure()->Square()) {
39              return head->GetNext();
40          } else {
41              return head;
42          }
43      } else {
44          std::shared_ptr<TListItem<T>> i = head->GetNext();
45          std::shared_ptr<TListItem<T>> pivot = head;
46          std::shared_ptr<TListItem<T>> lastElSwapped = (pivot->GetNext()->GetFigure()->
                Square() >= pivot->GetFigure()->Square()) ? pivot->GetNext() : pivot;
47
48          while ((i != nullptr) && (i->GetNext() != nullptr)) {
49              if (i->GetNext()->GetFigure()->Square() >= pivot->GetFigure()->Square()) {
50                  if (i->GetNext() == lastElSwapped->GetNext()) {
51                      lastElSwapped = lastElSwapped->GetNext();
52                  } else {
53                      std::shared_ptr<TListItem<T>> tmp = lastElSwapped->GetNext();
54                      lastElSwapped->SetNext(i->GetNext());
55                      i->SetNext(i->GetNext()->GetNext());
56                      lastElSwapped = lastElSwapped->GetNext();
57                      lastElSwapped->SetNext(tmp);
58                  }
59              }
60              i = i->GetNext();
61          }
62          return lastElSwapped;
63      }
64  }
65
66  template <class T>
```

```
67  void TList<T>::Sort()
68  {
69      if (head == nullptr)
70          return;
71      std::shared_ptr<TListItem<T>> tmp = head->GetNext();
72      head->SetNext(PSort(tmp));
73  }
74
75  template <class T>
76  void TList<T>::ParSort()
77  {
78      if (head == nullptr)
79          return;
80      std::shared_ptr<TListItem<T>> tmp = head->GetNext();
81      head->SetNext(PParSort(tmp));
82  }
83
84  template <class T>
85  std::shared_ptr<TListItem<T>> TList<T>::PParSort(std::shared_ptr<TListItem<T>> &head)
86  {
87      if (head == nullptr || head->GetNext() == nullptr) {
88          return head;
89      }
90
91      std::shared_ptr<TListItem<T>> partitionedEl = Partition(head);
92      std::shared_ptr<TListItem<T>> leftPartition = partitionedEl->GetNext();
93      std::shared_ptr<TListItem<T>> rightPartition = head;
94
95      partitionedEl->SetNext(nullptr);
96
97      if (leftPartition == nullptr) {
98          leftPartition = head;
99          rightPartition = head->GetNext();
100         head->SetNext(nullptr);
101     }
102
103     std::packaged_task<std::shared_ptr<TListItem<T>>(std::shared_ptr<TListItem<T>>&)>
104         task1(std::bind(&TList<T>::PParSort, this, std::placeholders::_1));
105     std::packaged_task<std::shared_ptr<TListItem<T>>(std::shared_ptr<TListItem<T>>&)>
106         task2(std::bind(&TList<T>::PParSort, this, std::placeholders::_1));
107     auto rightPartitionHandle = task1.get_future();
108     auto leftPartitionHandle = task2.get_future();
109
110     std::thread(std::move(task1), std::ref(rightPartition)).join();
111     rightPartition = rightPartitionHandle.get();
112     std::thread(std::move(task2), std::ref(leftPartition)).join();
113     leftPartition = leftPartitionHandle.get();
114     std::shared_ptr<TListItem<T>> iter = leftPartition;
115     while (iter->GetNext() != nullptr) {
```

```
116         iter = iter->GetNext();
117     }
118 
119     iter->SetNext(rightPartition);
120     return leftPartition;
121 }
```

3 Консоль

```
karma@karma:~/mai_study/OOP/lab8$ ./run
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Sort list
6) Print list
0) Exit
3
Enter side: 5
Enter index = 0
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Sort list
6) Print list
0) Exit
2
Enter side: 5
Enter smaller angle: 30
Enter index = 0
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Sort list
6) Print list
0) Exit
```

```
1
Enter bigger base: 5
Enter smaller base: 4
Enter left side: 4
Enter right side: 4
Enter index = 1
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Sort list
6) Print list
0) Exit
3
Enter side: 10
Enter index = 2
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Sort list
6) Print list
0) Exit
6
idx: 0   Side = 5,smaller_angle = 30,square = 12.5,type: rhomb

idx: 1   Sides =  5,square = 43.0119,type: pentagon

idx: 2   Smaller base = 4,bigger base = 5,left side = 4,right side = 4,square
= 7.96863,type: trapeze

idx: 3   Sides =  10,square = 172.048,type: pentagon


Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
```

```
5) Sort list
6) Print list
0) Exit
5
1 to regular sort,2 to parallel
2
idx: 0   Smaller base = 4,bigger base = 5,left side = 4,right side = 4,square
= 7.96863,type: trapeze

idx: 1   Side = 5,smaller_angle = 30,square = 12.5,type: rhomb

idx: 2   Sides =  5,square = 43.0119,type: pentagon

idx: 3   Sides =  10,square = 172.048,type: pentagon


Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Sort list
6) Print list
0) Exit
6
idx: 0   Smaller base = 4,bigger base = 5,left side = 4,right side = 4,square
= 7.96863,type: trapeze

idx: 1   Side = 5,smaller_angle = 30,square = 12.5,type: rhomb

idx: 2   Sides =  5,square = 43.0119,type: pentagon

idx: 3   Sides =  10,square = 172.048,type: pentagon

Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Sort list
6) Print list
```

```
0) Exit
2
Enter side: 50
Enter smaller angle: 90
Enter index = 0
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Sort list
6) Print list
0) Exit
3
Enter side: 10000
Enter index = 0
Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Sort list
6) Print list
0) Exit
6
idx: 0   Sides =  10000,square = 1.72048e+08,type: pentagon

idx: 1   Side = 50,smaller_angle = 90,square = 2500,type: rhomb

idx: 2   Smaller base = 4,bigger base = 5,left side = 4,right side = 4,square
= 7.96863,type: trapeze

idx: 3   Side = 5,smaller_angle = 30,square = 12.5,type: rhomb

idx: 4   Sides =  5,square = 43.0119,type: pentagon

idx: 5   Sides =  10,square = 172.048,type: pentagon

Choose an operation:
1) Add trapeze
2) Add rhomb
```

```
3) Add pentagon
4) Delete figure from list
5) Sort list
6) Print list
0) Exit
5
1 to regular sort,2 to parallel
1
idx: 0   Smaller base = 4,bigger base = 5,left side = 4,right side = 4,square
= 7.96863,type: trapeze

idx: 1   Side = 5,smaller_angle = 30,square = 12.5,type: rhomb

idx: 2   Sides =  5,square = 43.0119,type: pentagon

idx: 3   Sides =  10,square = 172.048,type: pentagon

idx: 4   Side = 50,smaller_angle = 90,square = 2500,type: rhomb

idx: 5   Sides =  10000,square = 1.72048e+08,type: pentagon


Choose an operation:
1) Add trapeze
2) Add rhomb
3) Add pentagon
4) Delete figure from list
5) Sort list
6) Print list
0) Exit
0
```

Лабораторная работа №9

Задача: Используя структуры данных, разработанные для лабораторной работы №6 (контейнер 1-ого уровня и классы-фигуры) необходимо разработать:

- Контейнер второго уровня с использованием шаблонов.
- Реализовать с помощью лямбда-выражений набор команд, совершающих операции над контейнером 1-ого уровня: генерация фигур со случайными значениями 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 конейнеру первого уровня.

Для создания потоков ипользовать механизмы:

- future
- packaged task/async

Для обеспечения потокобезопасности структур использовать механизмы:

- mutex
- lock quard

Фигуры: трапеция, ромб, пятиугольник.
Контейнер 1-ого уровня: связный список.
Контейнер 2-ого уровня: стек.

1 Описание

Лямбда-выражение – 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. В итоге, мы получаем крайне удобную конструкцию, которая позволяет сделать код более лаконичным и устойчивым к изменениям.
Непосредственное объявление лямбда-функции состоит из трех частей. Первая часть (квадратные скобки) позволяет привязывать переменные, доступные в текущей области видимости. Вторая часть (круглые скобки) указывает список принимаемых параметров лямбда-функции. Третья часть (фигурные скобки) содержит тело лямбда-функции.
В настоящее время, учитывая, что достигли практически потолка по тактовой частоте и дальше идет рост количества ядер, появился запрос на параллелизм. В результате снова в моде стал функциональный подход, так как он очень хорошо работает в условиях параллелизма и не требует явных синхронизаций. Поэтому сейчас усиленно думают, как задействовать растущее число ядер процессора и как обеспечить автоматическое распараллеливание. А в функциональном программировании практически основа всего – лямбда. Учитывая, что функциональные языки переживают второе рождение, было бы странным, если бы функциональный подход не добавляли во все популярные языки. C++ – язык, поддерживающий много парадигм, поэтому нет ничего странного в использовании лямбда-функций и лямбда-выражений в нем.

2 Исходный код

Описание классов фигур и классов-контейнеров, определенных раннее, остается неизменным.

```
1
2  int main(void)
3  {
4      TList<Figure> list;
5      typedef std::function<void(void)> Command;
6      TStack<std::shared_ptr<Command>> stack;
7      std::mutex mtx;
8
9      Command cmdInsert = [&]() {
10         std::lock_guard<std::mutex> guard(mtx);
11
12         uint32_t seed = std::chrono::system_clock::now().time_since_epoch().count();
```

```
13
14          std::default_random_engine generator(seed);
15          std::uniform_int_distribution<int> distFigureType(1, 3);
16          std::uniform_int_distribution<int> distFigureParam(1, 10);
17          for (int i = 0; i < 10; ++ i) {
18              std::cout << "Command: Insert" << std::endl;
19
20              switch(distFigureType(generator)) {
21                  case 1: {
22                      std::cout << "Inserted trapeze" << std::endl;
23
24                      int32_t big_base = distFigureParam(generator);
25                      int32_t small_base = distFigureParam(generator);
26                      int32_t l_side = distFigureParam(generator);
27                      int32_t r_side = distFigureParam(generator);
28
29                      list.PushFirst(std::shared_ptr<Trapeze>(new Trapeze(small_base,
                            big_base, l_side, r_side)));
30
31                      break;
32                  }
33
34                  case 2: {
35                      std::cout << "Inserted rhomb" << std::endl;
36
37                      int32_t side = distFigureParam(generator);
38                      int32_t small_angle = distFigureParam(generator);
39
40                      list.PushFirst(std::shared_ptr<Rhomb>(new Rhomb(side, small_angle)))
                            ;
41
42                      break;
43                  }
44
45                  case 3: {
46                      std::cout << "Inserted pentagon" << std::endl;
47
48                      int32_t side = distFigureParam(generator);
49
50                      list.PushFirst(std::shared_ptr<Pentagon>(new Pentagon(side)));
51
52                      break;
53                  }
54              }
55          }
56      };
57
58
59      Command cmdRemove = [&]() {
```

```
60          std::lock_guard<std::mutex> guard(mtx);
61
62          std::cout << "Command: Remove" << std::endl;
63
64          if (list.IsEmpty()) {
65              std::cout << "List is empty" << std::endl;
66          } else {
67              uint32_t seed = std::chrono::system_clock::now().time_since_epoch().count()
                    ;
68
69              std::default_random_engine generator(seed);
70              std::uniform_int_distribution<int> distSquare(1, 150);
71              double sqr = distSquare(generator);
72              std::cout << "Lesser than " << sqr << std::endl;
73
74              for (int32_t i = 0; i < 10; ++i) {
75                  auto iter = list.begin();
76                  for (int32_t k = 0; k < list.GetLength(); ++k) {
77                      if (iter->Square() < sqr) {
78                          list.Pop(k);
79                          break;
80                      }
81                      ++iter;
82                  }
83              }
84          }
85      };
86
87      Command cmdPrint = [&]() {
88          std::lock_guard<std::mutex> guard(mtx);
89
90          std::cout << "Command: Print" << std::endl;
91          if(!list.IsEmpty()) {
92              std::cout << list << std::endl;
93          } else {
94              std::cout << "List is empty." << std::endl;
95          }
96      };
97
98      stack.Push(std::shared_ptr<Command>(&cmdPrint, [](Command*){}));
99      stack.Push(std::shared_ptr<Command>(&cmdRemove, [](Command*){}));
100     stack.Push(std::shared_ptr<Command>(&cmdPrint, [](Command*){}));
101     stack.Push(std::shared_ptr<Command>(&cmdInsert, [](Command*){}));
102
103     while (!stack.IsEmpty()) {
104         std::shared_ptr<Command> cmd = stack.Top();
105         std::future<void> ft = std::async(*cmd);
106         ft.get();
107         stack.Pop();
```

```
108     }
109
110     return 0;
111 }
```

3 Консоль

```
karma@karma:~/mai_study/OOP/lab9$ ./run
Command: Insert
Inserted trapeze
Command: Insert
Inserted trapeze
Command: Insert
Inserted pentagon
Command: Insert
Inserted rhomb
Command: Insert
Inserted rhomb
Command: Insert
Inserted pentagon
Command: Insert
Inserted trapeze
Command: Insert
Inserted trapeze
Command: Insert
Inserted rhomb
Command: Insert
Inserted trapeze
Command: Print
idx: 0   Smaller base = 6,bigger base = 9,left side = 10,right side = 10,square
= 16.8869,type: trapeze

idx: 1   Side = 3,smaller_angle = 2,square = 0.314095,type: rhomb

idx: 2   Smaller base = 7,bigger base = 9,left side = 5,right side = 8,square
= -1,type: trapeze

idx: 3   Smaller base = 6,bigger base = 9,left side = 4,right side = 3,square
= 10.1225,type: trapeze
```

```
idx: 4   Sides =  6,square = 61.9372,type: pentagon

idx: 5   Side = 1,smaller_angle = 1,square = 0.0174524,type: rhomb

idx: 6   Side = 2,smaller_angle = 2,square = 0.139598,type: rhomb

idx: 7   Sides =  6,square = 61.9372,type: pentagon

idx: 8   Smaller base = 2,bigger base = 10,left side = 3,right side = 4,square
= -1,type: trapeze

idx: 9   Smaller base = 1,bigger base = 8,left side = 8,right side = 5,square
= 9.2915,type: trapeze


Command: Remove
Lesser than 16
Command: Print
idx: 0   Smaller base = 6,bigger base = 9,left side = 10,right side = 10,square
= 16.8869,type: trapeze

idx: 1   Sides =  6,square = 61.9372,type: pentagon

idx: 2   Sides =  6,square = 61.9372,type: pentagon
```

1 Выводы

В этом семестре я познакомилась с новой объектно-ориентированной парадигмой программирования. Это методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования. Курс представлял собой полноценный проект, который мы создавали на протяжении семестра. Каждая лабораторная работа была связана с предыдущей, что позволяло находить и исправлять недочеты прошлых версий. Эта практика очень полезна, так оно позволяет нам учиться работать “на длинных дистанциях“. Мною было сделано 9 лабораторных работ:

1. В первой работе я познакомилась с базовыми понятиями ООП, такими как наследование, полиморфизм и инкапсуляция. Было спроектированы классы фигур, заданные вариантом, в которых использовались перегруженные операторы, дружественные функции и операции ввода-вывода из стандартных библиотек.

2. Во второй работе я спроектировала динамическую структуру данных – список. Объекты передаются в методы контейнера по значению.

3. В третьей работе я познакомилась с технологией умных указателей. контейнер был переписан с использованием shared ptr.

4. В четвертой работе я узнала о шаблонах классов и их проектировании. Был построен шаблон динамической структуры данных.

5. В пятой работе у моего контейнера появились итераторы, что облегчило перемещение по списку.

6. В шестой работе был создан аллокатор памяти, помогающий оптимизировать выделение и освобождение памяти. Свободные блоки хранятся в аллокаторе в контейнере второго уровня – стеке.

7. В седьмой работе был запрограммирован контейнер в контейнере. То есть в контейнере первого уровня хранятся контейнеры второго уровня, внутри которых хранятся фигуры. Фигуры внутри контейнера второго уровня отсортированы по возрастанию площади. Если в ходе выполнения программы контейнер второго уровня полностью освобождается от фигур, то он удаляется из контейнера первого уровня.

8. В восьмой работе я познакомилась с параллельным программированием, что позволило осуществлять быструю сортировку несколько иным образом нежели

в классическом варианте: все рекурсивные вызовы теперь выполняются каждый в своем потоке.

9. В девятой работе я узнала о лямбда-выражениях. Теперь действия над контейнером первого уровня генерируются в виде команд, которые помещаются в контейнер второго уровня. В рамках данной работы это было еще полезно и как тестирование: можно было убедиться, что контейнер первого уровня работает корректно.

В итоге выполнения этого проекта я получила хорошие навыки программирования и проектирования на C++. Освоила ряд возможностей языка, которые уже к данному моменту понадобились мне при выполнении работ в других курсах. Думаю, что изучение этой парадигмы совершенно точно необходимо каждому современному программисту вне зависимости от того, чем он занимается. Поэтому нельзя останавливаться на достигнутом и нужно продолжить изучение ООП и C++ в частности.