Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторные работы по курсу Объектно-ориентированное программирование

Студент: Баранов А.А.
Преподаватель: Поповкин А.В.
Группа: М8о-207Б
Вариант: 6
Дата:
Подпись:

Москва, 2017

Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №1 по курсу Объектно-ориентированное программирование

Студент: Баранов А.А.
Преподаватель: Поповкин А.В.
Группа: М8о-207Б
Вариант: 6
Дата:
Оценка:
Подпись:

Москва, 2017

Лабораторная работа №1
Задача: Необходимо спроектировать и запрограммировать на языке C++ классы фигур, согласно вариантов 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.
- Должный иметь общий виртуальный метод расчета площади фигуры – Square.
- Должны иметь конструктор, считывающий значения основных параметров фигуры из стандартного потока cin.
- Должны быть расположенны в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp)

Фигуры: правильные пятиугольник, шестиугольник, восьмиугольник.

1 Описание

Абстракция данных - Абстрагирование означает выделение значимой информации и исключение из рассмотрения незначимой. В ООП рассматривают лишь абстракцию данных (нередко называя её просто "абстракцией"), подразумевая набор значимых характеристик объекта, доступный остальной программе.

Инкапсуляция - свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе. Одни языки (например, C++, Java или Ruby) отождествляют инкапсуляцию с сокрытием, но другие (Smalltalk, Eiffel, OCaml) различают эти понятия.

Наследование - свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью. Класс, от которого производится наследование, называется базовым, родительским или суперклассом. Новый класс потомком, наследником, дочерним или производным классом.

Полиморфизм подтипов (в ООП называемый просто "полиморфизмом") свойство системы, позволяющее использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта. Другой вид полиморфизма параметрический в ООП называют обобщённым программированием.

Класс - универсальный, комплексный тип данных, состоящий из тематически единого набора "полей"(переменных более элементарных типов) и "методов"(функций для работы с этими полями), то есть он является моделью информационной сущности с внутренним и внешним интерфейсами для оперирования своим содержимым (значениями полей). В частности, в классах широко используются специальные блоки из одного или чаще двух спаренных методов, отвечающих за элементарные операции с определенным полем (интерфейс присваивания и считывания значения), которые имитируют непосредственный доступ к полю. Эти блоки называются "свойствами"и почти совпадают по конкретному имени со своим полем (например, имя поля может начинаться со строчной, а имя свойства с заглавной буквы). Другим проявлением интерфейсной природы класса является то, что при копировании соответствующей переменной через присваивание, копируется только интерфейс, но не сами данные, то есть класс ссылочный тип данных. Переменная-объект, относящаяся к заданному классом типу, называется экземпляром этого класса. При этом в некоторых исполняющих системах класс также может представляться некоторым объектом при выполнении программы посредством динамической идентификации типа данных. Обычно классы разрабатывают таким образом, чтобы обеспечить отвечающие природе объ-

екта и решаемой задаче целостность данных объекта, а также удобный и простой интерфейс. В свою очередь, целостность предметной области объектов и их интерфейсов, а также удобство их проектирования, обеспечивается наследованием.

Объект - сущность в адресном пространстве вычислительной системы, появляющаяся при создании экземпляра класса (например, после запуска результатов компиляции и связывания исходного кода на выполнение).

2 Исходный код

figure_pentagon.cpp

Pentagon(std::istream &is); - Ввод из потока std::istream
Pentagon(); - Конструктор класса
double Square() override; - Получение площади
void Print() override; - Печать фигуры
virtual Pentagon(); - Деструктор класса

figure_hexagon.cpp

Hexagon(std::istream &is); - Ввод из потока std::istream
Hexagon(); - Конструктор класса
double Square() override; - Получение площади
void Print() override; - Печать фигуры
virtual Hexagon(); - Деструктор класса

figure_octagon.cpp

Octagon(std::istream &is); - Ввод из потока std::istream
Octagon(); - Конструктор класса
double Square() override; - Получение площади
void Print() override; - Печать фигуры
virtual Octagon(); - Деструктор класса

```
class Pentagon : public Figure {
        public:
            Pentagon();
            Pentagon(std::istream &is);
            double Square();
            void Print();
            virtual  Pentagon();
        private:
            double side;
};
```

```
class Hexagon : public Figure{
        public:
            Hexagon ();
            Hexagon(std::istream &is);
            double Square() override;
            void Print() override;
            virtual  Hexagon();
        private:
            double side;
            short int number;
};
class Octagon : public Figure{
        public:
            Octagon();
            Octagon(std::istream &is);
            double Square() override;
            void Print() override;
            virtual  Octagon();
        private:
            double side;
            short int number;
};
```

3 Вывод консоли

```
C: hello_world>main
Write number of sides and their length. This polygon must be regular.
Number:
5
Length:
10
-Pentagon has created.-
Length of side is equal: 10. Number of sides is 5.
Square is: 337.14
-Pentagon was deleted.-

C: hello_world C++>main
Write number of sides and their length. This polygon must be regular.
Number:
6
Length:
12
-Hexagon has created.-
Length of side is equal: 12. Number of sides is 6.
Square is: 374.123
```

-Hexagon was deleted.-

C: hello_world C++>main
Write number of sides and their length. This polygon must be regular.
Number:
8
Length:
14
-Octagon has created.-
Length of side is equal: 14. Number of sides is 8.
Square is: 946.372
-Octagon was deleted.-

4 Выводы

Благодаря данной лабораторной работе я познакомился с объектно-ориентированным программированием, вспомнил синтаксис C++. Необходимо было реализовать классы трех фигур, и не смотря на простоту задачи, она несет в себе все принципы ООП: класс, объект, инкапсуляция, абстрактный тип данных, наследование, полиморфизм. Выполнив задание, я стал свободнее ориентироваться в написании программ C++ с классами и усвоил важные понятия по работы=е с объектами.

Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу Объектно-ориентированное программирование

Студент: Баранов А.А.
Преподаватель: Поповкин А.В.
Группа: М8о-207Б
Вариант: 6
Дата:
Оценка:
Подпись:

Москва, 2017

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 ссыл-ке).
- 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 контей-нера.
- Классы должны быть расположены в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

Фигура: правильный пятиугольник.
Контейнер: массив.

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 отдель-ные 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 ново-го 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 осуществляет-ся 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 ка-честве 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 воз-действуют 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

TVector.cpp

TVector(int size); - Конструктор класса
bool Empty(); - Проверка на пустоту
int Size(); - Получение размера
void Print_node(int i); - Печать элемента
void Add(Pentagon element); - Добавление элемента
void Resize(int new_size); - Переопределение размера
void Delete_node(int i); - Удаление элемента
void Destroy(); - Удаление вектора
virtual ~TVector(); - Деструктор класса

TVectorItem.cpp

TVectorItem(Pentagon& pentagon); - Конструктор класса
TVectorItem(const TVectorItem& orig); - Конструктор класса
GetPentagon() const; - Получение пентагона
~TVectorItem(); - Деструктор класса

figure_pentagon.cpp

Pentagon(const Pentagon& orig); - Конструктор класса
operator«(std::ostream& os, const TVectorItem& obj); - Перегруженный оператор вывода
operator»(std::istream& is, Pentagon& obj); - Перегруженный оператор ввода

```
class TVector {
        private:
            int size;
            TVectorItem data;
        public:
            TVector(int size);
            bool Empty();
            int Size();
            void Print_node(int i);
            void Add(Pentagon element);
            void Resize(int new_size);
            void Delete_node(int i);
            void Destroy();
            virtual ~TVector();
};
class TVectorItem {
        private:
            Pentagon pentagon;
```

```
        public:
            TVectorItem(Pentagon& pentagon);
            TVectorItem(const TVectorItem& orig);
            friend std::ostream& operator«(std::ostream& os, const TVectorItem&
obj);
            Pentagon GetPentagon() const;
            virtual ~TVectorItem();
};
class Pentagon {
        public:
            Pentagon();
            Pentagon(std::istream &is);
            Pentagon(const Pentagon& orig);
            double Square();
            void Print();
            friend std::ostream& operator«(std::ostream& os, const Pentagon& obj);
            friend std::istream& operator»(std::istream& is, Pentagon& obj);
            virtual ~Pentagon();
        private:
            double side;
};
```

3 Вывод консоли

Code 1 means add element. Code 2 means print element. Code 3 means delete element. Code 4 means break.
Code:
1
Write length of side. This pentagon must be regular.
Length:
10
-Pentagon was created.-
Pentagon copy created
Vector item: created
-Pentagon was deleted.-
Code:
2
You can print elements. Write it's index.
0
Length of side is equal: 10.
Code:
3
You can delete elements. Write index.
0
Element was deleted, size of array was reduced.
Code:

2
You can print elements. Write it's index.
0
Incorrect index.
Code:
4

4 Выводы

Основной задачей этой работы было написание своей структуры данных (у меня вектор). Практика подобного программирования имеет положительное действие, т.к. обучающийся не бездумно использует библиотечные контейнеры, а осмысленно пишет свои, что очень важно на раннем этапе обучения. Свой вектор я реализовал на динамичеком массиве, написав также всю необходимую "обертку"для работы с ним и его элементами. В дальнейшем корректная реализация необходимой СД будет иметь ключевое значение.

Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу Объектно-ориентированное программирование

Студент: Баранов А.А.
Преподаватель: Поповкин А.В.
Группа: М8о-207Б
Вариант: 6
Дата:
Оценка:
Подпись:

Москва, 2017

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 контей-нера.
- Классы должны быть расположены в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

Фигуры: правильные пятиугольник, шестиугольник, восьмиугольник.
Контейнер: массив.

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 со-хранить 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

TVector.cpp

std::shared_ptr<TVectorItem> data; - добавлено использование умных указателей

std::shared_ptr<Pentagon> pentagon; - добавлено использование умных указателей

TVectorItem.cpp

TVectorItem(std::shared_ptr<Pentagon>& pentagon); - добавлено использование умных указателей

std::shared_ptr<Pentagon> GetPentagon() const; - добавлено использование умных указателей

```
class TVector {
        private:
                int size;
                std::shared_ptr<TVectorItem> data;
        public:
                TVector(int size);
                bool Empty();
                int Size();
                void Print_node(int i);
                void Add(std::shared_ptr<Pentagon>& element);
                void Resize(int new_size);
                void Delete_node(int i);
                void Destroy();
                virtual ~TVector();
};
class TVectorItem {
        private:
                std::shared_ptr<Pentagon> pentagon;
        public:
                TVectorItem(std::shared_ptr<Pentagon>& pentagon);
                friend std::ostream& operator«(std::ostream& os, const TVectorItem&
obj);
                std::shared_ptr<Pentagon> GetPentagon() const;
                virtual ~TVectorItem();
};
```

3 Вывод консоли

C: hello_world C++ OOP3>OOP3
Code 1 means add element. Code 2 means print element. Code 3 means delete element. Code 4 means break.
Code:
1
Write number of sides:
5
Write length of side. This polygon must be regular.
Length:
10
Code:
1
Write number of sides:
6
Write length of side. This polygon must be regular.
Length:
20
Code:

1
Write number of sides:
8
Write length of side. This polygon must be regular.
Length:
14
Code:
2
You can print elements. Write it's index.
2
Length of side is equal: 14. Number of sides is 8.
Code:
3
You can delete elements. Write index.
0
-Element was deleted-
Code:
4

4 Выводы

Выполнив эту работу, я освоил такой важный инструмент, как умный указатель. Зжесь мы использовали "shared_ptr данный подвид умных указателей является основным и его внутренняя работа заключается в подсчете ссылок на объект и его удалении при равенстве счетчика нулю. С умными указателями можно работать как с обычными, что очень удобно, однако есть несколько специальных функций, одна из них - "make_shared". Она возвращает умный указатель на объект, т.е. по сути создает его (указатель).

Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №4 по курсу Объектно-ориентированное программирование

Студент: Баранов А.А.
Преподаватель: Поповкин А.В.
Группа: М8о-207Б
Вариант: 6
Дата:
Оценка:
Подпись:

Москва, 2017

Лабораторная работа №4
Задача: Необходимо спроектировать и запрограммировать на языке C++ шаблон класса-контейнера первого уровня, содержащий все три фигуры класса фигуры, согласно вариантов задания (реализованную в ЛР1).
Классы должны удовлетворять следующим правилам:

- Требования к классам фигуры аналогичны требованиям из лабораторной ра-боты 1.
- Шаблон класса-контейнера должен соджержать объекты используя std:sharedptr<. . . >.
- Шаблон класса-контейнера должен иметь метод по добавлению фигуры в кон-тейнер.
- Шаблон класса-контейнера должен иметь методы по получению фигуры из контейнера (опередеяется структурой контейнера).
- Шаблон класса-контейнера должен иметь метод по удалению фигуры из кон-тейнера (опередеяется структурой контейнера).
- Шаблон класса-контейнера должен иметь перегруженный оператор по выводу контейнера в поток std::ostream («).
- Шаблон класса-контейнера должен иметь деструктор, удаляющий все элемен-ты контейнера.
- Классы должны быть расположенны в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

Фигуры: правильные пятиугольник, шестиугольник, восьмиугольник.
Контейнер: массив.

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, пере-числяемый тип, указатель на любой объект с глобально доступным именем, ссылка). Шаблоны используются в случаях дублирования одного и того же кода для несколь-ких 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 типобез-опасных классов. Иногда рекомендуется использовать шаблоны вместо макросов C и пустых указателей. Шаблоны особенно полезны при работе с коллекциями и умными указателями.

2 Исходный код

TVector.cpp

std::shared_ptr<TVectorItem<T»* data; - добавлено использование шаблонов классов

void Add(std::shared_ptr<T>& element); - добавлено использование шаблонов классов

TVectorItem.cpp

std::shared_ptr<T> figure; - добавлено использование шаблонов классов

TVectorItem(std::shared_ptr<T>& figure); - добавлено использование шаблонов классов

template <class A> friend std::ostream& operator«(std::ostream& os, const std::shared_ptr<TVectorItem<A»& obj); - добавлено использование шаблонов классов

std::shared_ptr<T> GetFigure() const; - добавлено использование шаблонов классов

```
template <class T> class TVector {
        private:
                int size;
                std::shared_ptr<TVectorItem<T»* data;
        public:
                TVector(int size);
                bool Empty();
                int Size();
                void Print_node(int i);
                void Add(std::shared_ptr<T>& element);
                void Resize(int new_size);
                void Delete_node(int i);
                void Destroy();
                virtual  TVector();
};
template <class T> class TVectorItem {
        private:
                std::shared_ptr<T> figure;
        public:
                TVectorItem(std::shared_ptr<T>& figure);
                template <class A> friend std::ostream& operator«(std::ostream& os,
const std::shared_ptr<TVectorItem<A»& obj);
                std::shared_ptr<T> GetFigure() const;
                virtual  TVectorItem(); };
```

3 Вывод консоли

```
C: hello_world C++ OOP4>OOP4
Code 1 means add element. Code 2 means print element. Code 3 means delete element.
Code 4 means break.
Code:
1
Write number of sides:
5
Write length of side. This polygon must be regular.
Length:
10
Code:
1
Write number of sides:
6
Write length of side. This polygon must be regular.
Length:
20
Code:
1
```

Write number of sides:
8
Write length of side. This polygon must be regular.
Length:
14
Code:
2
You can print elements. Write it's index.
2
Length of side is equal: 14. Number of sides is 8.
Code:
3
You can delete elements. Write index.
0
-Element was deleted-
Code:
4

4 Выводы

Данная лабораторная работа помогла разобраться с шаблонами классов. Необходимо было осуществить с их помощью коррекную работу контейнера первого уровня с тремя разными элементами, что вполне удалось. Стоит отметить, что шаблоны сильно помогают сделать код независимым и универсальным, так некоторые программы из одного проекта могут быть использованы либо с минимальными изменениями, либо вообще без них. Реализовав тот же вектор с помощью щаблонов, можно использовать его дальше где угодно.

Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №5 по курсу Объектно-ориентированное программирование

Студент: Баранов А.А.
Преподаватель: Поповкин А.В.
Группа: М8о-207Б
Вариант: 6
Дата:
Оценка:
Подпись:

Москва, 2017

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: правильные пятиугольник, шестиугольник, восьмиугольник.
Контейнер: массив.

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 воз-можности 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

TIterator.hpp

std::shared_ptr<node> node_ptr; - умный указатель на элемент вектора

TIterator(std::shared_ptr<node> n); - конструктор класса

std::shared_ptr<T> operator* (); - перегруженный оператор разыменования указателя

std::shared_ptr<T> operator-> (); - перегруженный оператор разыменования указателя

void operator++ (); - перегруженный оператор инкрементирования

TIterator operator++ (int); - перегруженный оператор инкрементирования, возвращающий итерирюемый элемент

bool operator== (TIterator const& i); - перегруженный оператор равенства

bool operator!= (TIterator const& i); - перегруженный оператор неравенства

```
template <class node, class T> class TIterator {
        private:
                std::shared_ptr<node> node_ptr;
        public:
                TIterator(std::shared_ptr<node> n) {
                        node_ptr = n;
                }
                std::shared_ptr<T> operator* () {
                        return node_ptr->GetFigure();
                }
                std::shared_ptr<T> operator-> () {
                        return node_ptr->GetFigure();
                }
                void operator++ () {
                        node_ptr = node_ptr->GetNext();
                }
                TIterator operator++ (int) {
                        TIterator iter(*this);
                        ++(*this);
                        return iter;
                }
                bool operator== (TIterator const& i) {
                        return node_ptr == i.node_ptr;
                }
                bool operator!= (TIterator const& i) {
                        return !(*this == i);
                }
};
```

3 Вывод консоли

anton@ubuntu: /Documents/OOP5$ ./OOP5 Code 1 == add element. Code 2 == print element. Code 3 == delete element. Code 4 == print all array. Code 0 == break.
Code:
1
Write number of sides:
5
Write length of side. This polygon must be regular.
Length:
10
-Pentagon was created.-
Vector item: created
Code:
1
Write number of sides:
6

Write length of side. This polygon must be regular.
Length:
20
-Hexagon was created.-
Vector item: created
Code:
1
Write number of sides:
8
Write length of side. This polygon must be regular.
Length:
30
-Octagon was created.-
Vector item: created
Code:
4
Element is pentagon. Length of side is equal: 10. Square is 172.048.
Element is hexagon. Length of side is equal: 20. Square is 1039.23.
Element is octagon. Length of side is equal: 30. Square is 4345.58.
Code:
0
Vector was deleted.

4 Выводы

В этой лабораторной работе необходимо было написать библиотечный файл - итератор, благодаря которому можно осуществлять проход по всем элементам контейнера. Так как у меня вектор, то итератор начинает с первого элемента и заканчивает последним. Итератор экономит место в коде основной программы за счет замены циклов одной строчкой for(auto i : vector) std::cout « *i « std::endl;. К тому же итераторы можно писать достаточно разнообразные, т.е. чтобы они не обходили контейнер определенным образом, что может быть полезно при определенных задачах. Также стоит отметить, что данный итератор не зависит от типа итерируемых элементов и работает с 5ти, 6ти и 8ми -уголниками.

Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №6 по курсу Объектно-ориентированное программирование

Студент: Баранов А.А.
Преподаватель: Поповкин А.В.
Группа: М8о-207Б
Вариант: 6
Дата:
Оценка:
Подпись:

Москва, 2017

Лабораторная работа №6
Задача: Используя структуры данных, разработанные для предыдущей лаборатор-ной работы (ЛР №5) спроектировать и разработать аллокатор памяти для динами-ческой структуры данных.

Цель построения аллокатора – минимизация вызова операции malloc. Аллокатор дол-жен 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 хране-ния 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: правильные пятиугольник, шестиугольник, восьмиугольник.
Контейнер 1-ого уровня: массив.
Контейнер 2-ого уровня: стек.

1 Описание

Аллокатор памяти – часть программы (как прикладной, так и операционной систе-мы), 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, мо-гут 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

TAllocationBlock.cpp

TAllocationBlock(int32_t size, int32_t count); - конструктор класса
void *Allocate(); - функция аллокации
void Deallocate(void *pointer); - функция деаллокации
bool Empty(); - проверка на пустоту
int32_t Size(); - взятие размера
virtual TAllocationBlock(); - деструктор класса

TStack.cpp

TStack();- конструктор класса
virtual TStack();- деструктор класса
void Push(const T &item); - добавление элемента
void Pop(); - изъятие элемента
bool IsEmpty() const; - проверка на пустоту
uint32_t GetSize() const; - получение размера
operator«(std::ostream &os, const TStack<A> &stack); - переопределенный оператор вывода

TStackItem.cpp

TStackItem(const T &val, TStackItem<T> *item);- конструктор класса
virtual TStackItem();- деструктор класса
void Push(const T &val);- добавление элемента
void SetNext(TStackItem<T> *item); - установление следующего по порядку
TStackItem<T> &GetNext() const; - получение следующего по порядку

```
class TAllocationBlock {
        public:
            TAllocationBlock(int32_t size, int32_t count);
            void *Allocate();
            void Deallocate(void *pointer);
            bool Empty();
            int32_t Size();
            virtual TAllocationBlock();
        private:
            unsigned char *_used_blocks;
            TStack<void *>_free_blocks;
};
template <class T> class TStack public: TStack(); virtual TStack();
            void Push(const T &item); void Pop(); bool IsEmpty() const; uint32_t GetSize() const;
            template <class A> friend std::ostream& operator«(std::ostream &os, const TStack<A> &stack);
        private: TStackItem<T> *head; uint32_t count; ;
template <class T> class TStackItem public: TStackItem(const T &val, TStackItem<T> *item); virtual TStackItem();
            void Push(const T &val); void SetNext(TStackItem<T> *item); TStackItem<T> &GetNext() const;
        private: T *value; TStackItem<T> *next; ;
```

3 Вывод консоли

```
anton@ubuntu: /Documents/OOP6$ ./OOP6
Memory init.
Code 1 == add element. Code 2 == print element. Code 3 == delete element. Code 4 == print all array. Code 0 == break.
Code:
1
Write number of sides:
5
Write length of side. This polygon must be regular.
Length:
5
```

-Pentagon was created.-
Vector item: created
Code:
1
Write number of sides:
6
Write length of side. This polygon must be regular.
Length:
20
-Hexagon was created.-
Vector item: created
Code:
1
Write number of sides:
8
Write length of side. This polygon must be regular.
Length:
10
-Octagon was created.-
Vector item: created
Code:
4
Element is pentagon. Length of side is equal: 5. Square is 43.0119.
Element is hexagon. Length of side is equal: 20. Square is 1039.23.
Element is octagon. Length of side is equal: 10. Square is 482.843.
Code:
0
Vector was deleted.
Memory freed.

4 Выводы

Задачей данной лабораторной работы было написание аллокатора памяти для контейнера 1го уровня с использованием контейнера 2го уровня. Реализовав её, я укрепил свои навыки программирования C++ и разобрался с достаточно важной темой, ведь написание аллокаторов сильно помогает увеличить контроль за памятью. Вся память выделяется именно аллокатором и именно таким образом, какой мы задали, это минимизирует вызовы функций malloc и free, а так же new и delete, которые мы переопределили для корректной работы программы.

Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №7 по курсу Объектно-ориентированное программирование

Студент: Баранов А.А.
Преподаватель: Поповкин А.В.
Группа: М8о-207Б
Вариант: 6
Дата:
Оценка:
Подпись:

Москва, 2017

Лабораторная работа №7
Задача: Необходимо реализовать динамическую структуру данных – "Хранилище объектов"и алгоритм работы с ней. "Хранилище объектов"представляет собой контейнер бинарное дерево. 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 вто-рого 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 возраста-нию 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: правильные пятиугольник, шестиугольник, восьмиугольник.
Контейнер 1-ого уровня: стек.
Контейнер 2-ого уровня: массив.

1 Описание

Принцип открытости/закрытости (OCP) – принцип ООП, устанавливающий следу-ющее 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 кол-лекции, 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 узкоспециализиро-ванные решения. Примерами контейнеров в C++ являются контейнеры из стандарт-ной библиотеки (STL) – map, vector и т.д. В контейнерах часто встречаются реализа-ции 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

TStack_1.h

```
   TStack_1(); - конструктор класса
   void push(const O&); - добавить элемент
   void print(); - распечатать
   void removeLesser(const double&); - удалить меньший
template <typename Q, typename O> class TStack_1
        private:
              class Node
                    public:
                          Q data;
                          std::shared_ptr<Node> next;
                          Node();
                          Node(const O&);
                          int itemsInNode;
                    ;
              std::shared_ptr<Node> head;
              int count;
        public:
```

```
    TStack_1();
    void push(const O&);
    void print();
    void removeLesser(const double&);
;
```

3 Вывод консоли

```
C: hello_world++ OOP7>OOP7
Memory init.
Code 1 == add element. Code 2 == print all array. Code 3 == delete element. Code
0 == break.
Code:
1
Write number of sides:
7
Write length of side. This polygon must be regular.
Incorrect number. You can add only pentagon, hexagon and octagon.
Code:
1
Write number of sides:
5
Write length of side. This polygon must be regular.
7
-Pentagon was created.-
Pentagon copy created
Vector item: created
Item was added
Code:
1
Write number of sides:
5
Write length of side. This polygon must be regular.
6
-Pentagon was created.-
Pentagon copy created
Vector item: created
Item was added
Code:
1
Write number of sides:
5
Write length of side. This polygon must be regular.
5
-Pentagon was created.-
Pentagon copy created
```

Vector item: created
Item was added
Code:
1
Write number of sides:
5
Write length of side. This polygon must be regular.
4
-Pentagon was created.-
Pentagon copy created
Vector item: created
Item was added
Code:
1
Write number of sides:
5
Write length of side. This polygon must be regular.
3
-Pentagon was created.-
Pentagon copy created
Vector item: created
Item was added
Code:
1
Write number of sides:
5
Write length of side. This polygon must be regular.
2
-Pentagon was created.-
Pentagon copy created
New node
++++++ NEW VECTOR ++++++
Vector item: created
Item was added
Code:
1
Write number of sides:
5
Write length of side. This polygon must be regular.
1
-Pentagon was created.-
Pentagon copy created
Vector item: created
Item was added
Code:
2

Element is pentagon. Length of side is equal: 1. Square is 1.72048.
Element is pentagon. Length of side is equal: 2. Square is 2.65839.

Element is pentagon. Length of side is equal: 7. Square is 84.3034.
Element is pentagon. Length of side is equal: 6. Square is 61.9372.
Element is pentagon. Length of side is equal: 5. Square is 43.0119.
Element is pentagon. Length of side is equal: 4. Square is 27.5276.
Element is pentagon. Length of side is equal: 3. Square is 15.4843.

Code:
0
-Octagon was deleted.-
-Hexagon was deleted.-
-Pentagon was deleted.-
Memory freed.

4 Выводы

Выполнив задание этой лабораторной работы - написание контейнера второго уровня, я еще больше улучшил свои знания ООП, ведь данный метод программирования позволяет, как бы мы не усложняли задачу, решить её: элементы контейнера второго уровня при должном уровне абстракции - просто объекты, как и все остальное, и не имеет значения, что эти объекты сами являются контейнерами. Также в данной ЛР были реализованы методы работы с эелементами как контейнера 2го уровня, так и 1го, что улучшило понимание классов и шаблонов.

Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №8 по курсу Объектно-ориентированное программирование

Студент: Баранов А.А.
Преподаватель: Поповкин А.В.
Группа: М8о-207Б
Вариант: 6
Дата:
Оценка:
Подпись:

Москва, 2017

Лабораторная работа №8
Задача: Используя структуры данных, разработанные для лабораторной работы №6 (контейнер 1-ого уровня и классы-фигуры) разработать алгоритм быстрой сортировки для класса-контейнера.

Необходимо разработать два вида алгоритма: 1. Обычный, без параллельных вызовов. 2. С использованием параллельных вызовов. В этом случае, каждый рекурсивный вызов сортировки должен создаваться в отдельном потоке.

Для создания потоков 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: правильные пятиугольник, шестиугольник, восьмиугольник.
Контейнер 1-ого уровня: массив.

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 най-ти 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

TVector.cpp

void TVector<T>::Sort() - рекурсивная сортировка вектора
TVector<T>::sort_in_background() - доп. функция для реализации многопоточности
TVector<T>::Sort_parallel() - многопоточная сортировка вектора

```
template <class T> void TVector<T>::Sort() {
        if (this->size > 0) {
            for (int32_t i = 0; i != this->size - 1; i++) {
                if (this->data[i]->GetFigure()->Square() > this->data[i +
1]->GetFigure()->Square()) swap(this->data[i], this->data[i + 1]);
            }
            this->size–;
            Sort();
        } else {
            for (int32_t i = 0; this->data[i] != nullptr; i++) {this->data[i]-
>SetNext(this->data[i + 1]); this->size++;}// normalize order of elements
        }
}
template<class T > std::future<void> TVector<T>::sort_in_background() {
```

```
        std::packaged_task<void(void) > task(std::bind(std::mem_fn(&TVector<T>::Sort_parallel),
this));
        std::future<void> res(task.get_future());
        std::thread th(std::move(task));
        std::cout « std::this_thread::get_id() « std::endl;
        th.detach();
        return res;
}
template <class T> void TVector<T>::Sort_parallel() {
        if (this->size > 0) {
                for (int32_t i = 0; i != this->size - 1; i++)
                        if (this->data[i]->GetFigure()->Square() > this->data[i + 1]->GetFigure()-
>Square()) swap(this->data[i], this->data[i + 1]);

                this->size–;
                std::future<void> result = this->sort_in_background();
                result.get();
        } else {
                for (int32_t i = 0; this->data[i] != nullptr; i++) this->data[i]->SetNext(this-
>data[i + 1]); this->size++;// normalize order of elements }
}
```

3 Вывод консоли

```
anton@ubuntu: /Documents/OOP8$ ./OOP8.out
   Memory init.
   Code 1 == add element. Code 2 == print element. Code 3 == delete element. Code
4 == print all array. Code 5 == sort elements. Code 0 == break.
   Code:
   1
   Write number of sides:
   5
   Write length of side. This polygon must be regular.
   Length:
   5
   -Pentagon was created.-
   Vector item: created
   Code:
   1
   Write number of sides:
   6
   Write length of side. This polygon must be regular.
   Length:
   4
   -Hexagon was created.-
   Vector item: created
```

Code:
1
Write number of sides:
8
Write length of side. This polygon must be regular.
Length:
3
-Octagon was created.-
Vector item: created
Code:
1
Write number of sides:
5
Write length of side. This polygon must be regular.
Length:
2
-Pentagon was created.-
Vector item: created
Code:
1
Write number of sides:
5
Write length of side. This polygon must be regular.
Length:
1
-Pentagon was created.-
Vector item: created
Code:
4
Element is pentagon. Length of side is equal: 5. Square is 43.0119.
Element is hexagon. Length of side is equal: 4. Square is 41.5692.
Element is octagon. Length of side is equal: 3. Square is 43.4558.
Element is pentagon. Length of side is equal: 2. Square is 6.88191.
Element is pentagon. Length of side is equal: 1. Square is 1.72048.
Code:
5
Code:
4
Element is pentagon. Length of side is equal: 1. Square is 1.72048.
Element is pentagon. Length of side is equal: 2. Square is 6.88191.
Element is hexagon. Length of side is equal: 4. Square is 41.5692.
Element is pentagon. Length of side is equal: 5. Square is 43.0119.
Element is octagon. Length of side is equal: 3. Square is 43.4558.
Code:
0
Vector was deleted.

Memory freed.

4 Выводы

В этой лабораторной работе я реализовал две сортировки массива: простую и многопоточную. Простая сортировка рекурсивно проходит через массив и сортирует его методом пузырька, вторая же сортировка делает все тоже самое, но с каждой итерацией вызывает новый поток и делает это итерацию в нем. Для этого было необходимо написать вспомогательную функцию, в которой используется future и thread для реализации многопоточности. Стоит отметить, что вторая сортировка выполняется в фоне и непосредственно сразу в нескольких потоках, а не в разных, но поочереди. Данная реализация помогает оптимизировать и ускорить сортировку массива пузырьком.

Я смог освоить азы многопоточного программирования и научился использовать такие инструменты как future и packaged_task, что несомненно пригодится в будущем.

Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №9 по курсу Объектно-ориентированное программирование

Студент: Баранов А.А.
Преподаватель: Поповкин А.В.
Группа: М8о-207Б
Вариант: 6
Дата:
Оценка:
Подпись:

Москва, 2017

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 опе-рации над контейнером 1-ого уровня: генерация фигур со случайными значени-ями 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: правильные пятиугольник, шестиугольник, восьмиугольник.
Контейнер 1-ого уровня: массив.
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 устойчи-вым к изменениям.

Непосредственное объявление лямбда-функции состоит из трех частей. Первая часть (квадратные скобки) позволяет привязывать переменные, доступные в текущей об-ласти видимости. Вторая часть (круглые скобки) указывает список принимаемых па-раметров лямбда-функции. Третья часть (фигурные скобки) содержит тело лямбда-функции.

В настоящее время, учитывая, что достигли практически потолка по тактовой часто-те и дальше идет рост количества ядер, появился запрос на параллелизм. В резуль-тате снова в моде стал функциональный подход, так как он очень хорошо работает в условиях параллелизма и не требует явных синхронизаций. Поэтому сейчас уси-ленно думают, как задействовать растущее число ядер процессора и как обеспечить автоматическое распараллеливание. А в функциональном программировании прак-тически 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

Реализация ранее включенных в 9ю лабораторную функций и классов из предыдущих лабораторных неизменна. Были добавлены лямбда-выражения, суть которых - управление вектором фигур, как то: распечатывание, добавление, удаление по критерию.

```
int main(int argc, char** argv) {
	TVector<Figure>* vector_figure = new TVector<Figure>();
	typedef std::function<void (void)> command;
	TVector <command>* vector_cmd = new TVector<command>();
	command cmd_insert = [&]() {
	std::cout « "Command: Create pentagon, hexagon and octagon"« std::endl;
	std::default_random_engine generator;
	std::uniform_int_distribution<int> distribution(1, 100);
	for (int i = 0; i < 3; i++) {
		int side = distribution(generator);
		std::shared_ptr<Figure> tmp;
		 = std::make_shared<Pentagon>(side);
		vector_figure->Add(tmp);
		vector_figure->Resize((vector_figure->Size()) + 1);
		tmp = std::make_shared<Hexagon>(side);
		vector_figure->Add(tmp);
		vector_figure->Resize((vector_figure->Size()) + 1);
		tmp = std::make_shared<Octagon>(side);
		vector_figure->Add(tmp);
		vector_figure->Resize((vector_figure->Size()) + 1);
	}
	};
	command cmd_print = [&]() {
		std::cout « "Command: Print vector"« std::endl;
		for (auto i : *vector_figure) i->Print();
		vector_figure->Print_node(vector_figure->Size() - 1);
	};
	command cmd_remove = [&]() {
		int size = 0;
		std::cout « "Write max square to delete."« std::endl;
		std::cin » size;
		std::cout « "Command: Delete some nodes"« std::endl;
		vector_figure->Delete_by_size(size);
	};
	vector_cmd->Add_cmd(std::shared_ptr<command> (&cmd_insert, [](command*)
{}));
	vector_cmd->Resize((vector_cmd->Size()) + 1);
	vector_cmd->Add_cmd(std::shared_ptr<command> (&cmd_print, [](command*)
{}));
	vector_cmd->Resize((vector_cmd->Size()) + 1);
	vector_cmd->Add_cmd(std::shared_ptr<command> (&cmd_remove, [](command*)
{}));
	vector_cmd->Resize((vector_cmd->Size()) + 1);
	vector_cmd->Add_cmd(std::shared_ptr<command> (&cmd_print, [](command*)
{}));
	vector_cmd->Resize((vector_cmd->Size()) + 1);
```

```
        for (int i = 0; i != vector_cmd->Size(); i++) {
            std::shared_ptr<command> cmd = vector_cmd->Get_node(i);
            std::future<void> ft = std::async(*cmd);
            ft.get();
        }
        return 0;
}
```

3 Вывод консоли

```
anton@ubuntu: /Documents/OOP9$ ./OOP9.out
   Vector item: created
   Vector item: created
   Vector item: created
   Vector item: created
   Command: Create pentagon, hexagon and octagon
   Vector item: created
   Vector item: created
   Vector item: created
   Vector item: created
   Vector item: created
   Vector item: created
   Vector item: created
   Vector item: created
   Vector item: created
   Command: Print vector
   Element is pentagon. Length of side is equal: 1. Square is 1.72048.
   Element is hexagon. Length of side is equal: 1. Square is 2.59808.
   Element is octagon. Length of side is equal: 1. Square is 4.82843.
   Element is pentagon. Length of side is equal: 14. Square is 337.214.
   Element is hexagon. Length of side is equal: 14. Square is 509.223.
   Element is octagon. Length of side is equal: 14. Square is 946.372.
   Element is pentagon. Length of side is equal: 76. Square is 9937.48.
   Element is hexagon. Length of side is equal: 76. Square is 15006.5.
   Element is octagon. Length of side is equal: 76. Square is 27889.
   Element is octagon. Length of side is equal: 76. Square is 27889.
   Write max square to delete.
   1000
   Command: Delete some nodes
   -Pentagon was deleted.-
   -Hexagon was deleted.-
   -Octagon was deleted.-
   -Pentagon was deleted.-
   -Hexagon was deleted.-
   -Octagon was deleted.-
```

Command: Print vector
Element is pentagon. Length of side is equal: 76. Square is 9937.48.
Element is hexagon. Length of side is equal: 76. Square is 15006.5.
Element is octagon. Length of side is equal: 76. Square is 27889.

4 Выводы

Эта лабораторная работа познакомила меня с лямбда-выражениями. Лямбда-выражения - достаточно новая возможность в C++ и изучить её было не только интересно, но и полезно. В данной работе я реализовал вектор в котором элементами явдяются лямбда-выражения, управляющие основным вектором с фигурами. Пришлось активно использовать многопоточное программрование, особенно средства потокобезопасности: такие как mutex, recursive_mutex, lock_guard, unique_lock, без которых не удалось бы наладить и синхронизировать корректное взаимодействие множества потоков над ограниченным числом общих ресурсов.

Приобретенные мной знания не только во время выполнения этой работы, но и курса в целом, сильно пригодились для поднятия моего уровня программирования и знания теории, считаю что заложенный в этом семестре фундамент ООП будет мне служить в дальнейшем надежной опорой.

Ссылка на GitHub с полным кодом работ: https://github.com/NoruNoruBim/OOP