Московский авиационный институт (национальный исследовательский университет)

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

Лабораторная работа №1 по курсу «Объектно-ориентированное программирование»

Студентка: С. Мхитарян
Преподаватель: Поповкин А. В.
Группа: 08-207

Вариант: 15
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)

Фигура: шестиугольник, треугольник, 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

hexagon.cpp	
Hexagon(std::istream &is);	Ввод из потока std::istream
Hexagon(int32t i);	Конструктор класса
double Square() override;	Получение площади
void Print() override;	Печать фигуры
virtual ~Hexagon();	Деструктор класса
triangle.cpp	
Triangle(std::istream &is);	Ввод из потока std::istream
Triangle(sizet i,sizet j,sizet k);	Конструктор класса
double Square() override;	Получение площади
void Print() override;	Печать фигуры
virtual ~Triangle();	Деструктор класса
octagon.cpp	
Octagon(std::istream &is);	Ввод из потока std::istream
Octagon(int32t i);	Конструктор класса
double Square() override;	Получение площади
void Print() override;	Печать фигуры
virtual ~Octagon();	Деструктор класса

```
1
2  class Octagon : public Figure{
3  public:
4      Octagon();
5      Octagon(std::istream &is);
6      Octagon(int32_t i);
7      Octagon(const Octagon& orig);
8      double Square() override;
9      void Print() override;
10     virtual ~Octagon();
11 private:
```

```
12      int32_t side;
13  };
14
15  class Triangle : public Figure{
16  public:
17      Triangle();
18      Triangle(std::istream &is);
19      Triangle(size_t i,size_t j,size_t k);
20      Triangle(const Triangle& orig);
21      double Square() override;
22      void Print() override;
23      virtual ~Triangle();
24  private:
25      size_t side_a;
26      size_t side_b;
27      size_t side_c;
28  };
29
30  class Hexagon : public Figure{
31  public:
32      Hexagon();
33      Hexagon(int32_t i);
34      Hexagon(const Hexagon& orig);
35      Hexagon& operator++();
36      friend bool operator==(const Hexagon& left, const Hexagon& right);
37      friend Hexagon operator+(const Hexagon& left,const Hexagon& right);
38      friend std::ostream& operator<<(std::ostream& os, const Hexagon& obj);
39      friend std::istream& operator>>(std::istream& is, Hexagon& obj);
40      Hexagon(std::istream &is);
41      double Square() override;
42      int32_t Side();
43      void Print() override;
44      Hexagon& operator=(const Hexagon& right);
45      virtual ~Hexagon();
46  private:
47      int32_t side;
48  };
```

3 Консоль

```
sam@sam:~/git/oop-autumn/lab1$ ./main
Use 'help'or 'h'to get help.
h
Commands 'create_triangle'and 'cr_tr'create new triagle with your parameters.
Commands 'create_hexagon'and 'cr_hex'create new hexagon with your parameters.
Commands 'create_octagon'and 'cr_oct'create new octagon with your parameters.
Commands 'print_triangle'and 'pr_tr'output parameters of triagle.
```

```
Commands 'print_hexagon'and 'pr_hex'output parameters of hexagon.
Commands 'print_octagon'and 'pr_oct'output parameters of octagon.
Commands 'square_triangle'and 'sq_tr'output square of triagle.
Commands 'square_hexagon'and 'sq_hex'output square of hexagon.
Commands 'square_octagon'and 'sq_oct'output square of octagon.
Commands 'quit'and 'q'exit the program.
cr_tr
Enter two legs and hypotenuse
1 2 3
pr_tr
Side_A = 1,Side_B = 2,SIDE_C = 3
cr_hex
Enter side.
67
cr_oct
Enter side.
99
pr_hex
Side = 67
pr_oct
Side = 99
cr_oct
Octagon deleted
Enter side.
12
sq_tr
Square: 0
sq_oct
Square: 695.294
h
Commands 'create_triangle'and 'cr_tr'create new triagle with your parameters.
Commands 'create_hexagon'and 'cr_hex'create new hexagon with your parameters.
Commands 'create_octagon'and 'cr_oct'create new octagon with your parameters.
Commands 'print_triangle'and 'pr_tr'output parameters of triagle.
Commands 'print_hexagon'and 'pr_hex'output parameters of hexagon.
Commands 'print_octagon'and 'pr_oct'output parameters of octagon.
Commands 'square_triangle'and 'sq_tr'output square of triagle.
Commands 'square_hexagon'and 'sq_hex'output square of hexagon.
Commands 'square_octagon'and 'sq_oct'output square of octagon.
Commands 'quit'and 'q'exit the program.
q
```

```
Triangle deleted
Hexagon deleted
Octagon deleted
```

4 Выводы

В этой лабораторной были реализованы классы фигур(треугольника, шестиугольника, восьмиугольника). В первой лабораторной работе пришлось познакомится с синтаксисом C++, хоть он похож на C, но всё же отличается наличием ООП. Пришлось познакомится с различными понятиями, такими, как класс, объект, инкапсуляция, абстрактный тип данных, наследование, полиморфизм.

Московский авиационный институт (национальный исследовательский университет)

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу «Объектно-ориентированное программирование»

Студентка: С. Мхитарян
Преподаватель: А. В. Поповкин
Группа: 08-207

Вариант: 15
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 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: шестиугольник.
Контейнер: бинарное дерево.

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

hexagon.cpp	
int Side();	Сторона фигуры
double Square();	Площадь фигуры
void Print();	Печать фигуры

~Hexagon();	Деструктор
Hexagon& operator=(const Hexagon& right);	Переопределенный оператор равно
std::ostream& operator«(std::ostream& os, const Hexagon& obj);	Переопределенный оператор вывода в поток std::ostream
std::istream& operator»(std::istream& is, Hexagon& obj);	Переопределенный оператор ввода из потока std::istream
bool operator==(const Hexagon& left, const Hexagon& right);	Переопределенный оператор сравнения
TBinaryTreeItem.cpp	
TBinaryTreeItem();	Конструктор класса
int Side();	Сторона фигуры
Hexagon GetHexagon();	Возвращает фигуру из вершины
~TBinaryTreeItem();	Деструктор
TBinaryTree.cpp	
TBinaryTree();	Конструктор класса
TBinaryTreeItem* find(int a);	Поиск вершины
void remove(int a);	Удаление вершины
void insert(Hexagon & & hexagon);	Вставка вершины
void print();	Печать дерева
bool empty();	Проверка на пустое дерево
virtual ~TBinaryTree();	Деструктор

```
1
2  class TBinaryTreeItem
3  {
4  public:
5      TBinaryTreeItem();
6      TBinaryTreeItem(Hexagon& hexagon);
7
8      int32_t Side();
9      Hexagon GetHexagon();
10     ~TBinaryTreeItem();
11     friend TBinaryTree;
12 private:
13     Hexagon hexagon;
14     TBinaryTreeItem* left;
15     TBinaryTreeItem* right;
16 };
17
18 class TBinaryTreeItem
19 {
20 public:
```

```
21      TBinaryTreeItem();
22      TBinaryTreeItem(Hexagon& hexagon);
23
24      int32_t Side();
25      Hexagon GetHexagon();
26      ~TBinaryTreeItem();
27      friend TBinaryTree;
28  private:
29      Hexagon hexagon;
30      TBinaryTreeItem* left;
31      TBinaryTreeItem* right;
32  };
33
34  class Hexagon : public Figure{
35  public:
36      Hexagon();
37      Hexagon(int32_t i);
38      Hexagon(const Hexagon& orig);
39      Hexagon& operator++();
40      friend bool operator==(const Hexagon& left, const Hexagon& right);
41      friend Hexagon operator+(const Hexagon& left,const Hexagon& right);
42      friend std::ostream& operator<<(std::ostream& os, const Hexagon& obj);
43      friend std::istream& operator>>(std::istream& is, Hexagon& obj);
44      Hexagon(std::istream &is);
45      double Square() override;
46      int32_t Side();
47      void Print() override;
48      Hexagon& operator=(const Hexagon& right);
49      virtual ~Hexagon();
50  private:
51      int32_t side;
52  };
```

3 Консоль

```
sam@sam:~/git/oop-autumn/lab2$ valgrind --leak-check=full ./main
==4940== Memcheck,a memory error detector
==4940== Copyright (C) 2002-2015,and GNU GPL'd,by Julian Seward et al.
==4940== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright
info
==4940== Command: ./main
==4940==
Use 'help'or 'h'to get help.
h
Commands 'insert'and 'ins'create new hexagon in bintree.
Commands 'remove'and 'rm'remove hexagon in bintree.
```

```
Commands 'find'and 'f'find hexagon with your side.
Commands 'destroy'and 'd'destroy bintree.
Commands 'print'and 'pr'print bintree.
Commands 'quit'and 'q'exit the program.
ins
6
Hexagon created: 6
Hexagon created: 0
Hexagon deleted
p

pr
6

find 6
6
find
6
Side = 6
Hexagon deleted
pr
6

d
Hexagon deleted
Tree destroy.
ins
12
Hexagon created: 12
Hexagon created: 0
Hexagon deleted
0
quit
Hexagon deleted
==4940==
==4940== HEAP SUMMARY:
==4940==     in use at exit: 0 bytes in 0 blocks
==4940==   total heap usage: 7 allocs,7 frees,74,848 bytes allocated
==4940==
==4940== All heap blocks were freed --no leaks are possible
==4940==
```

```
==4940== For counts of detected and suppressed errors,rerun with: -v
==4940== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

4 Выводы

В этой лабораторной было предложено написать свою структуру данных. В моем случае это бинарное дерево. Я сделал его бинарным деревом поиска. Конечно, все эти контейнеры есть в стандартной библиотеке шаблонов, но любому профессиональному программисту не составит труда реализовать свой список, стек, очередь или любую другую широко используемую структуру данных. Фигуры передаются в бинарное дерево "по значению чтобы в них хранился сам объект, а не его копия.

Московский авиационный институт (национальный исследовательский университет)

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу «Объектно-ориентированное программирование»

Студентка: С. Мхитарян
Преподаватель: Поповкин А. В.
Группа: 08-207

Вариант: 15
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 контейнера.
- Классы должны быть расположены в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

Фигуры: треугольник, шестиугольник, восьмиугольник
Контейнер: бинарное дерево.

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

TBinaryTreeItem.cpp	
TBinaryTreeItem(const std::sharedptr<Figure> &obj);	Конструктор класса
std::sharedptr<Figure> GetFigure();	Получение фигуры
~TBinaryTreeItem();	Деструктор класса
TBinaryTree.cpp	
TBinaryTree();	Конструктор класса
std::sharedptr<TBinaryTreeItem> find(std::sharedptr<Figure> &obj);	Поиск вершины
void remove(int32 side);	Удаление вершины
void insert(std::sharedptr<Figure> &obj);	Вставка вершины
void print();	Печать дерева
bool empty();	Проверка пустое ли дерево
virtual ~TBinaryTree();	Деструктор класса

```
1
2  class TBinaryTree
3  {
4  public:
5      friend std::ostream& operator<<(std::ostream& os, TBinaryTree& tree);
6      TBinaryTree();
7      std::shared_ptr<TBinaryTreeItem> find(std::shared_ptr<Figure> &obj);
8      void remove(int32_t side);
9      void insert(std::shared_ptr<Figure> &obj);
10     void print();
11     void print(std::ostream& os);
12     bool empty();
13     virtual ~TBinaryTree();
14 private:
15     std::shared_ptr<TBinaryTreeItem> head;
16     std::shared_ptr<TBinaryTreeItem> minValueNode(std::shared_ptr<TBinaryTreeItem> root
           );
17     std::shared_ptr<TBinaryTreeItem> deleteNode(std::shared_ptr<TBinaryTreeItem> root,
           int32_t side);
18     void print_tree(std::shared_ptr<TBinaryTreeItem> item, int32_t a, std::ostream& os)
           ;
19 };
20
21 class TBinaryTreeItem
22 {
23 public:
24     TBinaryTreeItem();
25     TBinaryTreeItem(const std::shared_ptr<Figure> &obj);
26
27     std::shared_ptr<Figure> GetFigure();
28     ~TBinaryTreeItem();
29     friend TBinaryTree;
30 private:
31     std::shared_ptr<Figure> item;
32     std::shared_ptr<TBinaryTreeItem> left;
33     std::shared_ptr<TBinaryTreeItem> right;
34
35 };
```

3 Консоль

```
sam@sam:~/git/oop-autumn/lab3$ valgrind --leak-check=full ./main
==5245== Memcheck,a memory error detector
==5245== Copyright (C) 2002-2015,and GNU GPL'd,by Julian Seward et al.
==5245== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright
info
==5245== Command: ./main
```

```
==5245==
Чтобы вызвать меню,нажмите 6
6
Список операций:
1) Добавление треугольника
2) Добавление восьмиугольник
3) Добавление шестиугольника
4) Удаление фигуры из дерева по максимальной стороне
5) Печать дерева
0) Выход
1
1 2 3
5
Side_A = 1,Side_B = 2,Side_C = 3

2
2
3
60
5
Side_A = 1,Side_B = 2,Side_C = 3
Side = 2
Side = 60

4
2
5
Side_A = 1,Side_B = 2,Side_C = 3
null
Side = 60

0
==5245==
==5245== HEAP SUMMARY:
==5245==     in use at exit: 0 bytes in 0 blocks
==5245==   total heap usage: 9 allocs,9 frees,75,056 bytes allocated
==5245==
==5245== All heap blocks were freed --no leaks are possible
==5245==
==5245== For counts of detected and suppressed errors,rerun with: -v
==5245== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

4 Выводы

В этой лабораторной работе необходимо было реализовать умные указатели. В данном случае sharedptr, который ведет подсчет ссылок на объект и если счет равен 0, то указатель на объект удаляется. Так же в данной лабораторной работе был изучен makeshared, который возвращает sharedptr на объект с числом 1.

Московский авиационный институт (национальный исследовательский университет)

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

Лабораторная работа №4 по курсу «Объектно-ориентированное программирование»

Студентка: С. Мхитарян
Преподаватель: Поповкин А. В.
Группа: 08-207

Вариант: 15
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 работы 1.
- Шаблон класса-контейнера должен соджержать объекты используя std:sharedptr<...>.
- Шаблон класса-контейнера должен иметь метод по добавлению фигуры в контейнер.
- Шаблон класса-контейнера должен иметь методы по получению фигуры из контейнера (опередеяется структурой контейнера).
- Шаблон класса-контейнера должен иметь метод по удалению фигуры из контейнера (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 элементы контейнера.
- Классы должны быть расположенны в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

Фигуры: треугольник, шестиугольник, восьмиугольник.
Контейнер: бинарное дерево.

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 работе с коллекциями и умными указателями.

2 Исходный код

TBinaryTreeItem.cpp	
TBinaryTreeItem(const std::sharedptr<T> &obj);	Конструктор класса
std::sharedptr<T> GetFigure();	Получение фигуры
~TBinaryTreeItem();	Деструктор класса
TBinaryTree.cpp	
std::sharedptr<TBinaryTreeItem<T» find(std::sharedptr<T> &obj);	Поиск вершины
void insert(std::sharedptr<T> &obj);	Вставка вершины

```
1
2  template <class T> class TBinaryTree
3  {
4  public:
5      TBinaryTree();
6      std::shared_ptr<TBinaryTreeItem<T>> find(std::shared_ptr<T> &obj);
7      void remove(int32_t side);
8      void insert(std::shared_ptr<T> &obj);
9      void print();
10     void print(std::ostream& os);
11     template <class A> friend std::ostream& operator<<(std::ostream& os, TBinaryTree<A>
           &tree);
```

```
12      bool empty();
13      virtual ~TBinaryTree();
14  private:
15      std::shared_ptr<TBinaryTreeItem<T>> head;
16      std::shared_ptr<TBinaryTreeItem<T>> minValueNode(std::shared_ptr<TBinaryTreeItem<T
            >> root);
17      std::shared_ptr<TBinaryTreeItem<T>> deleteNode(std::shared_ptr<TBinaryTreeItem<T>>
            root, int32_t side);
18      void print_tree(std::shared_ptr<TBinaryTreeItem<T>> item, int32_t a, std::ostream&
            os);
19  };
20
21  template <class T> class TBinaryTreeItem
22  {
23  public:
24      TBinaryTreeItem();
25      TBinaryTreeItem(const std::shared_ptr<T> &obj);
26
27      std::shared_ptr<T> GetFigure();
28      ~TBinaryTreeItem();
29      friend class TBinaryTree<T>;
30      //template <class A> friend std::ostream& operator<<(std::ostream &os,
            TBinaryTreeItem<A> &obj);
31  private:
32      std::shared_ptr<T> item;
33      std::shared_ptr<TBinaryTreeItem<T>> left;
34      std::shared_ptr<TBinaryTreeItem<T>> right;
35
36  };
```

3 Консоль

```
sam@sam:~/git/oop-autumn/lab4$ valgrind --leak-check=full ./main
==5417== Memcheck,a memory error detector
==5417== Copyright (C) 2002-2015,and GNU GPL'd,by Julian Seward et al.
==5417== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright
info
==5417== Command: ./main
==5417==
Чтобы вызвать меню,нажмите 6
6
Список операций:
1) Добавление треугольника
2) Добавление восьмиугольник
3) Добавление шестиугольника
4) Удаление фигуры из дерева по максимальной стороне
```

```
5) Печать дерева
0) Выход
1
1 2 3
2
2
3
45
5
Side_A = 1,Side_B = 2,Side_C = 3
Side = 2
Side = 45

4
3
5
Side = 45
Side = 2
null

0
==5417==
==5417== HEAP SUMMARY:
==5417==     in use at exit: 0 bytes in 0 blocks
==5417==   total heap usage: 9 allocs,9 frees,75,056 bytes allocated
==5417==
==5417== All heap blocks were freed --no leaks are possible
==5417==
==5417== For counts of detected and suppressed errors,rerun with: -v
==5417== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

4 Выводы

В этой лабораторной работе был спроектирован и запрограммирован на языке C++ шаблон класса-контейнера первого уровня(бинарное дерево), что позволило держать в вершинах бинарного дерева три фигуры на выбор(треугольник, шестиугольник, восьмиугольник). Были подробно изучены шаблоны и их правильное использование.

Московский авиационный институт (национальный исследовательский университет)

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

Лабораторная работа №5 по курсу «Объектно-ориентированное программирование»

Студентка: С. Мхитарян
Преподаватель: Поповкин А. В.
Группа: 08-207

Вариант: 15
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: треугольник, шестиугольник, восьмиугольник
Контейнер: бинарное дерево.

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      bool operator== (const TIterator &i) {
29          return (cur == i.cur );
30      }
31
32      bool operator!= (const TIterator &i) {
33          return !(cur == i.cur );
34      }
35
36  private:
37      std::shared_ptr<N> cur;
38  };
```

3 Консоль

```
sam@sam:~/git/oop-autumn/lab5$ valgrind --leak-check=full ./main
==5848== Memcheck,a memory error detector
==5848== Copyright (C) 2002-2015,and GNU GPL'd,by Julian Seward et al.
==5848== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright
info
==5848== Command: ./main
==5848==
Чтобы вызвать меню,нажмите 7
7
Список операций:
1) Добавление треугольника
2) Добавление восьмиугольник
3) Добавление шестиугольника
4) Удаление фигуры из дерева по максимальной стороне
5) Печать дерева
6) Печать дерева с итератором
0) Выход
1
```

```
1 2 3
2
2
3
10
1
2 20 1
2
68
3
30
3
89
5
Side_A = 1,Side_B = 2,Side_C = 3
Side = 2
Side = 10
null
Side_A = 2,Side_B = 20,Side_C = 1
null
Side = 68
Side = 30
Side = 89

6
Side = 2
Side_A = 1,Side_B = 2,Side_C = 3
Side = 10
Side_A = 2,Side_B = 20,Side_C = 1
Side = 30
Side = 68
Side = 89

0
==5848==
==5848== HEAP SUMMARY:
==5848==     in use at exit: 0 bytes in 0 blocks
==5848==   total heap usage: 23 allocs,23 frees,76,048 bytes allocated
==5848==
==5848== All heap blocks were freed --no leaks are possible
==5848==
```

```
==5848== For counts of detected and suppressed errors,rerun with: -v
==5848== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

4 Выводы

В этой лабораторной работе был спроектирован и разработан Итератор для бинарного дерева. Начало итератора это самая малая вершина, то есть до конца влево от корня дерева, а последняя вершина в итерации это самая правая вершина вниз от корня. Так же были изучены разные типы итераторов. Итератор разработан в виде шаблона и работает с такими фигурами, как треугольник, шестиугольник, восьмиугольник.

Московский авиационный институт (национальный исследовательский университет)

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

Лабораторная работа №6 по курсу «Объектно-ориентированное программирование»

Студентка: С. Мхитарян
Преподаватель: Поповкин А. В.
Группа: 08-207

Вариант: 15
Дата:
Оценка:
Подпись:

Москва, 2017

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: треугольник, шестиугольник, восьмиугольник.
Контейнер 1-ого уровня: бинарное дерево.
Контейнер 2-ого уровня: бинарное дерево.

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(sizet size,sizet count);	
void *allocate();	Выделение памяти
void deallocate(void *pointer);	Освобождение памяти
bool hasfreeblocks();	Есть ли пустые блоки
virtual ~TAllocationBlock();	Деструктор

```
1
2  class TAllocationBlock {
3  public:
4      TAllocationBlock(size_t size,size_t count);
5      void *allocate();
6      void deallocate(void *pointer);
7      bool has_free_blocks();
```

```
8       virtual ~TAllocationBlock();
9   private:
10      size_t _size;
11      size_t _count;
12      char *_used_blocks;
13      void **_free_blocks;
14      size_t _free_count;
15  };
```

3 Консоль

```
sam@sam:~/git/oop-autumn/lab6$ valgrind --leak-check=full ./main
==2753== Memcheck,a memory error detector
==2753== Copyright (C) 2002-2015,and GNU GPL'd,by Julian Seward et al.
==2753== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright
info
==2753== Command: ./main
==2753==
TAllocationBlock: Memory init
Чтобы вызвать меню,нажмите 7
7
Список операций:
1) Добавление треугольника
2) Добавление восьмиугольник
3) Добавление шестиугольника
4) Удаление фигуры из дерева по максимальной стороне
5) Печать дерева
6) Печать дерева с итератором
0) Выход
1
1 2 3
2
2
3
5
5
Side_A = 1,Side_B = 2,Side_C = 3
Side = 2
Side = 5

0
```

```
TAllocationBlock: Memory freed
==2753==
==2753== HEAP SUMMARY:
==2753==     in use at exit: 0 bytes in 0 blocks
==2753==   total heap usage: 11 allocs,11 frees,82,304 bytes allocated
==2753==
==2753== All heap blocks were freed --no leaks are possible
==2753==
==2753== For counts of detected and suppressed errors,rerun with: -v
==2753== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

4 Выводы

В этой лабораторной работе был спроектирован и разработан аллокатор памяти для бинарного дерева. Был минимизирован вызов операции malloc. Для вызова аллокатора были переопределены операторы new и delete у классов-фигур. Для хранения свободных блоков было реализовано бинарное дерево. Аллокатор выделяет большие блоки памяти на 100 фигур.

Московский авиационный институт (национальный исследовательский университет)

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

Лабораторная работа №7 по курсу «Объектно-ориентированное программирование»

Студентка: С. Мхитарян
Преподаватель: Поповкин А. В.
Группа: 08-207

Вариант: 15
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 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: треугольник, шестиугольник, восьмиугольник.
Контейнер 1-ого уровня: бинарное дерево.
Контейнер 2-ого уровня: бинарное дерево.

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.

TTree.cpp	
void recRemByType(std::sharedptr<Node>&, int&);	Удаление всех фигур одного типа
void recInsert(std::sharedptr<Node>&, O&);	Вставка фигуры
void recInorder(std::sharedptr<Node>&);	Печать
void recRemLesser(std::sharedptr<Node>&, int32t&);	Удаление по стороне одной фигуры

```
1
2 template <class Q, class O> class TTree {
3 private:
4     class Node {
```

```
5       public:
6           Q data;
7           std::shared_ptr<Node> son;
8           std::shared_ptr<Node> sibling;
9           Node();
10          Node(O&);
11          int itemsInNode;
12      };
13
14      std::shared_ptr<Node> root;
15
16      void recRemByType(std::shared_ptr<Node>&, int&);
17      void recInsert(std::shared_ptr<Node>&, O&);
18      void recInorder(std::shared_ptr<Node>&);
19      void recRemLesser(std::shared_ptr<Node>&, int32_t&);
20  public:
21      TTree();
22
23      void insert(O&);
24      void inorder();
25      void removeByType(int&);
26      void removeLesser(int32_t&);
27      void RM();
28  };
```

3 Консоль

```
sam@sam:~/git/oop-autumn/lab7$ valgrind --leak-check=full ./main
==2570== Memcheck,a memory error detector
==2570== Copyright (C) 2002-2015,and GNU GPL'd,by Julian Seward et al.
==2570== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright
info
==2570== Command: ./main
==2570==
Чтобы вызвать меню,нажмите 7
7
Список операций:
1) Добавление треугольника
2) Добавление восьмиугольник
3) Добавление шестиугольника
4) Удаление всех фигур одного типа из дерева. 1)Треугольник 2)Восьмиугольник
3)Шестиугольник
5) Удаление фигуры из дерева по максимальной стороне
6) Печать дерева
```

```
0) Выход
2
2
2
4
2
5
6
Side = 2
Side = 4
Side = 5

3
8
3
10
2
20
6
Side = 2
Side = 4
Side = 5
Side = 8
Side = 10

Side = 20

4
2
6
Side = 4
Side = 5
Side = 8
Side = 10

5
8
6
Side = 4
Side = 5
Side = 10
```

```
0
==2570==
==2570== HEAP SUMMARY:
==2570==     in use at exit: 0 bytes in 0 blocks
==2570==   total heap usage: 36 allocs,36 frees,77,104 bytes allocated
==2570==
==2570== All heap blocks were freed --no leaks are possible
==2570==
==2570== For counts of detected and suppressed errors,rerun with: -v
==2570== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

4 Выводы

В этой лабораторной работе был реализован контейнер второго уровня. В вершинах которого хранятся бинарные деревья с с кол-вом вершин не больше 5. Было реализовано два вида удаления из контейнера второго уровня. Первый вид удаляет все фигуры одного типа(все треугольники или все шестиугольники, или все восьмиугольники). В добавок к этому была реализована обычная вставка в бинарное дерево и его печать.

Московский авиационный институт (национальный исследовательский университет)

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

Лабораторная работа №8 по курсу «Объектно-ориентированное программирование»

Студентка: С. Мхитарян
Преподаватель: Поповкин А. В.
Группа: 08-207

Вариант: 15
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 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: треугольник, шестиугольник, восьмиугольник.
Контейнер: бинарное дерево.

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  void TBinaryTree<T>::MySort(std::shared_ptr<T> *&arr, int l, int r)
4  {
5      int x = l + (r - l) / 2;
6      int i = l;
7      int j = r;
8
9      while(i <= j) {
10         while(arr[i]->Side() < arr[x]->Side()) {
11             i++;
12         }
13         while(arr[j]->Side() > arr[x]->Side()) {
14             j--;
15         }
16         if(i <= j) {
17             std::shared_ptr<T> tmp = arr[i];
18             arr[i] = arr[j];
```

```
19              arr[j] = tmp;
20              i++;
21              j--;
22          }
23      }
24      if (i < r) {
25          MySort(arr, i, r);
26      }
27
28      if (l < j) {
29          MySort(arr, l, j);
30      }
31  }
32
33  int Sort(std::shared_ptr<Figure> *&arr, int l, int r)
34  {
35      int x = l + (r - l) / 2;
36      int i = l;
37      int j = r;
38
39      while(i <= j) {
40          while(arr[i]->Side() < arr[x]->Side()) {
41              i++;
42          }
43          while(arr[j]->Side() > arr[x]->Side()) {
44              j--;
45          }
46          if(i <= j) {
47              std::shared_ptr<Figure> tmp = arr[i];
48              arr[i] = arr[j];
49              arr[j] = tmp;
50              i++;
51              j--;
52          }
53      }
54      if (i < r) {
55          std::packaged_task<int(std::shared_ptr<Figure> *&, int,int)> task(Sort);
56          auto result = task.get_future();
57
58          std::thread task_td(std::move(task), std::ref(arr), i, r);
59          task_td.join();
60          result.get();
61      }
62      if (l < j) {
63          std::packaged_task<int(std::shared_ptr<Figure> *&, int,int)> task(Sort);
64          auto result = task.get_future();
65
66          std::thread task_td(std::move(task), std::ref(arr), l, j);
67          task_td.join();
```

```
68          result.get();
69      }
70      return 0;
71  }
```

3 Консоль

```
sam@sam:~/git/oop-autumn/lab8$ valgrind --leak-check=full ./main
==2724== Memcheck,a memory error detector
==2724== Copyright (C) 2002-2015,and GNU GPL'd,by Julian Seward et al.
==2724== Using Valgrind-3.12.0.SVN and LibVEX; rerun with -h for copyright
info
==2724== Command: ./main
==2724==
Чтобы вызвать меню,нажмите 9
8
Введите количество вершин
3
Осталось ввести 3 вершин(у)
5
Осталось ввести 2 вершин(у)
2
Осталось ввести 1 вершин(у)
1
6
Side = 1
Side = 2
Side = 5

0
==2724==
==2724== HEAP SUMMARY:
==2724==     in use at exit: 0 bytes in 0 blocks
==2724==   total heap usage: 14 allocs,14 frees,75,736 bytes allocated
==2724==
==2724== All heap blocks were freed --no leaks are possible
==2724==
==2724== For counts of detected and suppressed errors,rerun with: -v
==2724== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

4 Выводы

В этой лабораторной работе была реализована обычная быстрая сортировки и сбыстрая сортировка с использованием потоков. Как эта сортировка проходит для бинарного дерева? Легко. Вводится размер массива, потом вводится n-элементов для массива, где n - размер массива. Затем массив сортируется с/без помощи потоков. Далее просто из отсортированного массива строится сбалансированное бинарное дерево.

Лабораторная помогла лучше разобраться в том, что такое потоки, как их представить, как проверить с помощью стороннего ПО.

Московский авиационный институт (национальный исследовательский университет)

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

Лабораторная работа №9 по курсу «Объектно-ориентированное программирование»

Студентка: С. Мхитарян
Преподаватель: Поповкин А. В.
Группа: 08-207

Вариант: 15
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 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: треугольник, шестиугольник, восьмиугольник.
Контейнер 1-ого уровня: бинарное дерево.
Контейнер 2-ого уровня: бинарное дерево.

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.

```

int main(void) {
  TBinaryTree<Figure> ttree;
  typedef std::function<void(void)> command;
  TTree<command> tree(4);
  command cmdInsert = [&]() {
    std::cout << "Command: Insert" << std::endl;
    std::default_random_engine generator;
    std::uniform_int_distribution<int> distribution(1, 10);
    for (int i = 0; i < 10; i++) {
      int side = distribution(generator);
      if ((side % 2) == 0) {
```

```
13          std::shared_ptr<Figure> ptr = std::make_shared<Triangle>(Triangle(side, side,
                side));
14          if (ttree.find(ptr) == nullptr) {
15              ttree.insert(ptr);
16          }
17        } else if((side % 3) == 0) {
18          std::shared_ptr<Figure> ptr = std::make_shared<Octagon>(Octagon(side));
19          if (ttree.find(ptr) == nullptr) {
20              ttree.insert(ptr);
21          }
22        } else {
23          std::shared_ptr<Figure> ptr = std::make_shared<Hexagon>(Hexagon(side));
24          if (ttree.find(ptr) == nullptr) {
25              ttree.insert(ptr);
26          }
27        }
28      }
29    };
30    command cmdPrint = [&]() {
31      std::cout << "Command: Print" << std::endl;
32      for (auto i : ttree) {
33        i->Print();
34      }
35    };
36    command cmdRemove = [&]() {
37      std::cout << "Command: Remove" << std::endl;
38      std::default_random_engine generator;
39      std::uniform_int_distribution<int> distribution(1, 10);
40      int side = distribution(generator);
41      std::cout << "Lesser than " << side << std::endl;
42      for (int i = 0; i < 10; i++) {
43
44        for (auto iter: ttree) {
45          if (iter->Side() < side) {
46            ttree.remove(iter->Side());
47            break;
48          }
49        }
50      }
51    };
52
53    tree.insert(std::shared_ptr<command>(&cmdInsert, [](command*){}));
54    tree.insert(std::shared_ptr<command>(&cmdPrint, [](command*){}));
55    tree.insert(std::shared_ptr<command>(&cmdRemove, [](command*){}));
56    tree.insert(std::shared_ptr<command>(&cmdPrint, [](command*){}));
57    tree.inorder();
58
59    return 0;
60  }
```

3 Консоль

```
sam@sam:~/git/oop-autumn/lab9$ ./main
Command: Insert
Hexagon created: 1
Triangle created: 2,2,2
Triangle copy created
Triangle created: 8,8,8
Triangle copy created
Hexagon created: 5
Triangle created: 6,6,6
Triangle copy created
Octagon created: 3
Octagon copy created
Hexagon created: 1
Hexagon created: 7
Hexagon created: 7
Triangle created: 10,10,10
Triangle copy created
Command: Print
Side = 1
Side_A = 2,Side_B = 2,Side_C = 2
Side = 3
Side = 5
Side_A = 6,Side_B = 6,Side_C = 6
Side = 7
Side_A = 8,Side_B = 8,Side_C = 8
Side_A = 10,Side_B = 10,Side_C = 10
Command: Remove
Lesser than 1
Command: Print
Side = 1
Side_A = 2,Side_B = 2,Side_C = 2
Side = 3
Side = 5
Side_A = 6,Side_B = 6,Side_C = 6
Side = 7
Side_A = 8,Side_B = 8,Side_C = 8
Side_A = 10,Side_B = 10,Side_C = 10
```

4 Выводы

В этой лабораторной работе были реализованы лямбда-выражения, которые были помещены в вершины бинарного дерева и выполнялись поочередно. Так было реализовано удаление сторон меньшего, чем заданное число, которое генерировалось в каком-то диапазоне. Было реализовано добавление фигур с различными сторонами. Печать дерева.

Вот и подходит к концу курс Объектно-Ориентированного Программирования. Этот курс оказался очень полезным. Помог с другими предметами 3 семестра, к примеру, с Операционными Системами, а точнее с пониманием параллельного программирования.

Мои лабораторные работы можно найти по ссылке https://github.com/samvel63/oop-autumn

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: С. Мхитарян
Преподаватель: А. В. Поповкин
Группа: 08-207

Вариант: 15
Дата:
Оценка:
Подпись:

Москва, 2017