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: С. М. Бокоч
Преподаватель:
Группа: М8О-204Б

Дата:
Оценка:
Подпись:

Москва, 2017

Лабораторная работа №2

Задача: Цель работы Целью лабораторной работы является:

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

Задание Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня, содержащий одну фигуру (колонка фигура 1), согласно варианту задания (реализованную в ЛР1). 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 (тип фигуры, длины сторон, радиус и т.д).
- Классы фигур должны иметь переопределенный оператор ввода фигуры из потока std::istream (»). Оператор должен вводить основные параметры фигуры (длины сторон, радиус и т.д).
- Классы фигур должны иметь операторы копирования (=).
- Классы фигур должны иметь операторы сравнения с такими же фигурами (==).
- Класс-контейнер должен соджержать 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 (опередеряется структурой контейнера).
- Класс-контейнер должен иметь метод по удалению фигуры из контейнера (опередеряется структурой контейнера).
- Класс-контейнер должен иметь перегруженный оператор по выводу контейнера в поток std::ostream («).

- Класс-контейнер должен иметь деструктор, удаляющий все элементы контейнера.
- Классы должны быть расположенны в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

Нельзя использовать:

- Стандартные контейнеры std.
- Шаблоны (template).
- Различные варианты умных указателей (shared_ptr, weak_ptr).

Программа должна позволять:

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

1 Теория

Динамические структуры данных – это структуры данных, память под которые выделяется и освобождается по мере необходимости. Динамические структуры данных в процессе существования в памяти могут изменять не только число составляющих их элементов, но и характер связей между элементами. При этом не учитывается изменение содержимого самих элементов данных. Такая особенность динамических структур, как непостоянство их размера и характера отношений между элементами, приводит к тому, что на этапе создания машинного кода программа-компилятор не может выделить для всей структуры в целом участок памяти фиксированного размера, а также не может сопоставить с отдельными компонентами структуры конкретные адреса. Для решения проблемы адресации динамических структур данных используется метод, называемый динамическим распределением памяти, то есть память под отдельные элементы выделяется в момент, когда они "начинают существовать"в процессе выполнения программы, а не во время компиляции. Компилятор в этом случае выделяет фиксированный объем памяти для хранения адреса динамически размещаемого элемента, а не самого элемента. Динамическая структура данных характеризуется тем что:

1. она не имеет имени; (чаще всего к ней обращаются через указатель на адрес)

2. ей выделяется память в процессе выполнения программы;
3. количество элементов структуры может не фиксироваться;
4. размерность структуры может меняться в процессе выполнения программы;
5. в процессе выполнения программы может меняться характер взаимосвязи между элементами структуры.

При передаче по значению содержимое аргумента копируется в формальный параметр подпрограммы. Изменения, сделанные в параметре, не влияют на значение переменной, используемой при вызове.

2 Описание программы

```
1  //Square.h
2  #ifndef SQUARE_H
3  #define SQUARE_H
4
5  #include <iostream>
6  #include "Figure.h"
7
8  class Square : public Figure
9  {
10 public:
11   Square();
12   Square(size_t a);
13   Square(std::istream& is);
14   Square(const Square& object1);
15
16   void Print() const override;
17   double Area() const override;
18
19
20   Square& operator=(const Square& obj);
21   Square& operator++();
22   Square& operator++(int);
23     bool operator==(const Square& lhs) const;
24   friend std::istream& operator>>(std::istream& is, Square& square);
25   friend std::ostream& operator<<(std::ostream& os, const Square& square);
26     ~Square();
27 private:
28   size_t side;
29 };
30
31 #endif
32 #include "Square.h"
33
34 Square::Square() : Square(0) {
35 }
36 Square::Square(size_t sideA) : side(sideA) {
37   //std::cout << "Square created with side " << side << std::endl;
38 }
39 Square::Square(const Square& object1) {
40   //std::cout << "Square copy created\n";
41   side = object1.side;
42 }
43 Square::Square(std::istream& is)
44 {
45   std::cout << "Enter side: ";
46   is >> side;
47 }
```

```
48
49  void Square::Print() const
50  {
51    std::cout << "Figure type: Square" << std::endl;
52    std::cout << "Side size: " << side << std::endl;
53  }
54
55  double Square::Area() const
56  {
57    return side * side;
58  }
59  Square::~Square() {
60    //std::cout << "Square deleted" << std::endl;
61  }
62
63  std::ostream& operator<<(std::ostream& os, const Square& square){
64    os << "Side A = " << square.side;
65    return os;
66  }
67
68  std::istream& operator>>(std::istream& is, Square& square) {
69    is >> square.side;
70    return is;
71  }
72  Square& Square::operator=(const Square& obj) {
73    //std::cout << "Using operator is = ";
74    if (this == &obj)
75      return *this;
76    side = obj.side;
77    return *this;
78  }
79
80  Square& Square::operator++(){
81    ++side;
82    return *this;
83  }
84  Square& Square::operator++(int){
85    Square *old_obj = new Square(*this);
86    side++;
87    return *old_obj;
88  }
89
90  bool Square::operator==(const Square &lhs) const{
91      return this->side == lhs.side;
92  }
93  //TArray.h
94  #ifndef PROG_TArray_H
95  #define PROG_TArray_H
96  #include "Square.h"
```

```
97  #include "iostream"
98  class TArray {
99  public:
100     TArray();
101     explicit TArray(const size_t &);
102     TArray(TArray&);
103     explicit TArray(TArray*);
104     void Push_back(const Square &);
105     bool Delete(const Square &);
106     bool Replace(const Square &, const Square &);
107     size_t Size() const;
108     size_t Capacity() const;
109     Square& operator[](size_t);
110     Square& operator[](size_t) const;
111     friend std::istream &operator>>(std::istream &, TArray&);
112     friend std::ostream &operator<<(std::ostream &, const TArray &);
113     ~TArray();
114 private:
115     Square *_data;
116     size_t _capacity;
117     size_t _size;
118 };
119 #endif // PROG_TArray_H
```

3 Выводы

В данной лабораторной работе я получил навыки программирования классов на языке C++, познакомился с перегрузкой операторов и дружественными функциями. Это было очень познавательно.