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: М. В. Спиридонов
Преподаватель:
Группа: М8О-206Б

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  //main.cpp
2  #include "triangle.h"
3  #include "tMass.h"
4
5  int main (int argc, char **argv) {
6      TMass *mass = new TMass();
7      mass->AddToFirstFree(Triangle(1, 1, 1));
8      mass->AddToFirstFree(Triangle(2, 2, 2));
9      mass->AddToFirstFree(Triangle(3, 3, 3));
10     mass->AddToFirstFree(Triangle(4, 4, 4));
11
12     std::cout << " Internal mass size = " << mass->GetInternalMassSize()
13               << std::endl;
14
15     std::cout << " All mass:\n" << *mass << std::endl;
16
17     std::cout << " " << mass->Replace(Triangle(1, 2, 2), Triangle(10, 10, 10))
18               << std::endl; //
19     std::cout << " " << mass->Replace(Triangle(3, 3, 3), Triangle(10, 10, 10))
20               << std::endl; //
21
22     std::cout << " All mass:\n" << *mass << std::endl;
23     delete mass;
24
25     return 0;
26 }
27 //tMass.h
28 #ifndef PROG_TMASS_H
29 #define PROG_TMASS_H
30
31 #include "triangle.h"
32 #include "iostream"
33
34 class TMass {
35 public:
36     TMass ();
37
38     TMass (size_t n);
39
40     void AddToFirstFree (const Triangle &tr);
41
42     bool Delete (const Triangle &tr);
43
44     bool Replace (const Triangle &oldT, const Triangle &newT);
45
46     Triangle &GetTriangle (int n) const;
47
```

```
48      size_t GetInternalMassSize ();
49
50      size_t GetItemsCount ();
51
52      friend std::ostream &operator << (std::ostream &os, const TMass &mass);
53
54      ~TMass ();
55
56  private:
57      Triangle *_trianglePrivateMass;
58      size_t _size;
59      size_t _countOfElements;
60      size_t const SIZE_DIFF = 1;
61      size_t const DEFAULT_COUNT = 0;
62
63      void Resize (size_t newSize);
64  };
65
66  #endif // PROG_TMASS_H
67  //tMass.cpp
68  #include "tMass.h"
69
70  TMass::TMass (size_t n) {
71      this->_trianglePrivateMass = new Triangle[n];
72      this->_size = n;
73      this->_countOfElements = DEFAULT_COUNT;
74  #ifndef NDEBUG
75      std::cout << "TMass created with size = " << n << std::endl;
76  #endif
77  }
78
79  std::ostream &operator << (std::ostream &os, const TMass &mass) {
80      for (size_t i = 0; i < mass._countOfElements; ++i) {
81          os << " " << mass.GetTriangle(i) << " " << std::endl;
82      }
83      return os;
84  }
85
86  void TMass::AddToFirstFree (const Triangle &tr) {
87      size_t firstFree = 0;
88      if (_size == 0) {
89          Resize(SIZE_DIFF);
90      } else {
91          Triangle t = Triangle();
92          for (size_t i = 0; i < _size; ++i) {
93              firstFree = i;
94              if (this->_trianglePrivateMass[i] == t) {
95                  break;
96              }
```

```
97          }
98          if (firstFree == _size - SIZE_DIFF) {
99              Resize(_size + SIZE_DIFF);
100             firstFree = _size - SIZE_DIFF;
101         }
102     }
103     this->_trianglePrivateMass[firstFree] = tr;
104     this->_countOfElements++;
105 }
106
107 bool TMass::Delete (const Triangle &tr) {
108     bool res = false;
109
110     for (int i = 0; i < _countOfElements; ++i) {
111         if (this->_trianglePrivateMass[i] == tr) {
112             this->_trianglePrivateMass[i] = Triangle();
113             res = true;
114             break;
115         }
116     }
117     if (res) {
118         this->_countOfElements--;
119     }
120     return res;
121 }
122
123 void TMass::Resize (size_t newSize) {
124 #ifndef NDEBUG
125     std::cout << "TMass resized " << this->_size << "->" << newSize << std::endl;
126 #endif
127
128     Triangle *new_arr = new Triangle[newSize];
129
130     if (this->_size < newSize) {
131         this->_size = newSize;
132     }
133
134     for (size_t i = 0; i < this->_size; ++i)
135         new_arr[i] = _trianglePrivateMass[i];
136
137     delete[] _trianglePrivateMass;
138
139     this->_trianglePrivateMass = new_arr;
140 }
141
142 Triangle &TMass::GetTriangle (int n) const {
143     Triangle res;
144     if (n < _size) {
145         res = this->_trianglePrivateMass[n];
```

```
146     }
147     return res;
148 }
149 
150 TMass::TMass () {
151     this->_trianglePrivateMass = new Triangle[DEFAULT_COUNT];
152     this->_size = DEFAULT_COUNT;
153     this->_countOfElements = DEFAULT_COUNT;
154 #ifndef NDEBUG
155     std::cout << "TMass created with null " << std::endl;
156 #endif
157 }
158 
159 TMass::~TMass () {
160     delete[] _trianglePrivateMass;
161     this->_size = DEFAULT_COUNT;
162     this->_countOfElements = DEFAULT_COUNT;
163 
164 #ifndef NDEBUG
165     std::cout << "TMass deleted " << std::endl;
166 #endif
167 }
168 
169 size_t TMass::GetInternalMassSize () {
170     return this->_size;
171 }
172 
173 size_t TMass::GetItemsCount () {
174     return this->_countOfElements;
175 }
176 
177 bool TMass::Replace (const Triangle &oldT, const Triangle &newT) {
178 #ifndef NDEBUG
179     std::cout << "Try replace (" << oldT << ")->(" << newT << ")" << std::endl;
180 #endif
181     bool res = Delete(oldT);
182     if (res) {
183         AddToFirstFree(newT);
184     }
185 #ifndef NDEBUG
186     if (res) {
187         std::cout << "Sucess replace (" << oldT << ")->(" << newT << ")"
188                 << std::endl;
189     } else {
190         std::cout << "Fail replace (" << oldT << ")->(" << newT << ")" << std::endl;
191     }
192 #endif
193     return res;
194 }
```

```
195 //triangle.h
196 #ifndef TRIANGLE_H
197 #define TRIANGLE_H
198 #include <iostream>
199 #include "figure.h"
200 class Triangle : public Figure {
201  public:
202   Triangle();
203   Triangle(std::istream& is);
204   Triangle(size_t i, size_t j, size_t k);
205   Triangle(const Triangle& orig);
206 
207   double Square() override;
208   void Print() override;
209 
210   virtual ~Triangle();
211 
212  private:
213   size_t _sideA;
214   size_t _sideB;
215   size_t _sideC;
216 };
217 #endif // !TRIANGLE_H
218 //triangle.cpp
219 #include <iostream>
220 #include <cmath>
221 #include "triangle.h"
222 Triangle::Triangle() : Triangle(0, 0, 0)
223 {
224 }
225 
226 Triangle::Triangle(size_t i, size_t j, size_t k) : _sideA(i), _sideB(j), _sideC(k)
227 {
228 #ifndef NDEBUG
229     std::cout << "T created(" << _sideA + ", " << _sideB + ", " << _sideC + ")" << std
        ::endl;
230 #endif
231 }
232 Triangle::Triangle(std::istream &is)
233 {
234     bool ret = true;
235     int sideA;
236     int sideB;
237     int sideC;
238     while (ret)
239     {
240         is >> sideA;
241         is >> sideB;
242         is >> sideC;
```

```
243         if (sideA > 0 && sideB > 0 && sideC > 0)
244         {
245             ret = false;
246             _sideA = sideA;
247             _sideB = sideB;
248             _sideC = sideC;
249         }
250         else
251         {
252             std::cout << "Invalid value, please input one again" << std::endl;
253         }
254     }
255 #ifndef NDEBUG
256     std::cout << "T created(" << _sideA << ", " << _sideB << ", " << _sideC << ")" <<
        std::endl;
257 #endif
258 }
259 double Triangle::Square()
260 {
261     double res = 0.0;
262     double perimetr = (double)(_sideA + _sideB + _sideC);
263     double p = perimetr / 2.;
264     res = sqrt(p * (p - _sideA) * (p - _sideB) * (p - _sideC));
265     return res;
266 }
267 void Triangle::Print()
268 {
269     std::cout << "Figure = Triangle. Sides:" << std::endl;
270     std::cout << "A=" << _sideA << " "
271               << "B=" << _sideB << " "
272               << "C=" << _sideC << " "
273               << std::endl;
274 }
275 Triangle::~Triangle()
276 {
277 #ifndef NDEBUG
278     std::cout << "T destructed" << std::endl;
279 #endif
280 }
```

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.