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

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. Квадрат, треугольник, прямоугольник.

Контейнер. Массив.

1 Теория

Итератор — объект, позволяющий программисту перебирать все элементы коллекции без учёта особенностей её реализации. Предназначен итератор исключительно для последовательного доступа к элементам

Итераторы позволили алгоритмам получать доступ к данным, содержащимся в контейнере, независимо от типа контейнера. Но для этого в каждом контейнере потребовалось определить класс итератора. Таким образом алгоритмы воздействуют на данные через итераторы, которые знают о внутреннем представлении контейнера.

Существует пять категорий итераторов. Ниже описаны категории в порядке возрастания силы.

1. Итератор вывода.
2. Итератор ввода.
3. Однонаправленный.
4. Двунаправленный.
5. По произвольному доступу.

2 Листинг

```
1  //TVector.h
2
3  #ifndef TVECTOR_H
4  #define TVECTOR_H
5
6  #include "iostream"
7  #include <memory>
8
9  template <class T>
10 class Iter {
11 public:
12     Iter (int n, T *arr) {
13         _n = n;
14         _arr = arr;
15     }
16
17     Iter (Iter *i) {
18         this->_n = i->_n;
19         this->_arr = i->_arr;
20     }
21
22     T &operator * () {
23         return _arr[_n];
24     }
25
26     T &operator -> () {
27         return _arr[_n];
28     }
29
30     void operator ++ () {
31         _n++;
32     }
33
34     Iter operator ++ (int) {
35         Iter *tmp = new Iter(*this);
36         ++(*this);
37         return *tmp;
38     }
39
40     bool operator == (Iter const &i) {
41         return _n == i._n &&
42                _arr == i._arr;
43     }
44
45     bool operator != (Iter const &i) {
46         return !(*this == i);
47     }
```

```
48
49  private:
50      int _n;
51      T *_arr;
52  };
53
54  template <class T>
55  class TVector {
56  public:
57
58      void FastPushBack (T &data);
59
60      void Clear ();
61
62      T &operator [] (int index);
63
64      int Size () const;
65
66      int Capacity () const;
67
68      Iter <T> begin ();
69
70      Iter <T> end ();
71
72      TVector &operator = (const TVector <T> &inp);
73
74      TVector ();
75
76      TVector (const TVector <T> &inp);
77
78      friend std::ostream &operator << (std::ostream &os, const TVector <T> &tVector);
79
80      explicit TVector (int n);
81
82      ~TVector ();
83
84  private:
85      T *privateArray;
86      int privateArraySize;
87      int privateArrayOccupiedSize;
88      static int const SIZE_DIFFERENCE = 1;
89      static int const DEFAULT_INT_VALUE = 0;
90  };
91
92  #include "TVector.hpp"
93
94  #endif //TVECTOR_H
95  //TVector.hpp
96  #ifndef TVECTOR_HPP
```

```cpp
#define TVECTOR_HPP

#include "TVector.h"

template <class T>
std::ostream &operator << (std::ostream &os, const TVector <T> &tVector) {
    for (int i = 0; i < tVector.Capacity(); ++i) {
        os << tVector[i] << std::endl;
    }
    return os;
}

template <class T>
void TVector <T>::FastPushBack (T &data) {
    if (privateArraySize == privateArrayOccupiedSize) {
        privateArraySize *= 2;
        T *result = new T[privateArraySize];

        for (int index = DEFAULT_INT_VALUE; index <= privateArrayOccupiedSize; index++)
             {
            if (index != privateArrayOccupiedSize) {
                result[index] = privateArray[index];
            } else {
                result[index] = data;
                break;
            }
        }
        delete[] privateArray;
        privateArray = result;
        privateArrayOccupiedSize++;
    } else {
        privateArray[privateArrayOccupiedSize] = data;
        privateArrayOccupiedSize++;
    }
}

template <class T>
void TVector <T>::Clear () {
    delete[] privateArray;
    privateArraySize = DEFAULT_INT_VALUE;
    privateArrayOccupiedSize = DEFAULT_INT_VALUE;
    privateArray = new T[privateArraySize];
}

template <class T>
T &TVector <T>::operator [] (int index) {
    return this->privateArray[index];
}

```

```
145 template <class T>
146 int TVector <T>::Size () const {
147     return privateArraySize;
148 }
149
150 template <class T>
151 int TVector <T>::Capacity () const {
152     return this->privateArrayOccupiedSize;
153 }
154
155 template <class T>
156 TVector <T> &TVector <T>::operator = (const TVector <T> &inp) {
157     return *this;
158 }
159
160 template <class T>
161 TVector <T>::TVector () {
162     privateArraySize = DEFAULT_INT_VALUE;
163     privateArrayOccupiedSize = DEFAULT_INT_VALUE;
164     privateArray = new T[privateArraySize];
165 }
166
167 template <class T>
168 TVector <T>::TVector (const int n) {
169     privateArraySize = n;
170     privateArrayOccupiedSize = DEFAULT_INT_VALUE;
171     privateArray = new T[privateArraySize];
172 }
173
174 template <class T>
175 TVector <T>::TVector (const TVector <T> &inp) {
176     privateArraySize = inp.privateArraySize;
177     privateArrayOccupiedSize = inp.privateArrayOccupiedSize;
178     privateArray = inp.privateArray;
179 }
180
181 template <class T>
182 Iter<T> TVector<T>::begin () {
183     return Iter<T>(0,privateArray);
184 }
185
186 template <class T>
187 Iter<T> TVector<T>::end () {
188     return Iter<T>(privateArrayOccupiedSize,privateArray);
189 }
190 template <class T>
191 TVector <T>::~TVector () {
192     delete[] privateArray;
193     privateArraySize = DEFAULT_INT_VALUE;
```

```
194     privateArrayOccupiedSize = DEFAULT_INT_VALUE;
195 }
196
197 #endif // TVECTOR_HPP
```

3 Выводы

В данной лабораторной работе я получил навыки программирования итераторов на языке C++, закрепил навык работы с шаблонами классов. Контейнеры очень удобны в спользовании для реализации различных видов алгоритмов внутри контейнера, к примеру сортировка.