Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

Лабораторная работа № 10

по курсу «Языки и методы программирования»

«Реализация итераторов на языке C++»

Студент группы ИУ9-22Б Федуков А. А.

Преподаватель Посевин Д. П.

22 мая 2024 г.

Цель работы

Данная работа предназначена для приобретения навыков разработки контейнерных классов с итераторам.

Задание

Согласно выбранному из таблицы описанию требуется составить контейнерный класс (или шаблон контейнерного класса) и итератор для перебора содержимого объектов этого класса. Если в варианте задания говорится о константном итераторе, значит итератор не должен поддерживать изменение содержимого объектов контейнерного класса.

Задание 1

Последовательность целых 32-разрядных чисел, понимаемая как одно большое двоичное число, с константным однонаправленным итератором по длинам непрерывных последовательностей нулевых битов в нём.

Реализация

Я описал классы в файле заголовков Numbers.h
Сгенерировал его экземпляры и проверил работоспособность необходимых функций уже в main.cpp

Код

Листинг 1: Файл Numbers.h

```
1 #ifndef NUMBERS_H
2 #define NUMBERS_H
3
4 #include <iostream>      // std::cout
5 #include <iterator>      // std::iterator, std::input_iterator_tag
6 #include <vector>
7 #include <initializer_list>
```

```
8
9
10 class MyIterator
11 {
12   int* p;
13 public:
14   MyIterator(int* x) :p(x) {}
15   MyIterator(const MyIterator& mit) : p(mit.p) {}
16   MyIterator& operator++() {++p;return *this;}
17   MyIterator operator++(int) {MyIterator tmp(*this); operator++();
      return tmp;}
18   bool operator==(const MyIterator& rhs) const {return p==rhs.p;}
19   bool operator!=(const MyIterator& rhs) const {return p!=rhs.p;}
20   int& operator*() {return *p;}
21 };
22
23 using namespace std;
24
25 inline string getBits(long long x)
26 {
27     string s = "";
28     while (x > 0)
29     {
30         s = to_string(x % 2) + s;
31         x /= 2;
32     }
33
34     return s;
35 }
36
37 template <typename T>
38 class Numbers
39 {
40 private:
41     vector<string> dataBits;
42     T* store;
43     size_t size;
44     typedef T* iterator;
45     typedef const T* const_iterator;
46
47 public:
48     void add(T x);
49     void updateIter();
50     string get();
51     Numbers(std::initializer_list<double> values);
52     ~Numbers();
```

```
53      iterator begin() { return &store[0]; }
54      const_iterator begin() const { return &store[0]; }
55      iterator end() { return &store[size]; }
56      const_iterator end() const { return &store[size]; }
57  };
58  template <typename T>
59  Numbers<T>::~Numbers(){
60      if (store)
61      {
62          delete[] store;
63      }
64
65  }
66  template <typename T>
67  string Numbers<T>::get(){
68      string s = "";
69      for (auto &&i : dataBits)
70          s += i;
71      return s;
72  }
73
74  template <typename T>
75  void Numbers<T>::add(T x)
76  {
77      dataBits.push_back(getBits(x));
78      updateIter();
79  }
80
81  template <typename T>
82  Numbers<T>::Numbers(std::initializer_list<double> values)
83  {
84      for (auto &&x : values)
85          dataBits.push_back(getBits(x));
86      updateIter();
87
88  }
89
90  template <typename T>
91  void Numbers<T>::updateIter()
92  {
93      int c = 0;
94      vector<int> nums;
95      for (auto &&num : dataBits)
96      {
97          for (auto &&chr : num)
98          {
```

```
99              if (chr == '0')
100                 c += 1;
101             else
102             {
103                 if (c != 0)
104                     nums.push_back(c);
105                 c = 0;
106             }
107         }
108     }
109     if (c != 0)
110         nums.push_back(c);
111
112     size = nums.size();
113
114     store = new int[size];
115     copy(nums.begin(), nums.end(), store);
116
117 }
118
119 #endif
```

Листинг 2: Файл main.cpp

```
1
2 #include "Numbers.h"
3 #include <cmath>
4
5 int main()
6 {
7     cout << "From " << pow(2, 32) - 1 << " to " << pow(2, 33) << endl;
8     Numbers<int> numbers{pow(2, 32) * 2, 20, 30, 40, 50};
9
10    // Joined bit form
11    cout << numbers.get() << endl;
12
13    // Print lengths of 0 row in bit form
14    for (MyIterator it = numbers.begin(); it != numbers.end(); it++)
15        std::cout << *it << ' ';
16    std::cout << '\n';
17
18    numbers.add(1);
19    cout << numbers.get() << endl;
20
21    for (auto &&i : numbers)
22        cout << i << " ";
23    cout << endl;
```

```
24
25      return 0;
26  }
```

Вывод программы

Программа создала экземпляры класса и проитерировала числа по количеству подряд идущих нулей.

Листинг 3: Вывод программы

```
1 From 4.29497e+09 to 8.58993e+09
2 10000000000000000000000000000001010011110101000110010
3 33 1 2 1 1 3 2 1
4 10000000000000000000000000000001010011110101000110101
5 33 1 2 1 1 3 2 1
```

Задание 2

Множество целых чисел с константным однонаправленным итератором по всем тройкам чисел, которые могут представлять длины сторон прямоугольного треугольника.

Реализация

Я описал класс в файле заголовков Nums.h
Сгенирировал его экземпляры и проверил работоспособность необходимых функций уже в main.cpp

Код

Листинг 4: Файл Nums.h

```
1 #ifndef NUMS_H
2 #define NUMS_H
3
4 #include <vector>
5 #include <cmath>
6 #include <iostream>
```

```
7
8  using namespace std;
9
10
11 struct Triplet
12 {
13     int *data;
14     Triplet(){data = new int[3];}
15     Triplet(int a, int b, int c) : Triplet(){ data[0] = a; data[1] = b;
       data[2] = c; }
16     Triplet(const Triplet& trp): Triplet(trp.data[0], trp.data[1], trp.
       data[2]){}
17     ~Triplet(){if (data) delete[] data;}
18     int* operator*(){return data;}
19     Triplet& operator=(const Triplet& trp) {
20         if (this == &trp) return *this;
21         delete[] data;
22         data = new int[3];
23         copy(trp.data, trp.data + 3, data);
24         return *this;
25     }
26 };
27
28
29 class MyIterator
30 {
31   Triplet* p;
32 public:
33   MyIterator(Triplet* x) :p(x) {}
34   MyIterator(const MyIterator& mit) : p(mit.p) {}
35   MyIterator& operator++() {++p;return *this;}
36   MyIterator operator++(int) {MyIterator tmp(*this); operator++();
      return tmp;}
37   bool operator==(const MyIterator& rhs) const {return p==rhs.p;}
38   bool operator!=(const MyIterator& rhs) const {return p!=rhs.p;}
39
40   Triplet& operator*() {return *p;}
41   Triplet* operator->() {return p;}
42 };
43
44
45
46 class Nums
47 {
48 public:
49     typedef MyIterator iterator;
```

```
50
51 private:
52     vector<int> values;
53     Triplet* iterStore;
54     size_t iterSize = 0;
55     bool condition(const int a, const int b, const int c){
56         bool isMet = false;
57         int x = max(max(a, b), max(b, c));
58         if (x == a)
59             isMet = (pow(a, 2) == pow(b, 2) + pow(c, 2));
60         if (x == b)
61             isMet = (pow(b, 2) == pow(a, 2) + pow(c, 2));
62         if (x == c)
63             isMet = (pow(c, 2) == pow(a, 2) + pow(b, 2));
64
65         return isMet;
66     }
67     void updateStore(){
68         vector<Triplet> str;
69         for (size_t i = 0; i < values.size() - 2; i++)
70             for (size_t j = i+1; j < values.size() - 1; j++)
71                 for (size_t k = j+1; k < values.size(); k++)
72                     if (condition(values[i], values[j], values[k]))
73                         str.push_back(Triplet {values[i], values[j],
    values[k]});
74
75         iterSize = str.size();
76         iterStore = new Triplet[iterSize];
77         copy(str.begin(), str.end(), iterStore);
78     }
79
80 public:
81     Nums(initializer_list<int> xs): values(xs) {updateStore();};
82     ~Nums(){ if (iterStore) delete [] iterStore;}
83     iterator begin() { return MyIterator(iterStore); }
84     iterator end() { return MyIterator(iterStore + iterSize); }
85
86 };
87
88
89 #endif
```

Листинг 5: Файл main.cpp

```
1 #include <iostream>
2 #include "Nums.h"
3
```

```
4  int main()
5  {
6
7      Nums n{3, 4, 5, 12, 13};
8      for (auto &&i : n)
9          cout << "Square triangle: " << i.data[0] << " " << i.data[1] <<
       " " << i.data[2] << endl;
10     cout << "Manually: " << endl;
11     for (MyIterator i = n.begin(); i != n.end(); i++)
12         cout << "Square triangle: " << i->data[0] << " " << i->data[1]
       << " " << i->data[2] << endl;
13
14     return 0;
15 }
```

Вывод программы

Программа создала экземляры класса и протестировала все заявленные операции

Листинг 6: Вывод программы

```
1 Square triangle: 3 4 5
2 Square triangle: 5 12 13
3 Manually:
4 Square triangle: 3 4 5
5 Square triangle: 5 12 13
```

Вывод

По ходу выполнения данной лабораторной работы, я научился писать классы итераторы для других классов.