МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №6f

по дисциплине: Основы программирования
Тема: "Структуры в стиле C++"

Выполнил: ст. группы ПВ-211
Чувилко Илья Романович

Проверили:
Притчин Иван Сергеевич
Черников Сергей Викторович

Белгород 2022 г.

Содержание

1 Fraction

Fraction – структура для работы с дробями.

1.1 Заголовочный файл *Fraction.h*:

```
#include <cassert>
#include <iostream>

#include "../algorithms/algorithms.h"

struct Fraction {
  int numerator = 0;
  int denumerator = 1;

  // Конструктор для Fraction
  Fraction() = default;

  // Конструктор для Fraction
  Fraction(Fraction &f);

  // Конструктор для Fraction
  Fraction(int n, int d);

  // Возвращает true, если дробь положительна
  bool isPositive() const;

  // Приводит Fraction к нормальному виду
  void bringingToNormal();

  // Возвращает упрощенную дробь f структуры Fraction
  void simpleFraction();

  // Возвращает дробь, которая является результатом произведения дроби f1 и f2
  Fraction operator*(Fraction &other) const;

  // Возвращает дробь, которая является результатом делением дроби f1 на f2
  Fraction operator/(Fraction &other) const;

  // Возвращает дробь, которая является разницей f1 и f2
  Fraction operator-(Fraction &f);

  // Возвращает дробь, которая является суммой f1 и f2
  Fraction operator+(Fraction &f) const;
};

void tests_Fraction();
```

1.2 Файл реализации *Fraction.cpp*:

```
#include "Fraction.h"

using namespace std;

// Конструктор для Fraction
Fraction::Fraction(int n, int d) {
  numerator = n;
```

```
8     denumerator = d;
9     bringingToNormal();
10  }
11
12  // Конструктор для Fraction
13  Fraction::Fraction(Fraction &f) {
14    numerator = f.numerator;
15    denumerator = f.denumerator;
16    bringingToNormal();
17  }
18
19  // Возвращает true, если дробь положительна
20  bool Fraction::isPositive() const {
21    return (numerator >= 0) == (denumerator >= 0);
22  }
23
24  // Приводит Fraction к нормальному виду
25  void Fraction::bringingToNormal() {
26    if (numerator == 0) {
27      denumerator = 1;
28      return;
29    }
30    simpleFraction();
31    if (isPositive())
32      numerator = abs(numerator);
33    else
34      numerator = -abs(numerator);
35    denumerator = abs(denumerator);
36  }
37
38  // Возвращает упрощенную дробь f структуры Fraction
39  void Fraction::simpleFraction() {
40    assert(denumerator != 0);
41    int d = gcd(numerator, denumerator);
42    numerator /= d;
43    denumerator /= d;
44  }
45
46  // Возвращает дробь, которая является результатом произведения дроби f1 и f2
47  Fraction Fraction::operator*(Fraction &other) const {
48    Fraction res(this->numerator * other.numerator,
49                 this->denumerator * other.denumerator);
50    res.bringingToNormal();
51    return res;
52  }
53
54  // Возвращает дробь, которая является результатом делением дроби f1 на f2
55  Fraction Fraction::operator/(Fraction &other) const {
56    Fraction tmp1(other.denumerator, other.numerator);
57    Fraction tmp2((Fraction &) *this);
58    Fraction res;
59    res = tmp2 * tmp1;
60    return res;
61  }
62
63  // Возвращает дробь, которая является суммой f1 и f2
64  Fraction Fraction::operator+(Fraction &f) const {
65    int d = lcm(this->denumerator, f.denumerator);
66    Fraction res(this->numerator * (d / this->denumerator) +
67                 f.numerator * (d / f.denumerator),
```

```
                d);
  res.bringingToNormal();
  return res;
}

// Возвращает дробь, которая является разницей f1 и f2
Fraction Fraction::operator-(Fraction &f) {
  Fraction temp(-f.numerator, f.denumerator);
  Fraction res;
  res = temp + *this;
  return res;
}
```

Результаты работы программы:

1.3 Тесты:

```
void test_Fraction_constructor1() {
  Fraction f{};

  assert(f.numerator == 0);
  assert(f.denumerator == 1);
}

void test_Fraction_constructor2() {
  Fraction f1{1, 2};
  Fraction f2{f1};

  assert(f1.numerator == 1);
  assert(f1.denumerator == 2);
  assert(f1.numerator == f2.numerator);
  assert(f1.denumerator == f2.denumerator);
}

void test_Fraction_constructor3() {
  Fraction f1{2, 4};

  assert(f1.numerator == 1);
  assert(f1.denumerator == 2);
}

void test_Fraction_constructor4() {
  Fraction f1{0, 4};

  assert(f1.numerator == 0);
  assert(f1.denumerator == 1);
}

void test_Fraction_constructor5() {
  Fraction f1{-1, -2};

  assert(f1.numerator == 1);
  assert(f1.denumerator == 2);
}

void test_Fraction_constructor6() {
  Fraction f1{1, -2};

  assert(f1.numerator == -1);
  assert(f1.denumerator == 2);
```

```
}

void test_Fraction_constructor() {
  test_Fraction_constructor1();
  test_Fraction_constructor2();
  test_Fraction_constructor3();
  test_Fraction_constructor4();
  test_Fraction_constructor5();
  test_Fraction_constructor6();
}

void test_Fraction_mul() {
  Fraction f1(2, 5);
  Fraction f2(10, 8);

  Fraction res{};
  res = f1 * f2;

  assert(res.numerator == 1);
  assert(res.denumerator == 2);
}

void test_Fraction_div() {
  Fraction f1(5, 2);
  Fraction f2(10, 8);

  Fraction res{};
  res = f1 / f2;

  assert(res.numerator == 2);
  assert(res.denumerator == 1);
}

void test_Fraction_plus() {
  Fraction f1(5, 2);
  Fraction f2(10, 8);

  Fraction res{};
  res = f1 + f2;

  assert(res.numerator == 15);
  assert(res.denumerator == 4);
}

void test_Fraction_minus() {
  Fraction f1(5, 2);
  Fraction f2(10, 8);

  Fraction res{};
  res = f1 - f2;

  assert(res.numerator == 5);
  assert(res.denumerator == 4);
}

void tests_Fraction() {
  test_Fraction_constructor();
  test_Fraction_mul();
  test_Fraction_div();
  test_Fraction_plus();
```

```
104     test_Fraction_minus();
105 }
```

2 Material Point

MaterialPoint – структура для работы с материальными точками.

2.1 Заголовочный файл *MaterialPoint.h*:

```
1
2 #include <iostream>
3 #include <cmath>
4 #include <cassert>
5
6 class MaterialPoint {
7 private:
8   double _position = 0;         // Положение материальной точки
9   double _speed = 0;           // Скорость материальной точки
10  double _acceleration = 0;   // Ускорение материальной точки
11
12 public:
13  // Конструктор материальной точки
14  explicit MaterialPoint(double speed = 0, double acceleration = 0);
15
16  // Сеттер скорости
17  void setSpeed(double speed);
18
19  // Сеттер ускорения
20  void setAcceleration(double acceleration);
21
22  // Геттер позиции
23  double getPosition() const;
24
25  // Геттер скорости
26  double getSpeed() const;
27
28  double getAcceleration() const;
29
30  // Двигает материальную точку по числовой прямой на протяжении duration секунд
31  void move(int duration);
32 };
33
34 void tests_MaterialPoint();
```

2.2 Файл реализации *.cpp*:

```
1
2 #include "MaterialPoint.h"
3
4 // Конструктор материальной точки
5 MaterialPoint::MaterialPoint(double speed, double acceleration) {
6   _speed = speed;
7   _acceleration = acceleration;
8 }
9
10 // Сеттер скорости
11 void MaterialPoint::setSpeed(double speed) {
12   _speed = speed;
13 }
14
```

```
// Сеттер ускорения
void MaterialPoint::setAcceleration(double acceleration) {
  _acceleration = acceleration;
}

// Геттер позиции
double MaterialPoint::getPosition() const {
  return _position;
}

// Геттер скорости
double MaterialPoint::getSpeed() const {
  return _speed;
}

double MaterialPoint::getAcceleration() const {
  return _acceleration;
}

// Двигает материальную точку по числовой прямой на протяжении duration секунд
void MaterialPoint::move(int duration) {
  _position += duration * _speed + _acceleration * pow(duration, 2) / 2;
  _speed += duration * _acceleration;
}
```

2.3 Тесты:

```
void tests_MaterialPoint_constructor1() {
  MaterialPoint m;

  assert(m.getPosition() == 0);
  assert(m.getSpeed() == 0);
  assert(m.getAcceleration() == 0);
}

void tests_MaterialPoint_constructor2() {
  MaterialPoint m(1);

  assert(m.getPosition() == 0);
  assert(m.getSpeed() == 1);
  assert(m.getAcceleration() == 0);
}

void tests_MaterialPoint_constructor3() {
  MaterialPoint m(1, 2);

  assert(m.getPosition() == 0);
  assert(m.getSpeed() == 1);
  assert(m.getAcceleration() == 2);
}

void tests_MaterialPoint_constructor() {
  tests_MaterialPoint_constructor1();
  tests_MaterialPoint_constructor2();
  tests_MaterialPoint_constructor3();
}

void tests_MaterialPoint() {
  tests_MaterialPoint_constructor();
```

```
33 }
```

3 BitSet

BitSet – структура для хранения множества в битовом представлении.

3.1 Заголовочный файл *BitSet.h*:

```
1
2  #include <iostream>
3  #include <string>
4  #include <vector>
5  #include <cassert>
6
7  using namespace std;
8
9  class BitSet {
10   private:
11   // максимальное поддерживаемое значение битсетом как таковым
12   static const int _MAX_VALUE_SUPPORTED = 31;
13   // максимальное поддерживаемое значение
14   // для создаваемой структуры
15   const int _maxValue = _MAX_VALUE_SUPPORTED;
16   // поле для хранения значений
17   uint32_t _values = 0;
18   // мощность множества
19   unsigned _power = 0;
20
21   public:
22   BitSet() = default;
23
24   BitSet(uint32_t maxValue);
25
26   BitSet(const vector<int> &v);
27
28   uint32_t getValues() const;
29
30   void setValues(uint32_t values);
31
32   unsigned getPower() const;
33
34   bool empty() const;
35
36   bool find(unsigned x) const;
37
38   void insert(unsigned x);
39
40   void erase(unsigned x);
41
42   bool operator==(BitSet &other) const;
43
44   bool operator!=(BitSet &other) const;
45
46   bool isSubset(const BitSet &set) const;
47
48   static BitSet intersection_(const BitSet &lhs, const BitSet &rhs);
49
50   static BitSet difference_(const BitSet &lhs, const BitSet &rhs);
51
52   static BitSet symmetricDifference_(const BitSet &lhs,
53                                      const BitSet &rhs);
```

```
54
55    static BitSet complement_(const BitSet &set);
56
57    friend ostream& operator<<(std::ostream &out, const BitSet &f);
58 };
59
60 void tests_BitSet();
```

3.2 Файл реализации *BitSet.cpp*:

```
1  #include "BitSet.h"
2
3
4  BitSet::BitSet(uint32_t maxValue) {
5    insert(maxValue);
6  }
7
8  BitSet::BitSet(const vector<int> &v) {
9    for (auto &i: v) {
10     insert(i);
11   }
12 }
13
14 uint32_t BitSet::getValues() const {
15   return _values;
16 }
17
18 void BitSet::setValues(uint32_t values) {
19   _values = values;
20   _power = 0;
21   for (int i = 0; i < 32; i++)
22     if (find(i))
23       _power++;
24 }
25
26 unsigned BitSet::getPower() const {
27   return _power;
28 }
29
30 bool BitSet::empty() const {
31   return _power == 0;
32 }
33
34 bool BitSet::find(unsigned x) const {
35   return (_values >> x) % 2;
36 }
37
38 void BitSet::insert(unsigned x) {
39   try {
40     if (x > _MAX_VALUE_SUPPORTED)
41       throw (string) "invalid_argument: x > 31";
42     if (!find(x)) {
43       _values |= 1 << x;
44       _power++;
45     }
46   } catch (const string &s) {
47     cerr << s;
48     exit(2);
49   }
```

```
50 }
51
52 void BitSet::erase(unsigned x) {
53   try {
54     if (x > _MAX_VALUE_SUPPORTED)
55       throw (string) "invalid_argument: x > 31";
56     if (find(x)) {
57       _values &= ~(0 & (1 << x));
58       _power--;
59     }
60   } catch (const string &s) {
61     cerr << s;
62     exit(2);
63   }
64 }
65
66 bool BitSet::operator==(BitSet &other) const {
67   return this->_values == other.getValues();
68 }
69
70 bool BitSet::operator!=(BitSet &other) const {
71   return this->_values != other.getValues();
72 }
73
74 bool BitSet::isSubset(const BitSet &set) const {
75   return set.getValues() == (set.getValues() | this->getValues());
76 }
77
78 BitSet BitSet::intersection_(const BitSet &lhs, const BitSet &rhs) {
79   BitSet res(0);
80   res.setValues(lhs.getValues() & rhs.getValues());
81   return res;
82 }
83
84 BitSet BitSet::difference_(const BitSet &lhs, const BitSet &rhs) {
85   BitSet res{};
86   res.setValues(intersection_(lhs, complement_(rhs)).getValues());
87   return res;
88 }
89
90 BitSet BitSet::symmetricDifference_(const BitSet &lhs,
91                                     const BitSet &rhs) {
92   BitSet res{};
93   res.setValues(
94     difference_(lhs, rhs).getValues() |
95     difference_(rhs, lhs).getValues());
96   return res;
97 }
98
99 BitSet BitSet::complement_(const BitSet &set) {
100   BitSet res(0);
101   res.setValues(~set.getValues());
102   return res;
103 }
104
105 ostream &operator<<(ostream &out, const BitSet &f) {
106   out << '{';
107   for (int i = 0; i < 31; i++)
108     if (f.find(i))
109       out << i << ", ";
```

```
  out << "\b\b}";
  return out;
}

void test_getValues() {
  BitSet a({1, 2, 3, 4});
  uint32_t res = 0;
  res += 1 << 1;
  res += 1 << 2;
  res += 1 << 3;
  res += 1 << 4;

  assert(a.getValues() == res);
}
```

3.3 Тесты:

```
void test_find1() {
  BitSet a(4);

  assert(a.find(4));
}

void test_find2() {
  BitSet a(4);

  assert(a.find(4));
}

void test_find() {
  test_find1();
  test_find2();
}

void test_getPower1() {
  BitSet a({1, 2, 3, 4, 5, 6, 7, 8, 9, 10});

  assert(a.getPower() == 10);
}

void test_getPower2() {
  BitSet a({1, 2, 3, 4, 5, 6, 7, 8, 9, 10});
  a.erase(1);
  a.erase(10);

  assert(a.getPower() == 8);
}

void test_getPower() {
  test_getPower1();
  test_getPower2();
}

void test_operators_equal() {
  BitSet a({1, 2, 3});
  BitSet b({1, 2, 3});

  assert(a == b);
}
```

```
43
44 void test_operators_notEqual() {
45   BitSet a({1, 2, 3});
46   BitSet b({1, 2});
47
48   assert(a != b);
49 }
50
51 void test_operators() {
52   test_operators_equal();
53   test_operators_notEqual();
54 }
55
56 void test_isSubset1() {
57   BitSet a({1, 2, 3});
58   BitSet b({1, 2});
59
60   assert(b.isSubset(a));
61 }
62
63 void test_isSubset2() {
64   BitSet a({1, 2, 3});
65   BitSet b({1, 2});
66
67   assert(!a.isSubset(b));
68 }
69
70 void test_isSubset() {
71   test_isSubset1();
72   test_isSubset2();
73 }
74
75 void test_intersection() {
76   BitSet a({1, 2, 3, 4});
77   BitSet b({3, 4, 5, 6});
78   BitSet c = BitSet::intersection_(a, b);
79
80   BitSet res({3, 4});
81
82   assert(res == c);
83   assert(c.getPower() == res.getPower());
84 }
85
86 void test_difference() {
87   BitSet a({1, 2, 3, 4});
88   BitSet b({3, 4, 5, 6});
89   BitSet c = BitSet::difference_(a, b);
90
91   BitSet res({1, 2});
92
93   assert(res == c);
94   assert(c.getPower() == res.getPower());
95 }
96
97 void test_symmetricDifference() {
98   BitSet a({1, 2, 3, 4});
99   BitSet b({3, 4, 5, 6});
100  BitSet c = BitSet::symmetricDifference_(a, b);
101
102  BitSet res({1, 2, 5, 6});
```

```
103
104   assert(res == c);
105   assert(c.getPower() == res.getPower());
106 }
107
108 void tests_BitSet() {
109   test_getValues();
110   test_find();
111   test_getPower();
112   test_operators();
113   test_isSubset();
114   test_intersection();
115   test_difference();
116   test_symmetricDifference();
117 }
```

4 Prefix Sum

PrefixSum – структура для хранения префиксных сумм.

4.1 Заголовочный файл *PrefixSum.h*:

```
1
2 #include <iostream>
3 #include <vector>
4 #include <cassert>
5
6 using namespace std;
7
8 struct PrefixSum {
9   private:
10  vector<int> prefixSum_{};
11
12  public:
13  PrefixSum(const int *a, size_t n);
14
15  PrefixSum(const vector<int> &a);
16
17  vector<int> &getPrefixSum();
18
19 // возвращает сумму элементов от l до r не включая r.
20 // например, для массива 1, 3, 5
21 // запрос getSum(0, 2) должен возвращать 1 + 3 = 4;
22  long long getSum(int l, int r);
23
24 // данный запрос должен каким-то образом,
25 // без хранения старого массива
26 // обновлять префиксный массив так, как будто мы
27 // заменили i-ый элемент старого массива на v
28  void set(size_t i, int v);
29 };
30
31 void tests_PrefixSum();
```

4.2 Файл реализации *PrefixSum.cpp*:

```
1 #include "PrefixSum.h"
2
3 PrefixSum::PrefixSum(const int *a, size_t n) {
4   int s = 0;
5   prefixSum_.push_back(s);
6   for (size_t i = 0; i < n; i++) {
7     s += a[i];
8     prefixSum_.push_back(s);
9   }
10 }
11
12 PrefixSum::PrefixSum(const vector<int> &a) {
13   int s = 0;
14   prefixSum_.push_back(s);
15   for (auto &i: a) {
16     s += i;
17     prefixSum_.push_back(s);
```

```
  }
}

vector<int> &PrefixSum::getPrefixSum() {
  return prefixSum_;
}

long long PrefixSum::getSum(int l, int r) {
  return prefixSum_[r] - prefixSum_[l];
}

void PrefixSum::set(size_t i, int v) {
  int iElementOld = prefixSum_[i + 1] - prefixSum_[i];
  int diff = v - iElementOld;
  for (size_t j = i + 1; j < prefixSum_.size(); j++)
    prefixSum_[j] += diff;
}
```

4.3 Тесты:

```
void test_constructor1() {
  int a[] = {1, 2, 5, 7};
  PrefixSum s(a, 4);

  vector<int> res{0, 1, 3, 8, 15};

  assert(res == s.getPrefixSum());
}

void test_constructor2() {
  vector<int> v{1, 2, 5, 7};
  PrefixSum s(v);

  vector<int> res{0, 1, 3, 8, 15};

  assert(res == s.getPrefixSum());
}

void test_constructor() {
  test_constructor1();
  test_constructor2();
}

void test_getSum() {
  vector<int> v{1, 3, 6, 9};
  PrefixSum s(v);

  assert(s.getSum(1, 4) == 18);
}

void test_set() {
  vector<int> v{1, 3, 6, 9};
  PrefixSum s(v);

  s.set(2, 4);

  vector<int> res{0, 1, 4, 8, 17};
  assert(s.getPrefixSum() == res);
}
```

```
40
41 void tests_PrefixSum() {
42   test_constructor();
43   test_getSum();
44   test_set();
45 }
```