ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Департамент прикладной математики

ОТЧЕТ
К ЛАБОРАТОРНОЙ РАБОТЕ 1
по дисциплине «Компьютерный практикум»

Работу выполнила
студентка группы БПМ 173 _____ М.В. Самоделкина
дата, подпись

Работу проверил _____ С.А. Булгаков
дата, подпись

Москва 2019

Содержание

Постановка задачи

Разработать класс, объект которого реализует тип данных, указанный в варианте. Обеспечить его произвольную размерность за счет использования в объекте динамических структур данных. Разработать необходимые конструкторы, деструктор, конструктор копирования, а также методы, обеспечивающие изменение отдельных составных частей объекта (например, коэффициентов полинома) и вывод его содержимого. Выполнить задания в соответствии с вариантом 4: целое произвольной длины во внешней форме представления в виде строки символов-цифр.

1 Основная часть

1.1 Общая идея решения задачи

Для решения задачи были использованы:

1. Алгоритмы стандартных библиотек <*algorithm*>: *max*, <*string*>: *push_back, reverse*.
2. Контейнер <*string*>.

Для обеспечения произвольной размерности использовался массив из элементов типа *char*. Был разработан класс *BigInt* со всеми необходимыми методами.

1.2 Структура и принципы действия

Для выполнения задания был создан класс *BigInt*, содержащий закрытые поля типа *char* value* - хранит одну символ-цифру числа без учета знака в одном элементе массива (для хранения используется десятичная система счисления), где единичный разряд - нулевой элемент массива, *size_t size* - хранит количество цифр в числе, *bool sign* - хранит знак числа.

Класс содержит конструктор с параметром (принимает строку): если в начале строки стоит знак минус, то *sign* примет значение *false*, инчает *true*. Также происходит проверка на то, что в строке все элементы цифры, в противном случае выбрасывается исключение. Конструктор инициализирует длину и заполняет массив элементами строки. Класс также содержит конструктор копирования и деструктор. Также реализована возможность вывода числа в поток с помощью перегрузки оператора *operator«*.

Для выполнения задачи мне потребовались следующие методы: аксессоры к закрытым полям (*getValue*, *getSize*, *getSign*), возможность изменения знака числа (*setSign*, *operator-*), возможность сравнения чисел (*operator<*, *operator==*), копирующий оператор присваивания. С помощью них были реализованы методы, обеспечивающие изменение отдельных составных частей объекта (в данном случае изменение самого числа и его знака) - методы сложения и вычитания (перегрузка операторов *operator+*, *operator-*, *operator+=*, *operator-=*).

Сложение было реализовано следующим образом: при равенстве знаков у чисел происходило посимвольное сложение, начиная с младшего разряда, также отдельная переменная учитывала перенос десятков к следующему разрядку. Если оба числа

были отрицательными, то на результат навешивался знак минус. Если числа были разных знаков, то результат сводился к операции разности.

Для операции разности была использована возможность сравнения чисел. Если оба числа положительные и уменьшаемое больше вычитаемого, то происходило посимвольное вычитание с возможностью занятия десятка у старшего разряда, иначе менялся порядок вычитания и у результата менялся знак на минус. При наличии разных знаков у чисел результат сводился к сложению. Если оба числа отрицательные, то меняли порядок разности, изменяя знаки чисел соответствующим образом.

1.3 Процедура получения исполняемых программных модулей

Программный код был скомпилирован с среде *Visual Studio 2017*. Компиляция раздельная. Код программы содержится в разных файлах. Никаких дополнительных ключей не добавлялось, использовались ключи, которые добавляются по умолчанию.

1.4 Результаты тестирования

Для тестирования программы был применен механизм UnitTest, все тесты были пройдены успешно. Также в функции main файла Main.cpp представлено дополнительное тестирование. Ожидаемый вывод функции:

```
-126765060022822940149670320537 6
-126765060022822940149670320537 6

128667160846894790715675442493073
1
265613988875874769338781322035779626829233452653394495974 57
```

Приложение А

полный код программы

А.1 - Main.cpp

```
// This is a personal academic project. Dear PVS-Studio, please check it.
// PVS-Studio Static Code Analyzer for C, C++, C#, and Java: http://www.viva64.com
#include "BigInt.h"

int main() {
  try {
    BigInt bi1("-1267650600228229401496703205376");
    BigInt bi2(bi1);
    BigInt bi3("-1299348114471230201171721456984490");
    BigInt bi4("");
    cout << bi1 << bi2 << bi4 << (bi1 - bi3);
  }
  catch (const invalid_argument& e) {
    cerr << e.what() << endl;
  }
  BigInt b1("1606938044258990275541962092341162602522202993782792835301376");
  BigInt b2("-2656139888758747693387813220357796268292334526533944959745749617390920
  ");
  BigInt b3("-2656139888758747693387813220357796252222954083944042204326128693979290
  ");
  b2 += b1;
  cout << (b2 == b3);
  cout << -b2;
  return 0;
}
```

А.2 - BigInt.h

```
// This is a personal academic project. Dear PVS-Studio, please check it.
// PVS-Studio Static Code Analyzer for C, C++, C#, and Java: http://www.viva64.com
#pragma once
//#include <cstring>
#include <string>
#include <iostream>
#include <algorithm>

using namespace std;

const int base = 10;

class BigInt {
  char* value; //□□□□□□□□□□
    □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ -
  size_t size;
```

```
16    bool sign;//pos - true, neg - false
17  public:
18    BigInt(string s) {
19      int i;
20      if (s[0] != '-') {
21        sign = true;
22        size = s.length();
23        i = s.length() - 1;
24      }
25      else {
26        sign = false;
27        size = s.length() - 1;
28        i = s.length() - 1;
29      }
30      value = new char[size];
31      int j = 0;
32      while (j < size) {
33        if (!isdigit(s[i])) {
34          throw invalid_argument("received not digit");
35        }
36        else
37          value[j++] = s[i--];
38      }
39    }
40    BigInt(const BigInt &newInt) {
41      char* newVal = newInt.getValue();
42      sign = newInt.getSign();
43      size = newInt.getSize();
44      value = new char[size];
45      memcpy(value, newVal, size * sizeof(char));
46    }
47    ~BigInt() {
48      delete[] value;
49    }
50    char* getValue() const {
51      return value;
52    }
53    unsigned int getSize() const {
54      return size;
55    }
56    bool getSign() const {
57      return sign;
58    }
59    void setSign(bool b) {
60      sign = b;
61    }
62    BigInt operator=(const BigInt &rBi) {
63      size = rBi.size;
64      sign = rBi.sign;
65      char* stmp = rBi.value;
66      delete[] value;
67      value = new char[size];
68      int i = 0;
69      while (i < size) {
70        if (!isdigit(stmp[i])) {
71          throw invalid_argument("received not digit");
72        }
```

```
73        else
74          value[i] = stmp[i];
75        i++;
76      }
77      return *this;
78    }
79
80    bool operator<(const BigInt &rBi) const {
81      bool rSign = rBi.sign;
82      if (sign != rSign)
83        return sign == true ? true : false;
84      unsigned int rSize = rBi.size;
85      if (size != rSize)
86        return size < rSize;
87      char* valBi = rBi.value;
88      int i = size - 1;
89      while (i >= 0) {
90        if (value[i] > valBi[i])
91          return false;
92        else if (value[i] < valBi[i])
93          return true;
94        i--;
95      }
96      return true;
97    }
98    bool operator==(const BigInt &rBi) const {
99      bool result = sign == rBi.sign && size == rBi.size;
100     for (int i = 0; i < size && result; i++) {
101       result = value[i] == rBi.value[i];
102     }
103     return result;
104   }
105   BigInt operator-() {
106     sign = !(sign);
107     return *this;
108   }
109   BigInt operator-(const BigInt &rBi) const {
110     bool rSign = rBi.sign;
111     if (sign && rSign) {
112       if (*this < rBi) {
113         BigInt tmp = rBi - *this;
114         tmp.setSign(false);
115         return tmp;
116       }
117       char* valBi = rBi.value;
118       unsigned int rSize = rBi.size;
119       string tmp = "";
120       int rem = 0;
121       for (int i = 0; i < max(size, rSize); i++) {
122         if (i >= rSize) {
123           tmp.push_back(((value[i] - 48) + rem + 20) % 10 + 48);
124           if ((value[i] - 48) + rem < 0)
125             rem = -1;
126           else {
127             rem = 0;
128           }
129         }
```

```
130           else {
131             char a = valBi[i];
132             char b = value[i];
133             tmp.push_back((-(valBi[i] - 48) + (value[i] - 48) + rem + 20)
                    % 10 + 48);
134             if (-(valBi[i] - 48) + (value[i] - 48) + rem < 0)
135               rem = - 1;
136             else {
137               rem = 0;
138             }
139           }
140         }
141         if (rem) {
142           tmp.push_back(abs(rem) + 48);
143         }
144         int i = 0;
145         while (tmp[i] == '0') {
146           tmp.pop_back();
147           i++;
148         }
149         reverse(tmp.begin(), tmp.end());
150         return BigInt(tmp);
151       }
152       else if (!sign && rSign) {
153         BigInt tmp1(*this);
154         tmp1.setSign(true);
155         BigInt tmp = tmp1 + rBi;
156         tmp.setSign(false);
157         return tmp;
158       }
159       else if (sign && !rSign) {
160         BigInt tmp(rBi);
161         tmp.setSign(true);
162         return *this + tmp;
163       }
164       else {
165         BigInt tmp(rBi);
166         BigInt tmp1(*this);
167         tmp.setSign(true);
168         tmp1.setSign(true);
169         return tmp - tmp1;
170       }
171
172     }
173     BigInt operator+(const BigInt &rBi) const {
174       char* valBi = rBi.value;
175       bool rSign = rBi.sign;
176       unsigned int rSize = rBi.size;
177       string tmp = "";
178       if (sign == rSign) {
179         unsigned int rem = 0;
180         for (int i = 0; i < max(size, rSize); i++) {
181           if (i >= size) {
182             tmp.push_back(((valBi[i] - 48) + rem) % 10 + 48);
183             rem = ((valBi[i] - 48) + rem) / 10;
184           }
185           else if (i >= rSize) {
```

```
186             tmp.push_back(((value[i] - 48) + rem) % 10 + 48);
187             rem = ((value[i] - 48) + rem) / 10;
188           }
189           else {
190             tmp.push_back(((valBi[i] - 48) + (value[i] - 48) + rem) % 10
                  + 48);
191             rem = ((valBi[i] - 48) + (value[i] - 48) + rem) / 10;
192           }
193         }
194         if (rem) {
195           tmp.push_back(rem);
196         }
197         if (!sign) {
198           tmp.push_back('-');
199         }
200         reverse(tmp.begin(), tmp.end());
201         return BigInt(tmp);
202       }
203       else if (!sign) {
204         BigInt tmp(*this);
205         tmp.setSign(true);
206         return rBi - tmp;
207       }
208       else {
209         BigInt tmp(rBi);
210         tmp.setSign(true);
211         return *this - tmp;
212       }
213     }
214     void operator+=(const BigInt&rBi) {
215       BigInt tmp = *this + rBi;
216       *this = tmp;
217     }
218     void operator-=(const BigInt&rBi) {
219       BigInt tmp = *this - rBi;
220       *this = tmp;
221     }
222     friend ostream & operator<< (ostream &out, const BigInt &bi);
223 };
224
225 ostream & operator << (ostream &out, const BigInt &bi)
226 {
227     char* val = bi.getValue();
228     if (!bi.getSign()) {
229       out << "-";
230     }
231     for (int i = bi.getSize() - 1; i >= 0; i--)
232       out << val[i];
233     out << endl;
234     return out;
235 }
```

A.3 - Unittest1.cpp

```
1 #include "stdafx.h"
2 #include "CppUnitTest.h"
3 #include "../BigInt/BigInt.h"
4 using namespace Microsoft::VisualStudio::CppUnitTestFramework;
```

```
 5
 6 namespace UnitTest1
 7 {
 8   TEST_CLASS(UnitTestEqual)
 9   {
10   public:
11
12     TEST_METHOD(TestMethodPosEqual)
13     {
14       BigInt b1("100");
15       BigInt b2("100");
16       Assert::IsTrue(b1 == b2);
17     }
18
19     TEST_METHOD(TestMethodNegEqual)
20     {
21       BigInt b1("-100");
22       BigInt b2("-100");
23       Assert::IsTrue(b1 == b2);
24     }
25
26     TEST_METHOD(TestMethodNotEqual)
27     {
28       BigInt b1("100");
29       BigInt b2("-100");
30       Assert::IsFalse(b1 == b2);
31     }
32
33   };
34   TEST_CLASS(UnitTestSum)
35   {
36   public:
37
38     TEST_METHOD(TestMethodPosPos)
39     {
40       BigInt b1("340282366920938463463374607431768211456");
41       BigInt b2("147808829414345923316083210206383297601");
42       BigInt b3("488091196335284386779457817638151509057");
43       Assert::IsTrue(b1 + b2 == b3);
44       Assert::IsTrue(b2 + b1 == b3);
45     }
46     TEST_METHOD(TestMethodPosNeg)
47     {
48       BigInt b1("170141183460469231731687303715884105728");
49       BigInt b2("-49269609804781974438694403402127765867");
50       BigInt b3("120871573655687257292992900313756339861");
51       Assert::IsTrue(b1 + b2 == b3);
52       Assert::IsTrue(b2 + b1 == b3);
53     }
54     TEST_METHOD(TestMethodPosNegNeg)
55     {
56       BigInt b1("
           160693804425899027554196209234116260252220299378279283530137
           6");
57       BigInt b2("
           -265613988875874769338781322035779626829233452653394495974574961737
           ");
```

```
58          BigInt b3("
                -26561398887587476933878132203577962522229540839440422043261286939
                ");
59          Assert::IsTrue(b1 + b2 == b3);
60          Assert::IsTrue(b2 + b1 == b3);
61      }
62      TEST_METHOD(TestMethodNegNeg)
63      {
64          BigInt b1("-126765060022822940149670320537 6");
65          BigInt b2("-129934811447123020117172145698449");
66          BigInt b3("-131202462047351249518668848903825");
67          Assert::IsTrue(b1 + b2 == b3);
68          Assert::IsTrue(b2 + b1 == b3);
69      }
70
71  };
72  TEST_CLASS(UnitTestSubs)
73  {
74  public:
75
76      TEST_METHOD(TestMethodPosPos)
77      {
78          BigInt b1("340282366920938463463374607431768211456");
79          BigInt b2("147808829414345923316083210206383297601");
80          BigInt b3("192473537506592540147291397225384913855");
81          Assert::IsTrue(b1 - b2 == b3);
82          Assert::IsTrue(b2 - b1 == -b3);
83      }
84      TEST_METHOD(TestMethodPosNeg)
85      {
86          BigInt b1("170141183460469231731687303715884105728");
87          BigInt b2("-49269609804781974438694403402127765867");
88          BigInt b3("219410793265251206170381707118011871595");
89          Assert::IsTrue(b1 - b2 == b3);
90          Assert::IsTrue(b2 - b1 == -b3);
91      }
92      TEST_METHOD(TestMethodPosNegNeg)
93      {
94          BigInt b1("
                160693804425899027554196209234116260252220299378279283530 1376"
                );
95          BigInt b2("
                -26561398887587476933878132203577962682923345265339449597457496173
                ");
96          BigInt b3("
                26561398887587476933878132203577962843617149691238477151653705408 2
                ");
97          Assert::IsTrue(b1 - b2 == b3);
98          Assert::IsTrue(b2 - b1 == -b3);
99      }
100     TEST_METHOD(TestMethodNegNeg)
101     {
102         BigInt b1("-126765060022822940149670320537 6");
103         BigInt b2("-129934811447123020117172145698449");
104         BigInt b3("128667160846894790715675442493073");
105         Assert::IsTrue(b1 - b2 == b3);
106         Assert::IsTrue(b2 - b1 == -b3);
```

```
107             }
108
109         };
110     }
```