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

Отчёт к лабораторной работе №5

по дисциплине

«Языки программирования»

Работу выполнила

Студент группы СКБ222 _____ М. Х. Халимов
подпись, дата

Работу проверил _____ С. А. Булгаков
подпись, дата

Москва 2022

Содержание

Постановка задачи

Необходимо разработать класс `bigint` для выполнения операций с целыми числами, содержащими произвольное количество знаков. Класс должен соответствовать следующему интерфейсу:

```
class bigint {
    char   *data;
    size_t  size;
    bool    neg ;

  public:
    bigint();
    bigint(const bigint&);
   ~bigint();

    explicit bigint(long);
    explicit bigint(unsigned long);
    explicit bigint(const char*);

    void print() const;

    static bigint add(const bigint& left, const bigint& right);
    static bigint sub(const bigint& left, const bigint& right);
    static bigint mul(const bigint& left, const bigint& right);
    static bigint div(const bigint& left, const bigint& right);
    static bigint div(const bigint& left, const bigint& right, bigint& rest);
};
```

Интерфейс класса необходимо разместить в заголовочном файле `bigint.h`. Заголовочный файл должен содержать защиту от повторного включения.

Реализацию методов класса необходимо вынести в файл исходного кода `bigint.cpp`.

В основной функции, размещенной в файле `main.cpp`, необходимо продемонстрировать применение разработанного класса и его методов.

1. Описание класса *bigint*

Объявляется класс *bigint*, который является совокупностью полей *char *data, size_t size, bool neg* и методов *bigint, bigint(const bigint&), ~bigint, bigint(long), bigint(unsigned long), bigint(const char*), print, add, sub, mul, div, div.*

Класс отвечает за создание длинных чисел, состоящих из произвольного количество символов. Числа могут иметь как знак *“+”*, так и знак *“-”*.

2. Описание методов класса

2.1 Конструктор объекта класса *bigint*

Метод используется для создания объекта класса *bigint*. Метод создает *“пустой”* объект класса *bigint*. Блок-схему метода можно рассмотреть в *Приложение А.1*.

2.2 Конструктор копирования

Метод используется для копирования объекта класса *bigint*. Блок-схему метода можно рассмотреть в *Приложение А.2*.

2.3 Деструктор

Метод используется для уничтожения объекта класса *bigint.* Блок-схему метода можно рассмотреть в *Приложение А.3*.

2.4 Конструктор объекта класса *bigint* при помощи переменной типа *long*

Метод используется для создания объекта класса *bigint.* На вход метод получает переменную типа *long*. В результате создается объект класса *bigint*. Блок-схему метода можно рассмотреть в *Приложение А.4.*

2.5 Конструктор объекта класса *bigint* при помощи переменной типа *unsigned long*

Метод используется для создания объекта класса *bigint.* На вход метод получает переменную типа *unsigned long*. В результате создается объект класса *bigint*. Блок-схему метода можно рассмотреть в *Приложение А.5*.

2.6 Конструктор объекта класса bigint при помощи массива типа *char*

Метод используется для создания объекта класса *bigint*. На вход метод получает массив типа *char*. В результате создается объект класса *bigint*. Блок-схему метода можно рассмотреть в *Приложение А.6*.

2.7 Метод вывода объекта класса *bigint*

Метод используется для вывода в консоль числа, созданного с помощью одного из конструкторов класса *bigint*. Блок-схему метода можно рассмотреть в *Приложение А.7*.

2.8. Метод нахождения суммы двух объектов класса *bigint*

На вход метод получает два длинных числа. Если знак обоих чисел одинаковый, то происходит сложение по модулю и присвоение их знака любого из чисел. Если знаки различны, вычисляется разница модулей, причем если модуль “левого” больше, чем модуль “правого”, то присваивается знак “+”, иначе знак “-”. В результате работы метод возвращает новый объект класса bigint. Блок-схему метода можно рассмотреть в *Приложение А.8*.

2.9. Метод нахождения разности двух объектов класса *bigint*

На вход метод получает два длинных числа. Создается новый объект класса *bigint*, который копирует “правое” число, затем знак нового числа меняется на противоположный, после чего вызывается метод сложения “левого” числа и новообразованного числа. В результате работы метод возвращает новый объект класса *bigint*. Блок-схему метода можно рассмотреть в *Приложение А.9*.

2.10. Метод нахождение произведения двух объектов класса *bigint*

На вход метод получает два длинных числа. Если знаки обоих чисел совпадают, то результат будет положительным, иначе – отрицательным. Умножение производится путем перемножения разрядов первого числа на разряды второго. В результате работы метод возвращает новый объект класса *bigint*. Блок-схему метода можно рассмотреть в *Приложение А.10*.

2.11. Метод нахождение частного двух объектов класса *bigint*

На вход метод получает два длинных числа. В общих чертах алгоритм работает так же, как и алгоритм для умножения. Серьезным различием является наличие проверки, является ли итоговое число нулем. В результате работы метод возвращает

новый объект класса *bigint*. Блок-схему метода можно рассмотреть в *Приложение А.11*.

2.12. Метод нахождение остатка от деления двух объектов класса *bigint*

На вход метод получается два длинных числа. В результате работы метод возвращает новый объект класса *bigint*. Блок-схему метода можно рассмотреть в *Приложение А.12*.

3. Функция main

Функция *main* включает в себя создание объектов класса *bigint* и вызов методов этого же класса. В результате работы функции в консоль выводится сумма, разность, произведение, частное и остаток, получаемый из заданных чисел. Блок-схему метода можно рассмотреть в *Приложение А.13*.

4. Результаты тестирования программы

Листинг

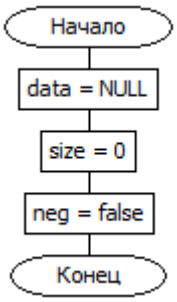
```
1001
1002
-700

---ADD---

2003
302
301

2003
302
301

---SUB---

-1
1702
1701

1
-1702
-1701
```

```
---MUL---

1003002
-701400
-700700

1003002
-701400
-700700

---DIV---

0
-1
0

1
0
-1

---DIV_R---

1001
302
-700

1
-700
301
```

Приложение А

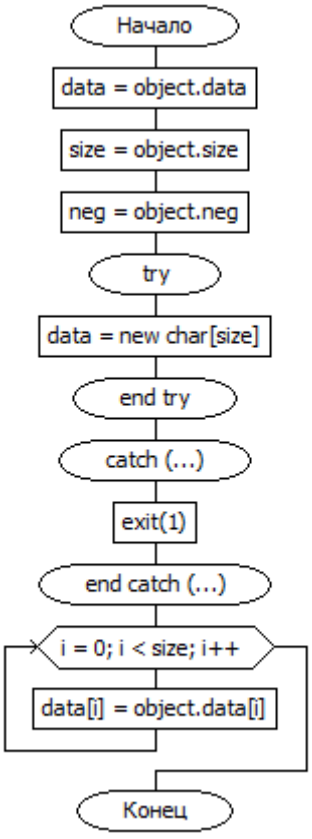
А.1.

Блок-схема конструктора объекта класса *bigint*



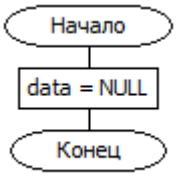
А.2.

Блок-схема конструктора копирования



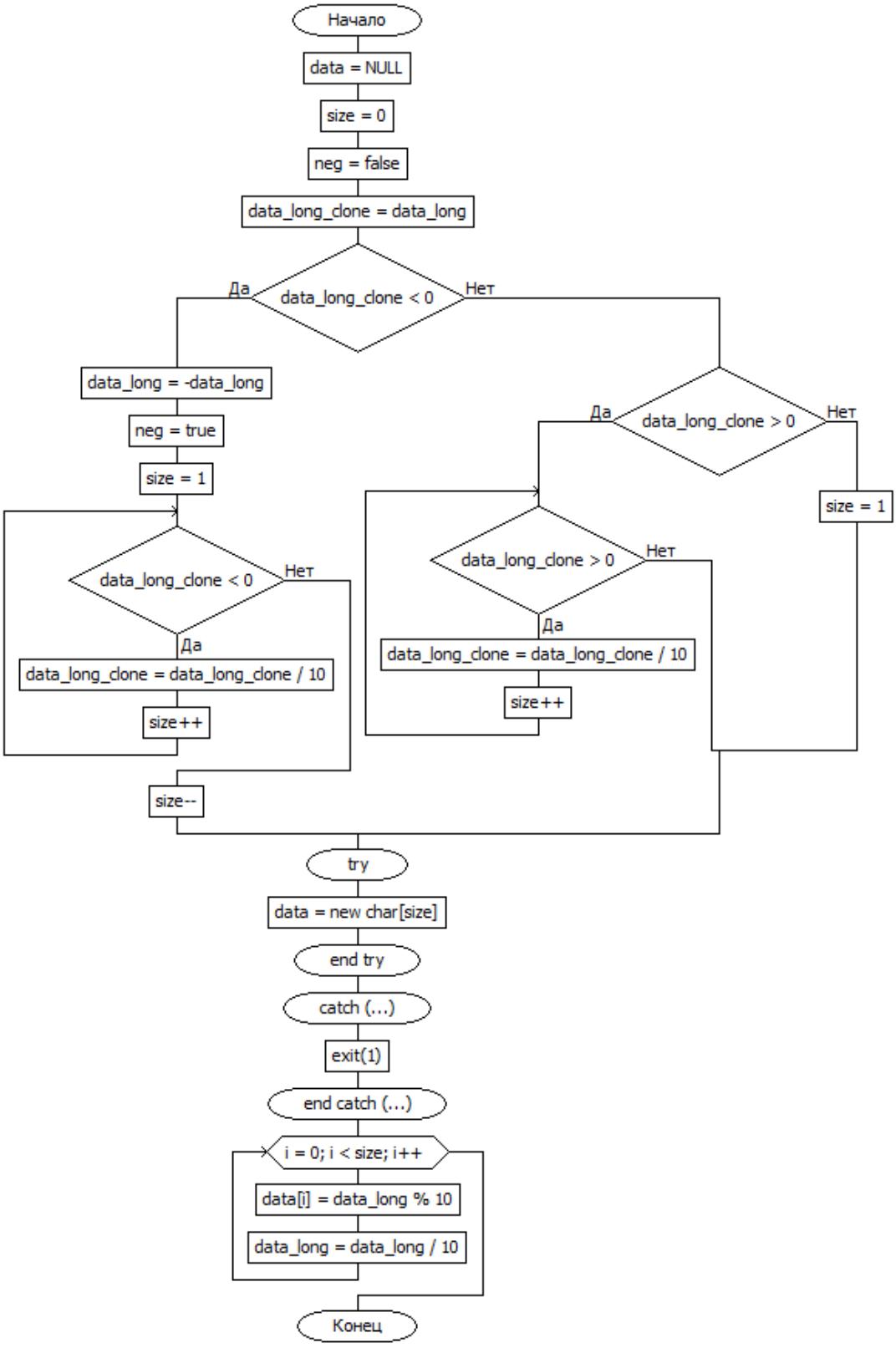
А.3.

Блок-схема деструктора



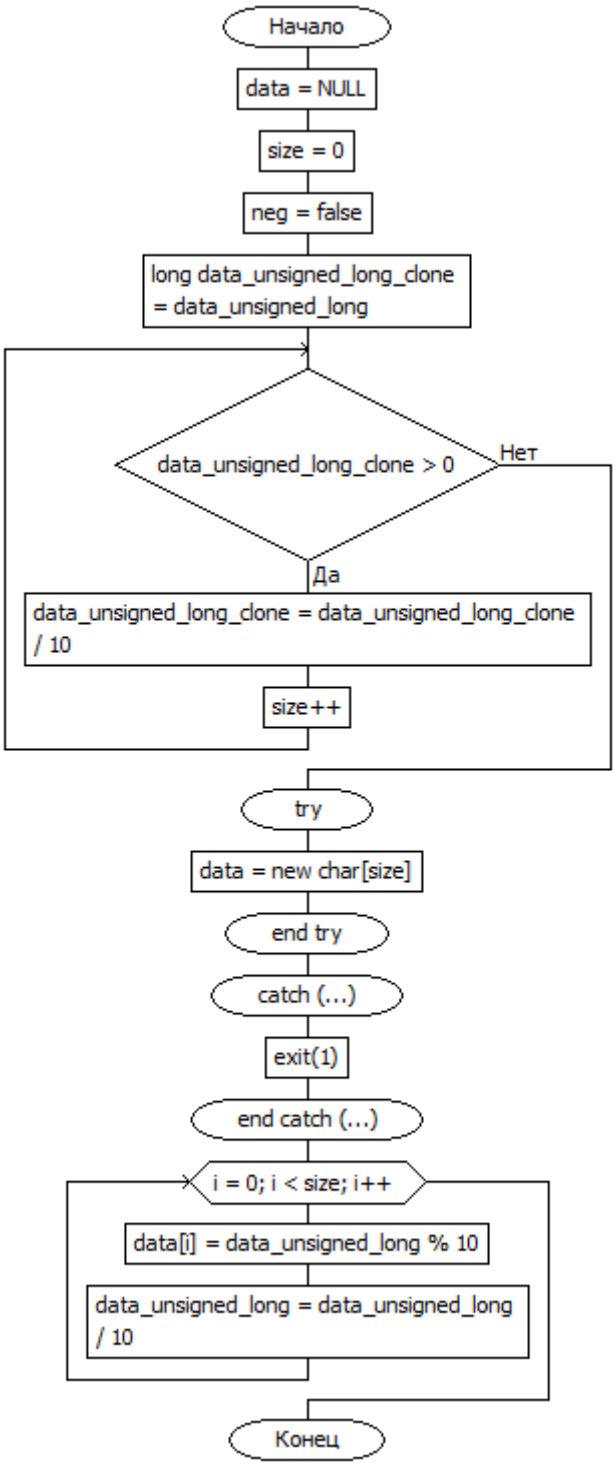
A.4.

Блок-схема конструктора объекта класса *bigint* с помощью переменной типа *long*



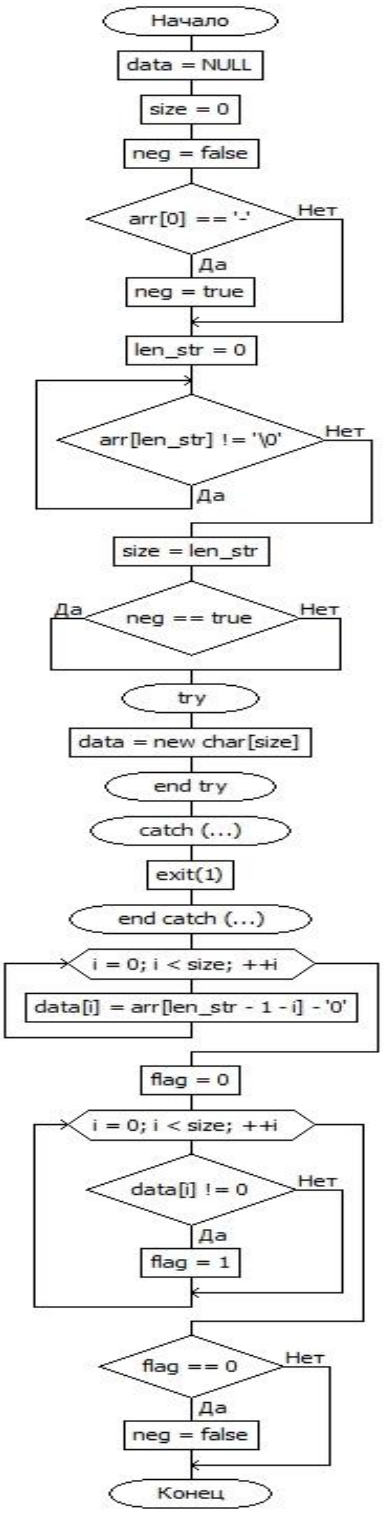
A.5.

Блок-схема конструктора объекта класса *bigint* с помощью переменной типа *unsigned long*



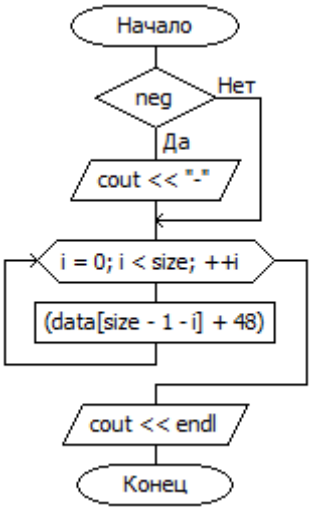
А.6.

Блок-схема конструктора объекта класса *bigint* с помощью массива типа *char*



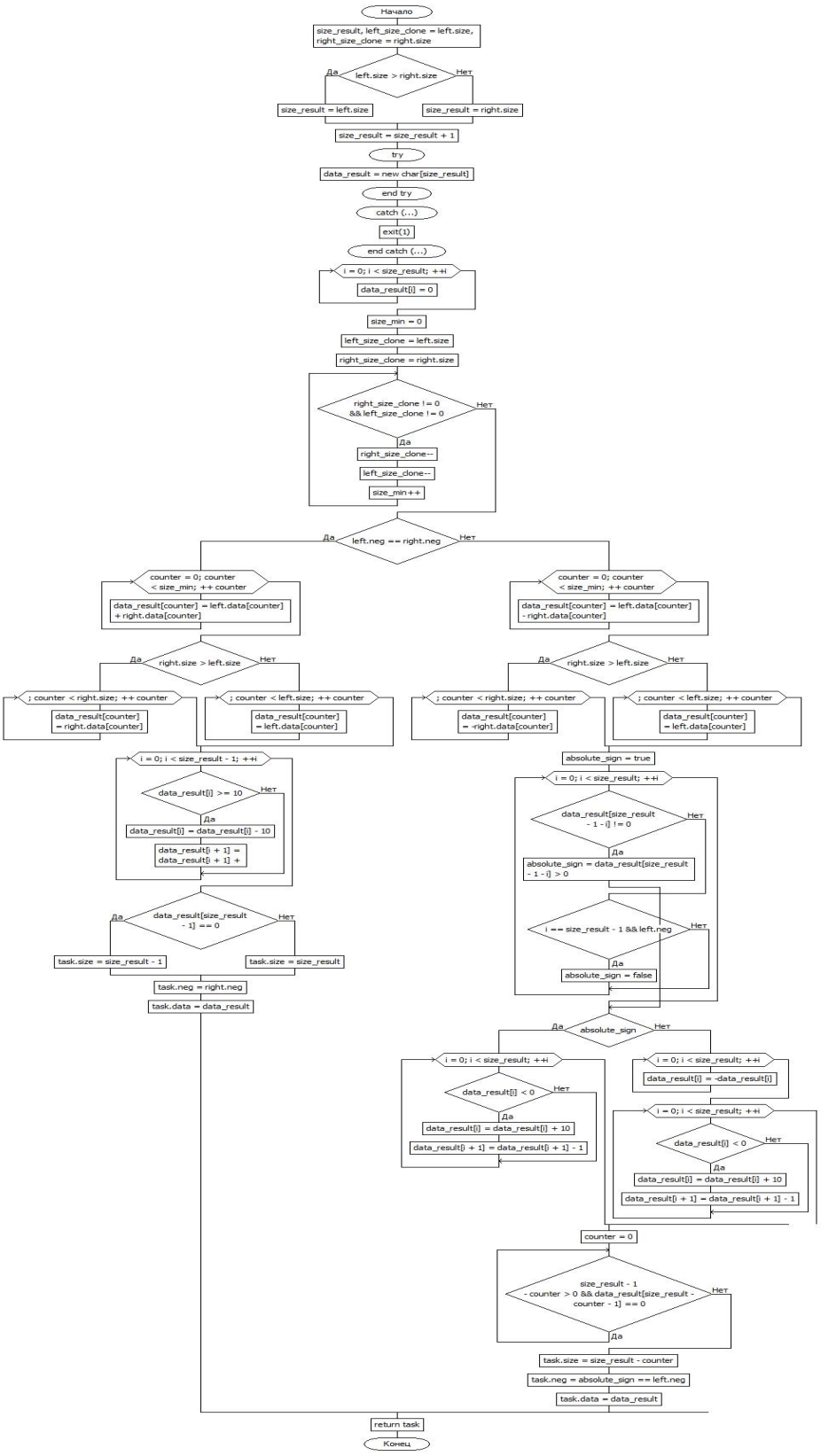
А.7.

Блок-схема метода *print*



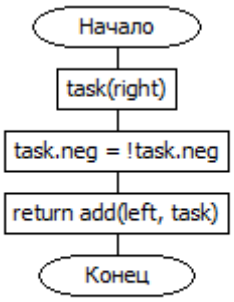
А.8.

Блок-схема метода, реализующего сложение двух длинных чисел



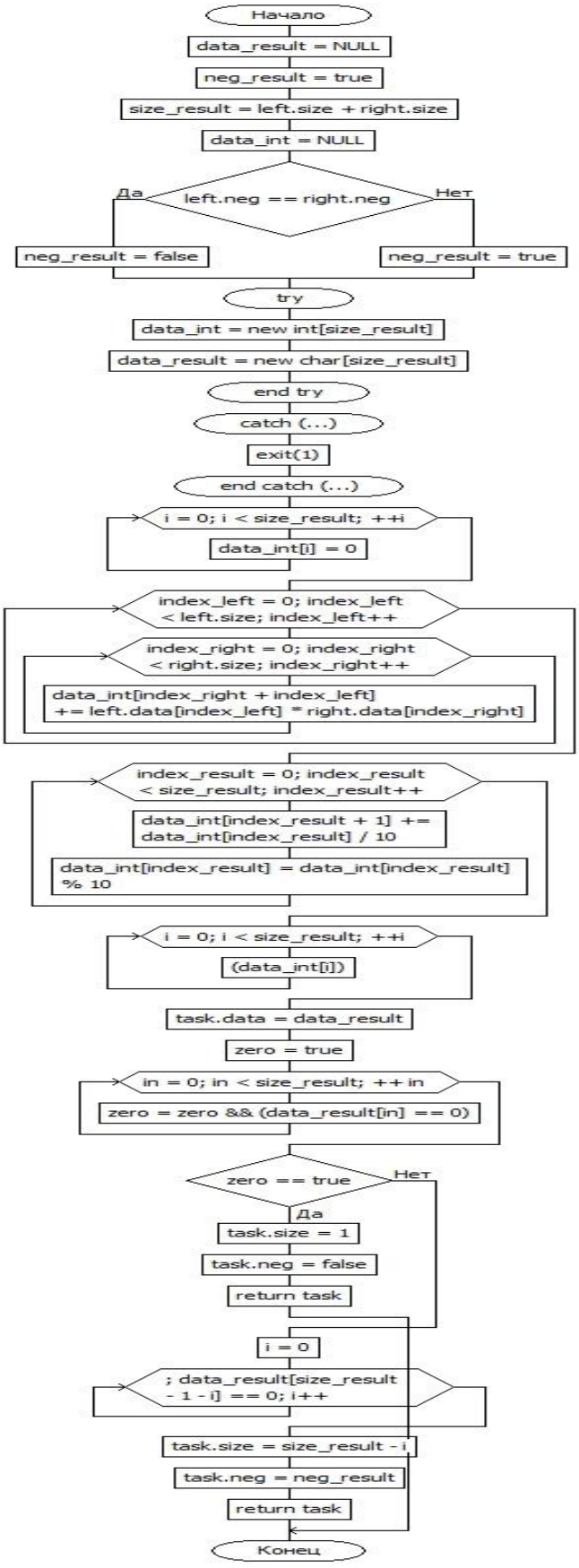
А.9.

Блок-схема метода, реализующего вычитание двух длинных чисел



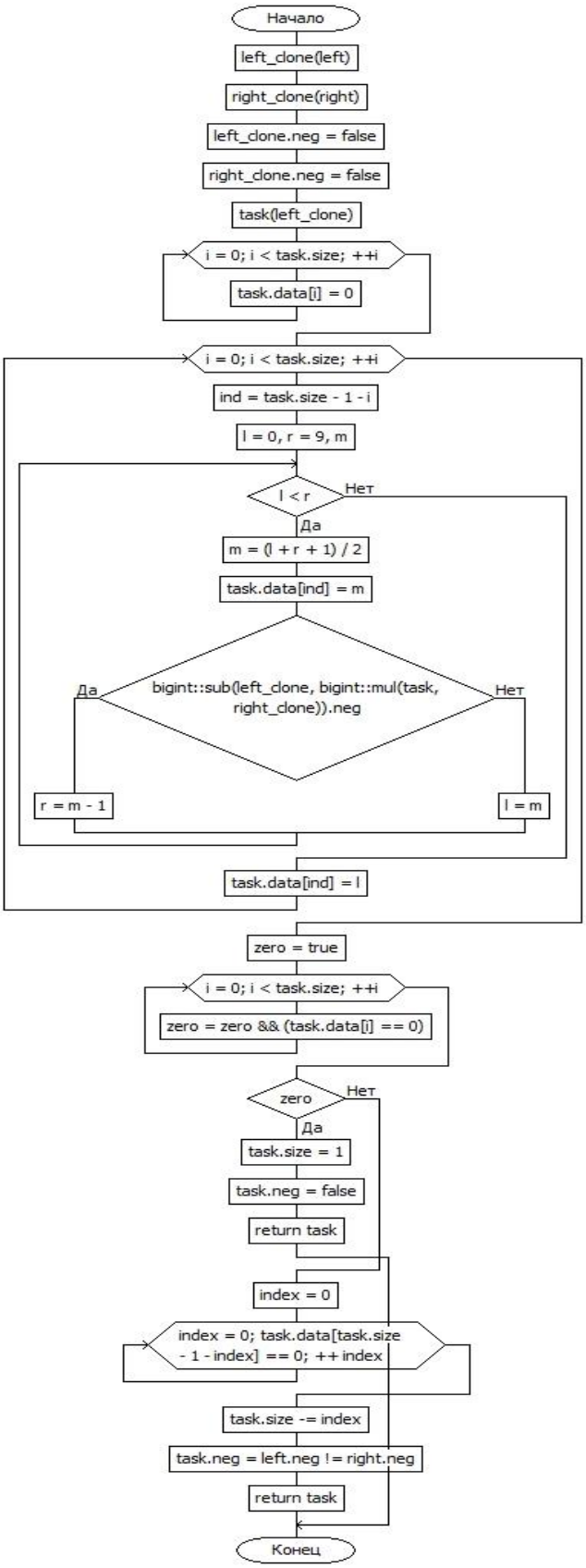
А.10.

Блок-схема метода, реализующего умножение двух длинных чисел



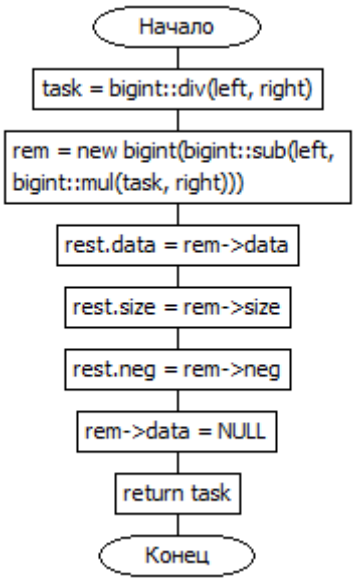
А.11.

Блок-схема метода, реализующего деление двух длинных чисел



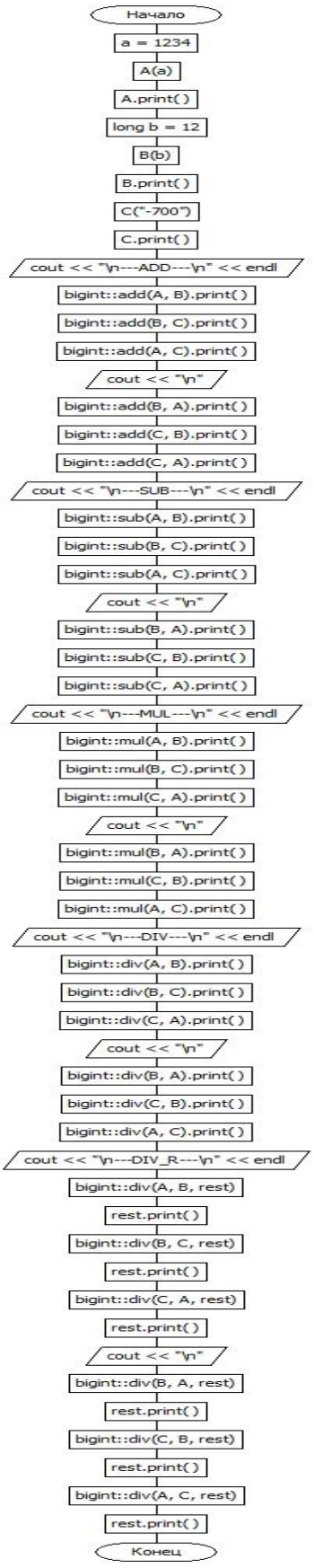
А.12.

Блок-схема метода, реализующего нахождение остатка от деления двух длинных чисел



A.13

Блок-схема функции *main*



Приложение Б

Исходный код программы

```
#include <iostream>
#include "bigint.h"
#include <cstdlib>

using std::cerr;
using std::cout;
using std::endl;
using std::exit;

bigint::bigint()
{
     data = NULL;
     size = 0;
     neg = false;
}

bigint::bigint(const bigint& object)
{
     data = object.data;
     size = object.size;
     neg = object.neg;

     try
     {
          data = new char [size];
     }
     catch(...)
     {
          cerr << "Error AM" << endl;
          exit(1);
     }

     for(size_t i = 0; i < size; i++)
          data[i] = object.data[i];
}

bigint::~bigint()
{
     delete[] data;
     data = NULL;
}

bigint::bigint(long data_long)
{
     data = NULL;
     size = 0;
     neg = false;
```

```
     long data_long_clone = data_long;


     if( data_long_clone < 0)
     {
          data_long = -data_long;
          neg = true;
          size = 1;
          while ( data_long_clone < 0 )
          {
               data_long_clone = data_long_clone / 10;
               size++;
          }
          size--;
     }
     else if (data_long_clone > 0)
     {
          while (data_long_clone > 0)
          {
               data_long_clone = data_long_clone / 10;
               size++;
          }
     }
     else
     {
          size = 1;
     }

     try
     {
          data = new char [size];
     }
     catch(...)
     {
          cerr << "Error AM" << endl;
          exit(1);
     }

     for(size_t i = 0; i < size; i++)
     {
          data[i] = data_long % 10;
          data_long = data_long / 10;
     }
}

bigint::bigint(unsigned long data_unsigned_long)
{
     data = NULL;
     size = 0;
     neg = false;

     unsigned long data_unsigned_long_clone = data_unsigned_long;
```

```
	while (data_unsigned_long_clone > 0)
	{
		data_unsigned_long_clone = data_unsigned_long_clone / 10;
		size++;
	}

	try
	{
		data = new char [size];
	}
	catch(...)
	{
		cerr << "Error AM" << endl;
		exit(1);
	}

	for(size_t i = 0; i < size; i++)
	{
		data[i] = data_unsigned_long % 10;
		data_unsigned_long = data_unsigned_long / 10;
	}
}

bigint::bigint(const char* arr)
{
	data = NULL;
	size = 0;
	neg = false;

	if(arr[0] == '-')
	{
		neg = true;
	}

	size_t len_str = 0;

	while (arr[len_str] != '\0')
	{
		++len_str;
	}

	size = len_str;

	if(neg == true)
	{
		--size;
	}

	try
	{
		data = new char[size];
```

```
    }
    catch (...)
    {
        cerr << "Error AM" << endl;
        exit(1);
    }



    for(size_t i = 0; i < size; ++i)
    {
        data[i] = arr[len_str - 1 - i] - '0';
    }

    int flag = 0;

    for(size_t i = 0; i < size; ++i)
    {
        if(data[i] != 0)
        {
            flag = 1;
        }
    }

    if (flag == 0)
    {
        neg = false;
    }
}

void bigint::print() const
{
    if (neg)
        cout << "-";
    for(size_t i = 0; i < size; ++i)
    {
        cout << static_cast<char>(data[size-1-i]+48);
    }
    cout << endl;
}

bigint bigint::add(const bigint& left, const bigint& right)
{
    char* data_result;
    //bool neg_result;
    size_t size_result, left_size_clone = left.size, right_size_clone
= right.size;

    bigint task;

    if (left.size > right.size)
    {
```

```
          size_result = left.size;
     }
     else
     {
          size_result = right.size;
     }

     size_result = size_result + 1; //Потому что может быть увеличение
разряда (это я для себя, потому что читая продолжив писать через 2
дня, я забыл что писал)

     try
     {
          data_result = new char[size_result];
     }
     catch (...)
     {
          cerr << "Error AM" << endl;
          exit(1);
     }

     for(size_t i = 0; i < size_result; ++i)
     {
          data_result[i] = 0;
     }

     size_t size_min = 0;
     left_size_clone = left.size;
     right_size_clone = right.size;

     while(right_size_clone != 0 && left_size_clone != 0)
     {
          right_size_clone--;
          left_size_clone--;
          size_min++;
     }

     if(left.neg == right.neg)
     {
          size_t counter;

          for(counter = 0; counter < size_min; ++counter)
          {
               data_result[counter] = left.data[counter] +
right.data[counter];
          }

          if(right.size > left.size)
          {
               for(; counter < right.size; ++counter)
               {
                    data_result[counter] = right.data[counter];
```

```
            }
        }
        else
        {
            for(; counter < left.size; ++counter)
            {
                data_result[counter] = left.data[counter];
            }
        }

        for(size_t i = 0; i < size_result - 1; ++i)
        {
            if(data_result[i] >= 10)
            {
                data_result[i] = data_result[i] - 10;
                data_result[i + 1] = data_result[i + 1] + 1;
            }
        }

        if(data_result[size_result - 1] == 0)
        {
            task.size = size_result - 1;
        }
        else
        {
            task.size = size_result;
        }
        task.neg = right.neg;
        task.data = data_result;
    }
    else
    {
        size_t counter;

        for(counter = 0; counter < size_min; ++counter)
        {
            data_result[counter] = left.data[counter] -
right.data[counter];
        }

        if(right.size > left.size)
        {
            for(; counter < right.size; ++counter)
            {
                data_result[counter] = -right.data[counter];
            }
        }
        else
        {
            for(; counter < left.size; ++counter)
            {
                data_result[counter] = left.data[counter];
```

```
            }
        }

        bool absolute_sign = true;

        for(size_t i = 0; i < size_result; ++i)
        {
            if(data_result[size_result - 1 - i] != 0)
            {
                absolute_sign = data_result[size_result - 1 - i] >
0;
                break;
            }
            if (i == size_result - 1 && left.neg)
            {
                absolute_sign = false;
            }
        }

        if(absolute_sign)
        {
            for (size_t i = 0; i < size_result; ++i)
            {
                if(data_result[i] < 0)
                {
                    data_result[i] = data_result[i] + 10;
                    data_result[i + 1] = data_result[i + 1] - 1;
                }
            }
        }
        else
        {
            for(size_t i = 0; i < size_result; ++i)
            {
                data_result[i] = -data_result[i];
            }
            for(size_t i = 0; i < size_result; ++i)
            {
                if(data_result[i] < 0)
                {
                    data_result[i] = data_result[i] + 10;
                    data_result[i + 1] = data_result[i + 1] - 1;
                }
            }
        }

        counter = 0;

        while(size_result - 1 - counter > 0 &&
data_result[size_result - counter - 1] == 0)
        {
            ++counter;
```

```
            }

            task.size = size_result - counter;
            task.neg = absolute_sign == left.neg;
            task.data = data_result;

        }

        return task;
}

bigint bigint::sub(const bigint& left, const bigint& right)
{
        bigint task(right);
        task.neg = !task.neg;

        return add(left, task);
}

bigint bigint::mul(const bigint& left, const bigint& right)
{
        char* data_result = NULL;
        bool neg_result = true;
        size_t size_result = left.size + right.size;
        int* data_int = NULL;
        bigint task;

        if (left.neg == right.neg)
        {
            neg_result = false;
        }
        else
        {
            neg_result = true;
        }

        try
        {
            data_int = new int[size_result];
            data_result = new char[size_result];
        }
        catch(...)
        {
            cerr << "Error AM" << endl;
            exit(1);
        }

        for(size_t i = 0; i < size_result; ++i)
        {
            data_int[i] = 0;
        }
```

```
    for(size_t index_left = 0; index_left < left.size; index_left++)
    {
        for(size_t index_right = 0; index_right < right.size;
index_right++)
        {
            data_int[index_right + index_left] +=
left.data[index_left] * right.data[index_right];
        }
    }

    for(size_t index_result = 0; index_result < size_result;
index_result++)
    {
        data_int[index_result + 1] += data_int[index_result] / 10;
        data_int[index_result] = data_int[index_result] % 10;
    }

    /*while(data_result[size_result] == 0)
    {
        size_result--;
    }*/

    for(size_t i = 0; i < size_result; ++i )
    {
        data_result[i] = static_cast<char>(data_int[i]);
    }

    delete[] data_int;

    task.data = data_result;

    bool zero = true;

    for(size_t in = 0; in < size_result; ++in)
    {
        zero = zero && (data_result[in] == 0);
    }

    if(zero == true)
    {
        task.size = 1;
        task.neg = false;
        return task;
    }

    size_t i = 0;

    for(; data_result[size_result - 1 - i] == 0; i++);


    task.size = size_result - i;
    task.neg = neg_result;
```

```
     return task;
}

bigint bigint::div(const bigint& left, const bigint& right)
{

     bigint left_clone(left);
     bigint right_clone(right);

     left_clone.neg = false;
     right_clone.neg = false;

     bigint task(left_clone);

     for (size_t i = 0; i < task.size; ++i)
     {
          task.data[i] = 0;
     }
     for(size_t i = 0; i < task.size; ++i)
     {
          size_t ind = task.size - 1 - i;
          char l = 0, r = 9, m;

          while(l < r)
          {
               m = (l + r + 1) / 2;
               task.data[ind] = m;
               if (bigint::sub(left_clone, bigint::mul(task,
right_clone)).neg)
               {
                    r = m - 1;
               }
               else
               {
                    l = m;
               }
          }

          task.data[ind] = l;
     }

     bool zero = true;

     for(size_t i = 0; i < task.size; ++i)
     {
          zero = zero && (task.data[i] == 0);
     }

     if(zero)
     {
          task.size = 1;
```

```
            task.neg = false;
            return task;
        }

        size_t index = 0;

        for(index = 0; task.data[task.size - 1 - index] == 0; ++index);

        task.size -= index;

        task.neg = left.neg != right.neg;

        return task;
}

bigint bigint::div(const bigint& left, const bigint& right, bigint&
rest)
{
        bigint task = bigint::div(left, right);
        bigint* rem = new bigint(bigint::sub(left, bigint::mul(task,
right)));

        rest.data = rem -> data;
        rest.size = rem -> size;
        rest.neg = rem -> neg;

        rem -> data = NULL;

        delete rem;

        return task;
}
```

Приложение В

UML-схема класса *bigint*