МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования «Санкт-Петербургский политехнический
университет Петра Великого»

Институт компьютерных наук и кибербезопасности

Высшая школа технологий искусственного интеллекта

Направление: 02.03.01 «Математика и компьютерные науки»

Отчет по курсовой работе по дисциплине «Дискретная математика»

Калькулятор «большой» конечной арифметики. Вариант 16

Студент,
группы 5130201/20001 _____ Якунин Д. Д.

Преподаватель _____ Востров А. В.

«_____»_____ 2023г.

Санкт-Петербург, 2023

Содержание

Введение **3**

1 Математическое описание **4**
1.1 Алгебраические структуры 4
1.2 Свойства операций 4
1.3 Таблицы операций малой конечной арифметики 4

2 Особенности реализации **6**
2.1 Класс myNum 6
2.1.1 Поля класса 6
2.1.2 Конструктор класса 6
2.1.3 Оператор сложения «+» 6
2.1.4 Оператор вычитания «-» 8
2.1.5 Оператор умножения «*» 9
2.1.6 Оператор деления «/» 9
2.1.7 Метод Pow 10
2.1.8 Операторы != и == 11
2.1.9 Метод absCmp 11
2.1.10 Метод abs 11
2.2 Функции вне классов 12
2.2.1 Функция index 12
2.2.2 Функция stringInput 12

3 Результаты работы программы **14**

Заключение **17**

Список литературы **18**

Введение

Курсовая работа представляет собой реализацию калькулятора «большой» конечной арифметики $< Z_i; +, * >$ (8 разрядов) для четырех действий (+, –, *, ÷) на основе «малой» конечной арифметики, где задано правило «+1» и выполняются свойства коммутативности (+,*), ассоциативности (+, *), дистрибутивности * относительно +, заданы аддитивная единица «a» и мультипликативная единица «b», а также выполняется свойство: для $(\forall x)\ x * a = a$. Дополнительно можно реализовать возведение в степень.

Правило «+1» приведено в таблице:

x	a	b	c	d	e	f	g	h
x+1	b	g	h	e	a	d	c	f

1 Математическое описание

1.1 Алгебраические структуры

Коммутативное кольцо с единицей - кольцо $< Z; +, * >$ такое, что:

- Умножение коммутативно $a * b = b * a$
- Существует единица, то есть кольцо - моноид по умножению $\exists 1 \in M\ (a * 1 = 1 * a = a)$

Малая конечная арифметика - конечное коммутативное кольцо с единицей $< Z_i; +, * >$, на котором определены действия вычитания и деления, причем деление определено частично.

Большая конечная арифметика - конечное коммутативное кольцо с единицей $< Z_i^n; +, * >$, на котором определены действия вычитания и деления, причем деление определено с остатком. В нащем случае n = 8 и i = 8: $< Z_8^8; +, * >$

В нашем случае операции $+$, *, - и $\div$ с остатком определяются посредством соответствующих операций малой конечной арифметики и отношений порядка.

1.2 Свойства операций

- Ассоциативность сложения и умножения:

  $\forall x, y, z \in M : x + (y + z) = (x + y) + z; x * (y * z) = (x * y) * z;$

- Коммутативность сложения и умножения:

  $\forall x, y, z \in M : x + y = y + x; x * y = y * x$

- Дистрибутивность сложения относительно умножения:

  $\forall x, y, z \in M : x * (y + z) = (x * y) + (x * z)$

- Существование нейтрального элемента по сложению:

  $\exists a \in M : \forall x \in M : x + a = a + x = x$

- Существование нейтрального элемента по умножению:

  $\exists b \in M : \forall x \in M : x * b = b * x = x$

1.3 Таблицы операций малой конечной арифметики

На Рис. 1. представлены таблицы для операций «$+$» и «*» и соответствующие им таблицы переносов «$+_p$» и «$*_p$».

+	a	b	c	d	e	f	g	h
a	a	b	c	d	e	f	g	h
b	b	g	h	e	a	d	c	f
c	c	h	d	b	g	a	f	e
d	d	e	b	h	f	c	a	g
e	e	a	g	f	d	h	b	c
f	f	d	a	c	h	g	e	b
g	g	c	f	a	b	e	h	d
h	h	f	e	g	c	b	d	a

$+_p$	a	b	c	d	e	f	g	h
a	a	a	a	a	a	a	a	a
b	a	a	a	a	b	a	a	a
c	a	a	a	b	b	b	a	a
d	a	a	b	b	b	b	b	b
e	a	b	b	b	b	b	b	b
f	a	a	b	b	b	b	a	b
g	a	a	a	b	b	a	a	a
h	a	a	a	b	b	b	a	b

*	a	b	c	d	e	f	g	h
a	a	a	a	a	a	a	a	a
b	a	b	c	d	e	f	g	h
c	a	c	b	g	f	e	d	h
d	a	d	g	h	g	d	h	a
e	a	e	f	g	b	c	d	h
f	a	f	e	d	c	b	g	h
g	a	g	d	h	d	g	h	a
h	a	h	h	a	h	h	a	a

$*_p$	a	b	c	d	e	f	g	h
a	a	a	a	a	a	a	a	a
b	a	a	a	a	a	a	a	a
c	a	a	b	g	g	b	a	b
d	a	a	g	h	f	c	b	c
e	a	a	g	f	d	h	b	c
f	a	a	b	c	h	c	b	g
g	a	a	a	b	b	b	a	b
h	a	a	b	c	c	g	b	g

Рис. 1. Таблицы по «+» и «*» с таблицами переносов

Из правила «+1» получим отношение порядка в малой конечной арифметике:

$$a \prec b \prec g \prec c \prec h \prec f \prec d \prec e$$

Примеры вычислений некоторых выражений:

1. $c + c = c + b + g = h + g = h + b + b = f + b = d$
2. $d + g = d + b + b = a$
3. $d * d = d(f + b) = df + d = d(h + b) + d = dh + h = d(c + b) + h = dc + g = g + g = h$
4. $h * h = h(c + b) = h + h = a$
5. $d * g = d(b + b) = d + d = h$

2 Особенности реализации

2.1 Класс myNum

Класс предназначен для хранения данных о числах нашей большой арифметики. Для реализации операций перегружены соответствующие операторы +, *, -, /.

Также для применения отношения порядка используется константный вектор с объектами типа char, идущими в том же порядке, что и в отношении порядка нашей арифметики:

```
const std::vector<char> digits = { 'a', 'b', 'g', 'c', 'h', 'f', 'd', 'e' }
```

2.1.1 Поля класса

- std::string num - строка, хранящая строку, содержащая модуль числа
- bool is_positive - переменная, хранящая знак числа

2.1.2 Конструктор класса

При создании объекта типа myNum вызывается конструктор, записывающий модуль числа в поле num, а знак в поле is_positive.

Реализация приведена в Листинге 1.

Листинг 1. Конструктор класса

```
1  myNum::myNum(std::string num) : myNum() {
2    bool flag = 0;
3    if (num[0] == '-') {
4      is_positive = 0;
5      flag = 1;
6    }
7    for (int i = 0; i < num.size() - flag; i++) {
8      this->num[this->num.size() - 1 - i] = num[num.size() - 1 -
           i];
9    }
10 }
```

2.1.3 Оператор сложения «+»

Вход: два объекта типа myType

Выход: объект типа myType - результат сложения

Метод реализует операцию сложения для двух чисел: сначала идет проверка на знак, после чего либо вызывается соответствующая разность, если числа разных знаков, либо числа складываются по модулю и результату присваоивается соответствующий знак.

Само сложение реализовано сложением соответствующих разрядов каждого числа и переносом с предыдущих разрядов. Делается это путем увеличения разряда первого слагаемого и уменьшения разряда второго слагаемого до тех пор, пока разряд второго слагаемого не станет равен нейтральному по сложению элементу a.

Реализация приведена в Листинге 2.

Листинг 2. Оператор сложения

```
myNum myNum::operator + (const myNum& other) const {
  myNum result;
  if (!this->is_positive and !other.is_positive) result.
     is_positive = 0;
  if (this->is_positive and !other.is_positive) return *this -
     other.abs();
  if (!this->is_positive and other.is_positive) return other -
     this->abs();

  char inShift = 'a';
  char outShift = 'a';
  char tmp = 'a';

  for (int i = num.size() - 1; i > -1; i--) {
    inShift = outShift;
    outShift = 'a';
    result.num[i] = this->num[i];

    while (inShift != 'a') {
      if (result.num[i] == 'e') {
        result.num[i] = 'a';
        outShift = 'b';
        inShift = digits[index(inShift) - 1];
        continue;
      }
      result.num[i] = digits[index(result.num[i]) + 1];
      inShift = digits[index(inShift) - 1];
    }

    tmp = other.num[i];
    while (tmp != 'a') {
      if (result.num[i] == 'e') {
        result.num[i] = 'a';
        outShift = 'b';
        tmp = digits[index(tmp) - 1];
        continue;
      }
      result.num[i] = digits[index(result.num[i]) + 1];
      tmp = digits[index(tmp) - 1];
    }
  }
  if (outShift != 'a') result.num = "overflow";
  return result;
}
```

2.1.4 Оператор вычитания «-»

Вход: два объекта типа myType

Выход: объект типа myType - результат вычитания

Оператор вычитания реализован аналогично оператору сложения, но теперь разряд первого числа будет уменьшаться до тех пор, пока разряд второго не не станет равен a.

Реализация приведена в Листинге 3.

Листинг 3. Оператор вычитания

```
myNum myNum::operator - (const myNum& other) const {
  myNum result;
  if (this->is_positive and !other.is_positive) return *this +
     other.abs();
  if (!this->is_positive and other.is_positive) {
    result = this->abs() + other.abs();
    result.is_positive = 0;
    return result;
  }
  if (!this->is_positive and !other.is_positive) return other.
     abs() - this->abs();

  if (other.absCmp(*this)) {
    result = other - *this;
    result.is_positive = 0;
    return result;
  }

  char inShift = 'a';
  char outShift = 'a';
  char tmp = 'a';

  for (int i = num.size() - 1; i > -1; i--) {
    inShift = outShift;
    outShift = 'a';
    result.num[i] = this->num[i];

    if (inShift != 'a') {
      if (result.num[i] == 'a') {
        result.num[i] = 'e';
        outShift = 'b';
      }
      else {
        result.num[i] = digits[index(result.num[i]) - 1];
      }
    }

    tmp = other.num[i];
    while (tmp != 'a') {
```

```
38          if (result.num[i] == 'a') {
39            result.num[i] = 'e';
40            outShift = 'b';
41            tmp = digits[index(tmp) - 1];
42            continue;
43          }
44          result.num[i] = digits[index(result.num[i]) - 1];
45          tmp = digits[index(tmp) - 1];
46        }
47      }
48      return result;
49  }
```

2.1.5 Оператор умножения «*»

Вход: два объекта типа myType

Выход: объект типа myType - результат умножения

Оператор умножения реализован через оператор сложения. Для умножения первый множитель складывается сам с собой до тех пор, пока счетчик i не станет равен второму множителю. С каждым сложением счетчик увеличивается на нейтральный по умножению элемент арифметики b.

Реализация приведена в Листинге 4.

Листинг 4. Оператор умножения

```
1   myNum myNum::operator * (const myNum& other) const {
2     myNum result;
3     for (myNum i; i != other.abs(); i = i + myNum("b")) {
4       result = result + *this;
5       if (result.num == "overflow") return result;
6     }
7     result.is_positive = (result.is_positive == other.
        is_positive);
8     return result;
9   }
```

2.1.6 Оператор деления «/»

Вход: два объекта типа myType

Выход: объект типа myType - результат деления

Перед выполнением деления идет проверка деления на 0. Если и делимое, и делитель равны нулю, то результатом будет множество всех чисел арифметики [-eeeeeeee; eeeeeeee]. Если же нулю равен только делитель, то результатом будет пустое множество.

Сам оператор деления реализован через операторы сложения и вычитания. Из делимого вычитается делитель до тех пор, пока делимое не станет меньше делителя и положительным одновременно. С каждым таким вычитанием частное увеличивается

на b. Как только вычитать больше будет не нужно, в результат записывается частное, а в остаток - делимое.

Реализация приведена в Листинге 5.

Листинг 5. Оператор деления

```
myNum myNum::operator / (const myNum& other) const {
  myNum result;
  myNum remainder(*this);
  myNum i("b");
  if (other == myNum("a").getNum()) {
    if (this->abs() == other.abs())
    result.num = "[-eeeeeeee; eeeeeeee]";
    else
    result.num = "empty set";
    goto stop;
  }

  if (other.is_positive != this->is_positive) i.is_positive = 0;

  while ((remainder.absCmp(other)) or (remainder.num == other.
     num) or (!remainder.is_positive)) {
    result = result + i;
    if (other.is_positive != remainder.is_positive)
    remainder = remainder + other;
    else
    remainder = remainder - other;
  }
  if (remainder.getNum() != "a")
  result.num = result.getNum() + '(' + remainder.getNum() + ')';
  else result.num = result.getNum();
  stop:
  result.is_positive = 1;
  return result;
}
```

2.1.7 Метод Pow

Вход: два объекта типа myType

Выход: объект типа myType - результат возведения в степень

Метод реализован через операторы умножения и сложения. Результатом будет умножение единицы на первое число до тех пор, пока счетчик не станет равен второму числу.

Реализация приведена в Листинге 6.

Листинг 6. Метод Pow

```
myNum myNum::Pow(const myNum& other) const {
  myNum result("b");
  for (myNum i; i != other.abs(); i = i + myNum("b")) {
```

```
    result = result * (*this);
  }
  if (!other.is_positive) result = myNum("b") / result;
  return result;
}
```

2.1.8 Операторы != и ==

Вход: два объекта типа myType

Выход: объект типа bool - результат сравнения чисел

В операторе != просходит сравнение модулей чисел и их знаков. При полном совпадении возвращается False, иначе - True.

Оператор == возвращает обратный оператору != результат.

Реализация приведена в Листингах 7 и 8.

Листинг 7. Оператор !=

```
bool myNum::operator != (const myNum& other) const {
  return (this->num != other.num or this->is_positive != other
      .is_positive);
}
```

Листинг 8. Оператор ==

```
bool myNum::operator == (const myNum& other) const {
  return !(*this != other);
}
```

2.1.9 Метод absCmp

Вход: два объекта типа myType

Выход: объект типа bool - результат сравнения

В методе поразрядно сравниваются модули двух чисел. Если первое число окажется строго больше второго, то метод вернет True, иначе - False.

Реализация приведена в Листинге 9.

Листинг 9. Метод absCmp

```
bool myNum::absCmp(const myNum& other) const {
  for (int i = 0; i < this->num.size(); i++) {
    if (index(this->num[i]) > index(other.num[i])) return 1;
    if (index(this->num[i]) < index(other.num[i])) return 0;
  }
  return 0;
}
```

2.1.10 Метод abs

Вход: объект типа myType

Выход: объект типа myType
Метод возвращает модуль числа.
Реализация приведена в Листинге 10.

Листинг 10. Метод abs

```
myNum myNum::abs() const {
  myNum result(this->num);
  result.is_positive = 1;
  return result;
}
```

2.2 Функции вне классов

2.2.1 Функция index

Вход: символ типа char
Выход: число типа int
Функция позволяет находить, каким по счету является символ в отношении порядка. Если нужного символа в арифметике нет, то будет выведено соотбщение об этом и функция вернет -1.
Реализация приведена в Листинге 11.

Листинг 11. Функция index

```
int index(char c) {
  auto itr = std::find(digits.begin(), digits.end(), c);

  if (itr != digits.cend()) {
    return std::distance(digits.begin(), itr);
  }
  else {
    std::cout << "\n\nElement not found!\n\n";
    return -1;
  }
}
```

2.2.2 Функция stringInput

Вход: сообщение для вывода
Выход: введенная строка типа string
Вспомогательная функция проверки корректности ввода пользователя. В случае, если пользователем будет введено что-то кроме существующего в нашей арифметике числа, то она попросит его повторить ввод. Реализована через методы cin.peek, cin.clear, cin.ignore.
Реализация приведена в Листинге 12.

Листинг 12. Функция stringInput

```
int index(char c) {
```

```
2       auto itr = std::find(digits.begin(), digits.end(), c);
3
4       if (itr != digits.cend()) {
5         return std::distance(digits.begin(), itr);
6       }
7       else {
8         std::cout << "\n\nElement not found!\n\n";
9         return -1;
10      }
11    }
```

3 Результаты работы программы

После запуска программы пользователю предлагается ввести два числа и операцию между ними. В случае некорректного ввода будет выведено сообщение "Incorrect input!"и пользователю нужно будет повторить ввод. Если же ввод был корректным, то на экран будет выведен ответ и предложение продолжить(Рис.2).

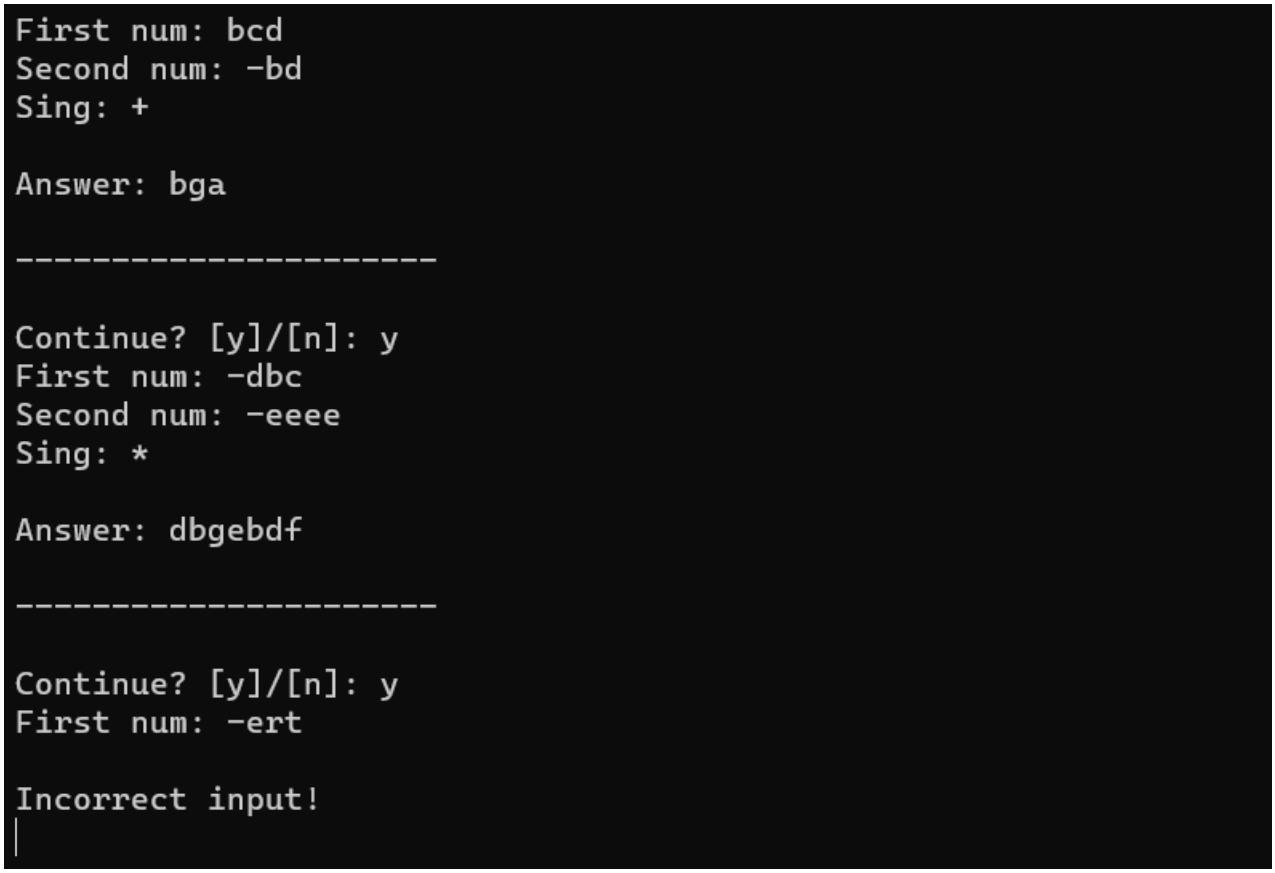

Рис. 2. Некорректный ввод

При делении на 0 будут получены соответствующие результаты (Рис.3).

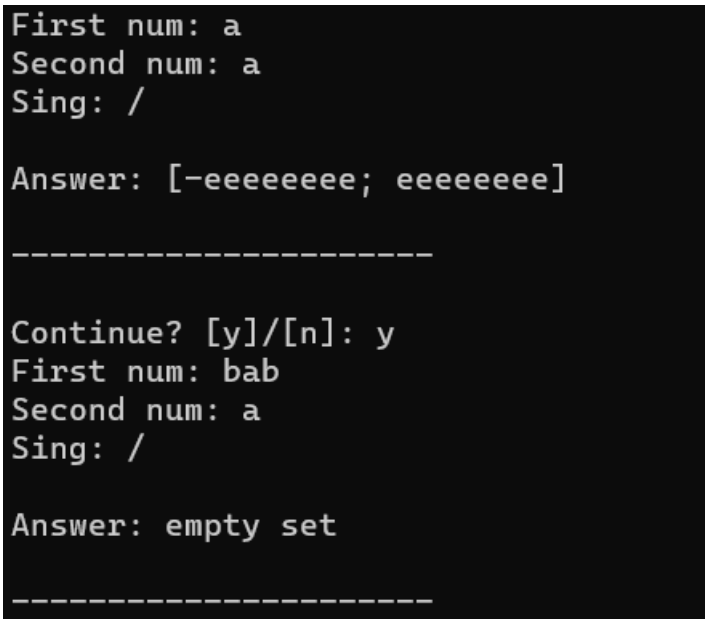

Рис. 3. Деление на 0

Если после деления есть остаток, то он будет записан в скобках после частного, иначе будет выведено только частное (Рис.4).

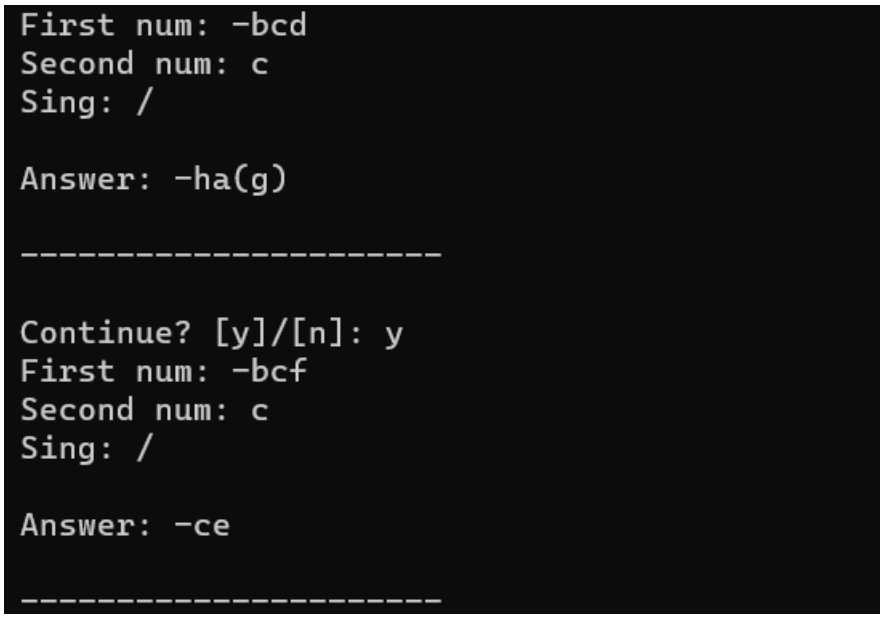

Рис. 4. Деление с остатком и без

Если результат операции выходит за границы арифметики, то будет выведено сообщение об переполнении (Рис.5).

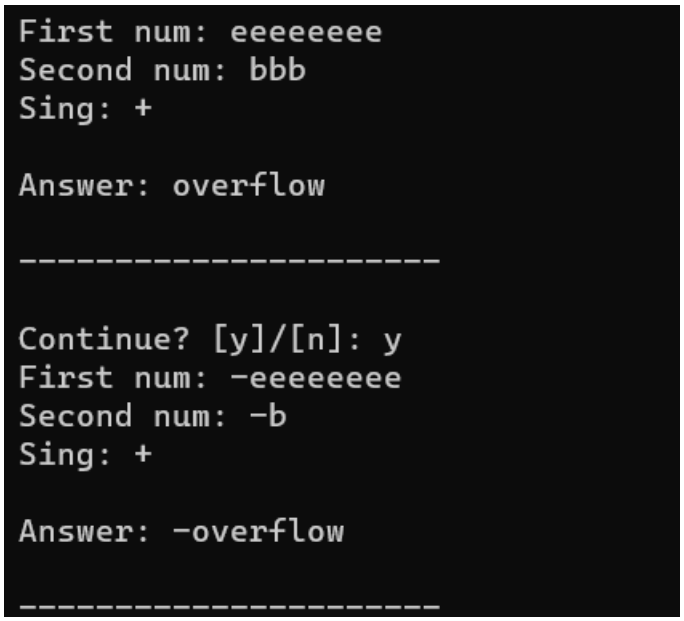

Рис. 5. Переполнение

Во всех операциях учитывается знак чисел (Рис.6).

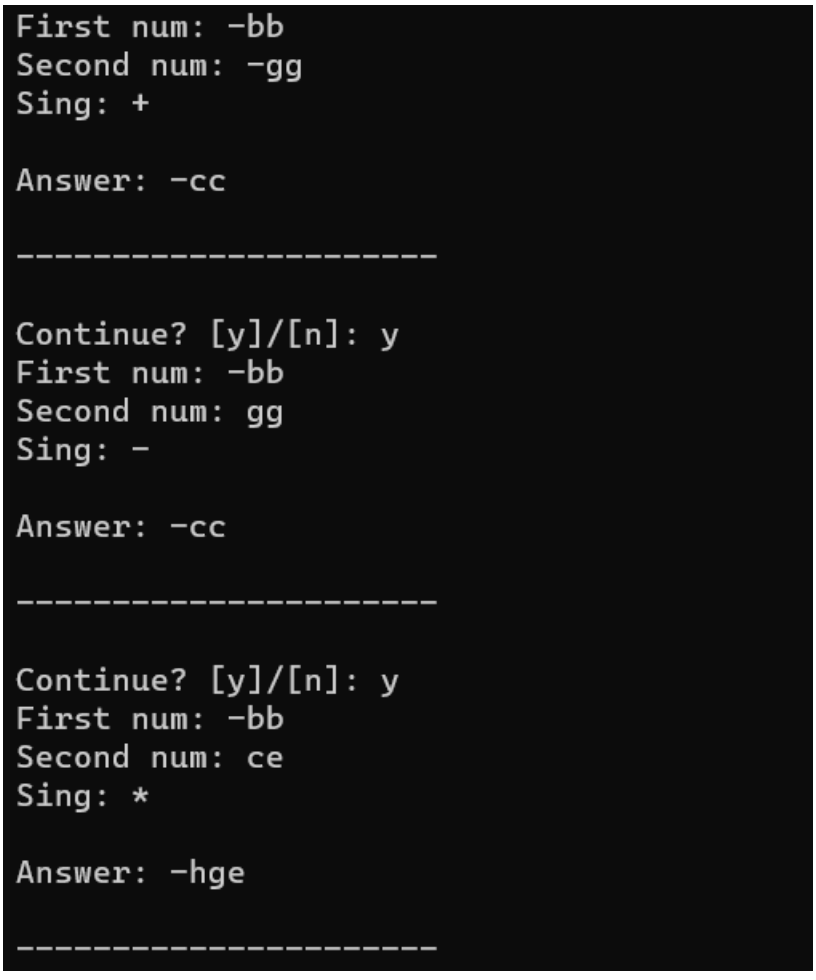

Рис. 6. Операции с отрицательными числами

Заключение

В процессе выполнения курсовой работы был реализован калькулятор «большой» восьмиразрядной арифметики $< Z_8^8, +, * >$ с операциями сложения, умножения, вычитания, деления и возведения в степень на основе «малой» арифметики $< Z_8, +, * >$ с заданным правилом «$+1$».

Числа представлены в памяти собственным типом myNum, в котором для выполнения операций перегружены операторы.

Достоинства программы:

- использование собственного класса и перегруженных операторов упрощает написание программы и увеличивает читаемость кода;
- операции умножения, деления и возведения в степень реализованы при помощи ранее реализованных операций.

Недостатки программы:

- операции умножения и деления, реализованные через сложение и умножение, страдают из-за этого в производительности
- операции реализованы через циклы, а числа арифметики внутри класса хранятся в виде строк типа string, что негативно влияет на производительность.

Масштабирование: На основе реализованного класса можно продолжать дописывать новые методы, например, НОК и НОД.

Список литературы

[1] Новиков, Ф.А. ДИСКРЕТНАЯ МАТЕМАТИКА ДЛЯ ПРОГРАММИСТОВ / 3-е издание. - Питер: ПитерПресс, 2009, (дата обращения - 14.11.2023)

[2] Оформление исходного кода программ в LaTeX - https://mydebianblog.blogspot.com/2012/12/latex.html?m=1, (дата обращения - 14.11.2023)

[3] Секция «Телематика» - https://tema.spbstu.ru/dismath/, (дата обращения - 14.11.2023)

[4] Поиск индекса заданного элемента в векторе на C++ - https://www.geeksforgeeks.org/how-to-find-index-of-a-given-element-in-a-vector-in-cpp/, (дата обращения - 14.11.2023)

[5] Перегрузка операторов в C++ - https://habr.com/ru/articles/489666/, (дата обращения - 14.11.2023)