МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО 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»
Вариант 20.

Калькулятор «большой» конечной арифметики.

Обучающийся: _____ Гладков И.А.

Руководитель: _____ Востров А.В.

«_____»_____20____г.

Санкт-Петербург, 2024

Содержание

Введение

Целью данной курсовой работы является разработка и реализация калькулятора «большой» конечной арифметики $< Z_8; +, * >$, то есть программы, которая может выполнять арифметические операции $(+, -, *, \div)$ над числами в конечном кольце $< Z_8; +, * >$. Для этого используется основание «малой» конечной арифметики, в которой сложение и умножение определены по правилу «+1», а также выполняются следующие свойства:коммутативность(+, *), ассоциативность(+, *), дистрибутивность * относительно +, существование нейтральных элементов: аддитивная единица «a» и мультипликативная единица «b». Программа также обеспечивает корректность ввода данных от пользователя и выводит результаты в удобном для чтения формате. Ниже представлено правило «+1» для данной курсовой работы.

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

1 Математическое описание

В этом разделе рассмотрены понятия и алгоритмы, которые лежат в основе реализации лабораторной работы.

1.1 Алгебраические структуры

Всюду определённая (тотальная) функция $\phi : M^n \to M$ называется n-арной(n-местной) операцией на M. Если операция ϕ — бинарная $\phi : (M \times M \to M)$, то будем писать в инфиксной форме $a\phi b$ вместо $\phi(a, b)$ или $a * b$, где * — знак операции. Множество M вместе с набором операций $\sum = \{\phi_1, \ldots, \phi_m\}, \phi_i : M^{n_i} \to M$, где n_i — арность операции ϕ_i, называется алгебраической структурой, универсальной алгеброй или просто алгеброй. При этом множество M называется основным (несущим) множеством или основой (носителем); вектор арностей $(n_1, \ldots, n_m)$ называется типом; множество операций $\sum$ называется сигнатурой. Операции ϕ_i конечноместны, сигнатура $\sum$ конечна. Носитель не обязательно конечен, но пе пуст.

1.1.1 Конечное коммутативное кольцо с единицей

Пусть M - конечное множество, а $+$ и $*$ - две операции на M. Тогда M называется конечным коммутативным кольцом с единицей, если выполняются следующие условия:

- $x + y = y + x$ и $x * y = y * x, \forall x, y \in M$ (коммутативность).

- $(x + y) + c = x + (y + c)$ и $(x * y) * c = x * (y * c) \forall x, y, c \in M$ (ассоциативность).

- $x * (y + c) = a * y + x * c$ и $(y + c) * x = y * x + c * x \forall x, y, c \in M$ (дистрибутивность).

- $\exists a \in M : x + a = a + x = x \forall x \in M$ (нейтральный элемент относительно сложения: аддитивная единица).

- $\exists b \in M : x * b = b * x = x \forall x \in M$ (нейтральный элемент относительно умножения: мультипликативная единица).

- $\forall x \in M \exists - x \in M : x + (-x) = (-x) + x = 0$ (противоположный элемент относительно сложения).

- Если $x * y = 0$, то либо $x = 0$, либо $y = 0$ (отсутствие нетривиальных делителей нуля).

Но кольце не все элементы имеют мультипликативный обратный, поэтому это свойство отсутствует. Примерами конечных коммутативных колец с единицей являются множество целых чисел, а также «малая» и «большая» конечные арифметики.

«Малая» конечная арифметика - это множество с двумя операциями, похожими на сложение и умножение. Эта алгебраическая структура обозначается как $< M; +, * >$. На множестве M также определены операции «вычитания» и частично «деления».

В этой курсовой работе множество M_8 состоит из 8 элементов, для которых действует правило «+1»: $\{a, b, c, d, e, f, g, h\}$. «Малую» конечную арифметику можно расширить до «большой» конечной арифметики, используя множество n-значных чисел, с арифметикой над M_8. В этой курсовой работе мы работаем с 8-значными числами, поэтому «большая» конечная арифметика обозначается

как $< M_8^8; +, * >$. Также стоит упомянуть, что на множестве M_8^8 определено действие «деление с остатком».

1.2 Вычисления отношения строгого линейного порядка и таблиц «малой» конечной арифметики

Как упоминалось ранее, для варианта 20 этой курсовой работы было установлено следующее дополнительное правило, обозначаемое как «+1»:

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

Как можно заметить, правило «+1» соответствует требованию «единственности результата». Имея в виду, что a является минимальным элементом, а $a + b$ представляет «единицу» в «малой» конечной арифметике, после выполнения ряда вычислений можно определить отношение строгого линейного порядка:

1. $a + b = b$
2. $b + b = f$
3. $f + b = h$
4. $h + b = e$
5. $e + b = d$
6. $d + b = g$
7. $g + b = c$

В процессе вычислений с 1 по 7 находится наименьший элемент, который больше текущего элемента. Полученное отношение строгого линейного порядка представлено на рисунке 1:

Для сложения «малой» конечной арифметики, нужно использовать полученное отношение и представить каждый элемент как сумму единиц. Тогда сложение двух элементов будет равно сумме их единиц, с учётом перехода от максимального элемента к минимальному. Например, e + h = 4 единицы + 3 единицы = 7 единиц = c. Также можно составить таблицу «переноса по сложению», в которой указано, сколько раз нужно переходить от максимального элемента к минимальному при сложении. Таблицы «сложения» и «переноса по сложению» показаны на таблицах 1 и 2. Аналогично можно составить таблицу умножения «малой» конечной арифметики, только необходимо находить не сумму единиц, а их произведение. Например, e + h = 4 единицы * 3 единицы = 12 единиц = 8 единиц + 4 единицы, следовательно получаем 1 перенос = b и 4 единицы = e. Таблицы «умножения» и «переноса по умножению» показаны на таблицах 3 и 4.

Рис. 1. Отношение строгого линейного порядка.

1.3 Таблицы операций малой конечной арифметики

Ниже представлены таблицы 1-4, которые отображают операции сложения и умножения с определенными нейтральными элементами для операции сложения (элемент a) и умножения (элемент b).

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

Таблица 1. Сложение.

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

Таблица 2. Перенос по сложению.

Примеры вычисления значений для таблиц сложения переноса по сложения(таблица 1-2):
В данных расчетах перенос будет указываться в квадратных скобках и будет увеличиваться на «b» (мультипликативная единица) при проходжении от максимального элемента к минимальному по таблице правила «+1»

- $d + a = [a]d$, так как «a» - аддитивная единица
- $d + b = [a]g$, так как «b» - мультипликативная единица (правило «+1»)
- $d+c = (d+g)+b = (d+d)+b+b = (d+e)+b+b+b = (d+h)+b+b+b+b = (d+f)+b+b+b+b+b = (d+b)+b+b+b+b+b+b = [a](g+b)+b+b+b+b+b = [a](c+b)+b+b+b+b = [b](a+b)+b+b+b = [b](b+b)+b+b = [b](f+b)+b = h+b = e$

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

Таблица 3. Умножение.

$*_s$	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	g	e	h	b	d	f
d	a	a	e	h	f	b	h	b
e	a	a	h	f	f	b	h	b
f	a	a	b	b	b	a	b	a
g	a	a	d	h	h	b	e	f
h	a	a	f	b	b	a	f	b

Таблица 4. Перенос по умножению.

Примеры вычисления значений для таблиц умножения переноса по умножения(таблица 1-2): В данных расчетах прри сложении элементов значения будут браться из таблицы слажения, переносы будут указываться в квадратных скобках и будут увеличиваться на значения из таблицы перехеодов по сложению при сложении двух элементов в примере.

- $d * b = [a]a$, так как «a» - аддитивная единица
- $d * b = [a]d$, так как «b» - мультипликативная единица
- $d * c = d * (g + b) = d * (g + b) = d * (d + b + b) = d * (e + b + b + b) = d * (h + b + b + b + b) = d * (f + b + b + b + b + b) = d * (b + b + b + b + b + b + b) = (d + d) + d + d + d + d + d = [b](f + d) + d + d + d + d = [b](c + d) + d + d + d = [f](e + d) + d + d = [h](b + d) + d = [h]g + d = [e]h.$

2 Особенности реализации

2.1 Класс Ar

Класс Tabl – основной класс в курсовой работе. Опираясь на правило «+1» класс создает таблицы малой конечной арифметики. Затем по созданным данным расширяет малую арифметику до большой конечной, используя множество 8-разрядных чисел. Он добавляет операции вычитания и деления с отатком. В классе Ar используются следующие переменные:

1. vector <char> PlusOneChar – массив хранящий внутри себя правило «+1», то есть содержащий символы, которые представляют собой результат сложения каждого символа с единицей.
2. char Graph[8] – массив, представляющий отношение порядка между символами.
3. regex power – регулярное выражение для проверки вводимых чисел.
4. string buf – строка, в которую записывается ввод пользователя, при выборе действия в главном меню.
5. string a – строка отвечающая за первый число в операциях.
6. string b – строка отвечающая за первый число в операциях.
7. char OperatorPlusChar[8][8] – массив, хранящий в себе значения таблицы сложения.
8. char OperatorPlusChar_Shift[8][8]– массив, хранящий в себе значения таблицы переноса по сложения.
9. char OperatorMultChar[8][8] – массив, хранящий в себе значения таблицы умножения.
10. char OperatorMultChar_Shift[8][8] – массив, хранящий в себе значения таблицы переноса по умножению
11. char Dictionary[8] = 'a','b','c','d','e','f','g','h' – массив хранящий числа как бы в алфавитном порядке(так будут идти числа в названиях строк и столбцов всех таблиц, так удобнее для восприятия пользователем). Также в описаниях он будет называться массивом названий.

2.1.1 Конструктор Ar

Вход: создание таблицы правила «+1», отношение порядка, а также таблиц сложения , переноса по сложению, умножения и переноса по умножению.

Выход: сформированные отношение порядка и таблиц правила «+1» сложения , переноса по сложению, умножения и переноса по умножению.

Вначале в массив отвечающий за правило «+1» добавляются элементы, порядок элементов производится в соответствии с вариантом. После формируются отношение порядка. Формируется регулярное выражение для корректности ввода числа пользователем. Затем вызываются методы формирования таблиц сложения, переноса по сложению, умножения и переноса по умножению(листинг 1).

```
1    Ar::Ar() {
2      PlusOneChar.push_back('b');
3      PlusOneChar.push_back('f');
```

```
    PlusOneChar.push_back('a');
    PlusOneChar.push_back('g');
    PlusOneChar.push_back('d');
    PlusOneChar.push_back('h');
    PlusOneChar.push_back('c');
    PlusOneChar.push_back('e');

    Graph[0] = 'a';
    Graph[1] = 'b';
    Graph[2] = 'f';
    Graph[3] = 'h';
    Graph[4] = 'e';
    Graph[5] = 'd';
    Graph[6] = 'g';
    Graph[7] = 'c';

    power= ("^(-?[a-h]{1,8})$");

    CreateOperatorPlus();
    CreateOperator_Shift(OperatorPlusChar_Shift,0);
    CreateOperatorMult();
    CreateOperator_Shift(OperatorMultChar_Shift,1);
  }

```

Листинг 1. Конструктор Ar

2.1.2 Конструктор CreateOperatorPlus

Вход: создание таблицы сложения.

Выход: сформированная таблица сложения.

Первая строчка и первый столбец таблицы сложения – это все цифры, которые как бы идут в алфавитном порядке, но на самом деле у них разные значения, это проще для визуального восприятия в таблице. Вторая строчка и второй столбец – это числа из таблицы «+1», повторяющие порядок, так как просто прибавляется «b», то есть мультипликативная единица. Далее находятся все остальные значения таблицы при помощи метода Summ, и записываются сразу в два поля таблицы, так как она симметричная. Листинг 2.

```
  void Ar::CreateOperatorPlus() {
    for (int i = 0; i < 8; i++) {
      OperatorPlusChar[i][0] = Dictionary[i];
      OperatorPlusChar[0][i] = Dictionary[i];
      OperatorPlusChar[i][1] = PlusOneChar[i];
      OperatorPlusChar[1][i] = PlusOneChar[i];
    }

```

```
      for (int i = 2; i < 8; i++) {
        for (int j = i; j < 8; j++) {
          char buf = Summ(Dictionary[i], Dictionary[j]);
          OperatorPlusChar[i][j] = buf;
          OperatorPlusChar[j][i] = buf;

        }
      }
    }

```

Листинг 2. Конструктор CreateOperatorPlus

2.1.3 Метод Summ

Вход: два символа из таблицы которые нужно сложить

Выход: результат сложения входных символов.

Значения находятся по сумме количеств мультипликативных единиц(«b») в числах, а точнее считает сколько «b» во втором числе операции, потом к первому числу прибавляет единицу нужное количесво раз по правилу «+1»(листинг 3). В методе используются методы SearchIndex DictionaryToInt. Они будут описаны ниже, листинге 4-5.

```
char Ar::Summ(char a, char b) {
  int k = 0;
  char result = a;
  char buf = b;
  char check = b;
  while (check != 'b') {
    buf = Dictionary[SearchIndex(buf)];
    check = PlusOneChar[DictionaryToInt(buf)];
    k++;
  }
  while (k > 0) {
    result = PlusOneChar[DictionaryToInt(result)];
    k--;
  }
  return result;
}

```

Листинг 3. Метод Summ

2.1.4 Методы SearchIndex, DictionaryToInt, SearchInGraph

Метод SearchIndex находит индекс в массиве, который отвечает за правило «+1». Используется в методах, где необходимо начать прибавлять по одному начиная с какогото числа.

Вход: число, индекс которого необходимо найти.

Выход: индекс входного числа в массиве правила «+1»

```
int Ar::SearchIndex(char a) {
  for (int i = 0; i < 8; i++)
  if (PlusOneChar[i]== a)
  return i;
};

```

Листинг 4. Метод SearchIndex

Метод DictionaryToInt находит индекс цифры в строке или столбце названий таблиц. Используется в различных методах, где это необходимо.

Вход: число, индекс которого необходимо найти.

Выход: индекс входного числа в массиве названий.

```
int Ar::DictionaryToInt(char s) {
  for (int i = 0; i < 8; i++) {
    if (Dictionary[i] == s)
    return i;
  }
}

```

Листинг 5. Метод DictionaryToInt

Метод SearchInGraph находит индекс числа в массиве отношении порядка, можно сказать, что он находит количество единиц в цифре. Используется в различных методах, где это необходимо.

Вход: число, индекст которого необходимо найти.

Выход: индекс входного числа в массиве отношении порядка, то есть количество единиц в цифре.

```
      int Ar::SearchInGraph(char a) {
      for (int i = 0; i < 8; i++)
      if (a == Graph[i])
      return i;
    };

```

Листинг 6. Метод SearchInGraph

2.1.5 Метод CreateOperatorMult

Вход: создание таблицы умножения.

Выход: сформированная таблица умножения.

Первая строчка и первый столбец таблицы умножения – это аддитивные единицы, а именно цифры «a». Вторая строчка и второй столбец – это числа из таблицы значений. Далее находятся все

остальные значения таблицы при помощи метода Mult, и записываются сразу в два поля таблицы, так как она симметричная. Смотрите листинг 7.

```
1   void Ar::CreateOperatorMult() {
2     for (int i = 0; i < 8; i++) {
3       OperatorMultChar[i][0] = 'a';
4       OperatorMultChar[0][i] = 'a';
5       OperatorMultChar[i][1] = Dictionary[i];
6       OperatorMultChar[1][i] = Dictionary[i];
7     }
8     for (int i = 2; i < 8; i++) {
9       for (int j = i; j < 8; j++) {
10        char buf = Mult(Dictionary[i], Dictionary[j]);
11        OperatorMultChar[i][j] = buf;
12        OperatorMultChar[j][i] = buf;
13
14      }
15    }
16  }
17
```

Листинг 7. Метод CreateOperatorMult

2.1.6 Метод Mult

Вход: два символа из таблицы которые нужно перемножить.

Выход: результат произведения входных символов.

Находится количество «b» во второй числе операции, затем данное количество раз будет складываться первое число с самим собой, то есть происходит умножение (листинг 8). Для реализации данных чисел используется метод SearchInPlus, описанный ниже, в листинге 10.

```
1   char Ar::Mult(char a, char b) {
2     char result = a;
3     char buf = a;
4     int k = SearchInGraph(b) - 1;
5     while (k > 0) {
6       result = SearchInPlus(result, buf);
7       k--;
8     }
9     return result;
10  }
11
```

Листинг 8. Метод Mult

2.1.7 Метод CreateOperator_Shift

Вход: флаг, показывающий операцию: 1 - умножение, 0 - сложения.

Выход: сформированный массив переноса по умножению, либо по сложению, в зависимости от входного параметра.

В методе происходит, либо умножение, либо сложение количества единиц двух цифр, в зависимости от входного параметра. Затем результат делится на 8 и результат деления находится в таблице отношения порядка, то есть результирующее количество единиц интерпретируется в цифру по таблице отношения порядка.

```
1   void Ar::CreateOperator_Shift( bool flag) {
2     int k;
3     for (int i = 0; i < 8; i++) {
4       for (int j = 0; j < 8; j++) {
5         k = flag? SearchInGraph(Dictionary[i]) * SearchInGraph(Dictionary[j])
6         : SearchInGraph(Dictionary[i]) + SearchInGraph(Dictionary[j]);
7         k = k / 8;
8         if (flag)
9         OperatorMultChar_Shift[i][j] = Graph[k];
10        else
11        OperatorPlusChar_Shift[i][j] = Graph[k];
12      }
13    }
14  }
15
```

Листинг 9. Метод CreateOperator_Shift

2.1.8 Методы SearchInPlus, SearchInPlus_Shift, SearchInMult, SearchInMult_Shift, SearchInSub

Метод SearchInPlus

Вход: два символа, для которых нужно найти значени в таблице сложения.

Выход: значение сложения двух символов из таблицы .

В методе происходит поиск по таблице сложения, найденный результат возвращается.

```
1   char Ar::SearchInPlus(char a, char b) {
2     return OperatorPlusChar[DictionaryToInt(a)][DictionaryToInt(b)];
3   };
4
```

Листинг 10. Метод SearchInPlus

Метод SearchInPlus_Shift

Вход: два символа, для которых нужно найти значени в таблице переноса по сложению.

Выход: значение переноса при сложении двух символов из таблицы .

В методе происходит поиск по таблице переноса по сложению, найденный результат возвращается.

```
1       char Ar::SearchInPlus_Shift(char a, char b) {
2         return OperatorPlusChar_Shift[DictionaryToInt(a)][DictionaryToInt(b)];
3       };
4
```

Листинг 11. Метод SearchInPlus_Shift

Метод SearchInMult

Вход: два символа, для которых нужно найти значение в таблице умножения

Выход: значение сложения двух символов из таблицы умножения .

В методе происходит поиск по таблице умножения, найденный результат возвращается.

```
1       char Ar::SearchInMult(char a, char b) {
2         return OperatorMultChar[DictionaryToInt(a)][DictionaryToInt(b)];
3       };
4
```

Листинг 12. Метод SearchInMult

Метод SearchInMult_Shift

Вход: два символа, для которых нужно найти значени в таблице переноса по уменожению

Выход: значение переноса приумножении двух символов из таблицы . В методе происходит поиск по таблице переноса по умножению, найденный результат возвращается.

```
1     char Ar::SearchInMult_Shift(char a, char b) {
2       return OperatorMultChar_Shift[DictionaryToInt(a)][DictionaryToInt(b)];
3     };
4
```

Листинг 13. Метод SearchInMult_Shift

Метод SearchInSub

Вход: два символа, разность которых нужно найти

Выход: значение разности двух символов из таблицы сложения .

В методе происходит поиск по таблице сложения, первая цифра является итогом сложения по таблице сложения, вторая будут находится в названии столбца либо строки, тогда результат будет находится в строке, либо столбце соответственно, так как таблица симметрична, относительно главной диагонали.

```
1     char Ar::SearchInSub(char a, char b) {
2       for (int i = 0; i < 8; i++) {
3         if (OperatorPlusChar[i][DictionaryToInt(b)] == a) {
4           return Dictionary[i];
5         }
6       }
7     };
8
```

Листинг 14. Метод SearchInSub

2.1.9 Методы Comparasion, compare

Метод Comparasion:

Вход: Два символа, которые неоходимо стравнить.

Выход: вывод, первая меньше второй или нет.

Метод сравнивает количество единиц в первой цифре, со второй. Если в первой меньше, то возвращается true, иначе false.

```
bool Ar::Comparasion(char a, char b) {
  for (int i = 0; i < 8; i++) {
    for (int j = 0; j < 8; j++) {
      if (Graph[i] == a && Graph[j] == b)
      if (i < j)
      return true;
      else
      return false;
    }
  }
}

```

Листинг 15. Метод Comparasion

Метод compare:

Вход: Два числа, которые необходимо сравнить.

Выход: вывод, больше, меньше, или равно.

Метод сравнивает два числа начиная со старшего разряда. Если они разных разрядов, то меньшеразрядное число заполняется незначащими нулями. Далее происходит по разрядная проверка. Если разряд первого чмсла меньше второго, то выводится «-1» (первое число меньше второго) если больше, то «1»(второе число больше первого), иначе по окончании цикла выводтся «0», это означает, что они равны.

```
flag = compare(a, b);
if ((minuses_a + minuses_b) == 1 && fl == 1
|| (minuses_a + minuses_b) != 1 && fl == 0) {
  answer = ToSub(a, b);
  if (answer.size() > 8)
  answer = "overflow";
  if (answer != "overflow" &&
  ((minuses_b == 1 && flag == -1 && fl == 1) ||
  (minuses_a == 1 && flag == 1 && fl == 1) ||
  ((minuses_a + minuses_b) == 0 && flag == -1 && fl == 0) ||
  ((minuses_a + minuses_b) == 2 && flag == 1 && fl == 0))) {
```

```
12            answer = '-' + answer;
13          }
14        }
15
```

Листинг 16. Метод compare

2.1.10 Метод ToSumm

Вход: два числа, которые необходимо сложить.

Выход Результат сложения двух чиселю.

Метод реализует сложение в столбик, то есть сложение происходит поразрядно.Начинается все с младшего разряда. Если происходит переход при сложении, то он прибавляется к следующему разряду, и так пока не закончатся разряды.

```
1       string Ar::ToSumm(string a, string b) {
2         char shift1 = 'a';
3         char shift2 = 'a';
4         char shift3 = 'a';
5         char solution;
6         string answer = "";
7         while (a.size() != b.size()) {
8           a.size() < b.size() ? a = 'a' + a : b = 'a' + b;
9         }
10        for (int i = a.size() - 1; i >= 0; i--) {
11          solution = SearchInPlus(a[i], b[i]);
12          shift2 = SearchInPlus_Shift(a[i], b[i]);
13          shift3 = SearchInPlus_Shift(solution, shift1);
14          solution = SearchInPlus(solution, shift1);
15          shift1 = SearchInPlus(shift2, shift3);
16          answer = solution + answer;
17          shift3 = 'a';
18          shift2 = 'a';
19          if (i == 0 && shift1 != 'a')
20          answer = shift1 + answer;
21        }
22        while (answer.size() > 1 && answer[0] == 'a')
23        answer.erase(0, 1);
24
25        return answer;
26      }
27
```

Листинг 17. Метод ToSumm

2.1.11 Метод ToSub

Вход: два числа, с которыми необходимо выполить разность.

Выход: Результат вычитания двух чисел.

Метод реализует вычитание в столбик, то есть происходит поразрядно. Все начинается с младшего разряда. Если символ в первом числе меньше символа во втором, происходит уменьшение, цифра разряда старше на «b», и в данном разряде происходит вычитание из наибольшого символа (той, которая содержит 7 единиц, то есть «c»), затем прибавляется «b» и после прибавляется изначальная символ, иначе с помощью метода SearchInSub нахobдтся разность символов.Листинг 18.

```
1       string Ar::ToSub(string a, string b) {
2         int flag;
3         char solution;
4         string answer = "";
5         flag = compare(a, b);
6         if (flag == 0) {
7           answer = 'a';
8         }
9         else {
10          if (flag == -1) {
11            a.swap(b);
12          }
13          for (int i = a.size() - 1; i >= 0; i--) {
14            if (i > 0) {
15              if (Comparasion(a[i], b[i])) {
16                int j = i - 1;
17                while (j >= 0){
18                  if (a[j] != 'a') {
19                    a[j] = SearchInSub(a[j], 'b');
20                    break;
21                  }
22                  else
23                  a[j] = SearchInSub(a[j], 'b');
24                  j--;
25                }
26                solution = SearchInSub('c', b[i]);
27                solution = SearchInPlus('b', solution);
28                solution = SearchInPlus(a[i], solution);
29              }
30              else {
31                solution = SearchInSub(a[i], b[i]);
32              }
33              answer = solution + answer;
34            }
35            else {
36              solution = SearchInSub(a[i], b[i]);
```

```
37                answer = solution == 'a' ? answer : solution + answer;
38              }
39            }
40          }
41          while (answer.size() > 1 && answer[0] == 'a')
42          answer.erase(0, 1);
43
44          return answer;
45        }
46
```

Листинг 18. Метод ToSub

2.1.12 Метод ToSumm_or_ToSub

Вход: два числа, которые ввел пользователь, либо в калькуляторе сложения, либо вычитания, а также флаг операции.

Выход:конечный результат операции над данными числами.

Данный метод, определяет, когда ему совершить операцию сложения, а когда вычитания, так как все зависит от входных данных, а именно у какого числа есть минус и входной флаг. Например, если флаг – 0 (т.е. вычитание) и на вход подается первое число положительное, а второе отрицательное, то из-за правила – минус на минус дает плюс– стоит выполнить сложение. Или, если в сложении, одно из чисел отрицательное, то стоит выполнить вычитание. При определенных условиях вызываются методы сложение или вычитания (ToSumm, ToSub). Смотрите листинг 19.

```
1       string Ar::ToSumm_or_ToSub(string a, string b, bool fl) {
2         string answer = "";
3         int minuses_a = 0;
4         int minuses_b = 0;
5         int flag;
6         if (a[0] == '-') {
7           minuses_a++;
8           a.erase(0, 1);
9         }
10        if (b[0] == '-') {
11          minuses_b++;
12          b.erase(0, 1);
13        }
14        while (a.size() != b.size()) {
15          a.size() < b.size() ? a = 'a' + a : b = 'a' + b;
16        }
17
18        flag = compare(a, b);
19        if ((minuses_a + minuses_b) == 1 && fl == 1
20        || (minuses_a + minuses_b) != 1 && fl == 0) {
21          answer = ToSub(a, b);
```

```
22            if (answer.size() > 8)
23              answer = "overflow";
24              if (answer != "overflow" &&
25              ((minuses_b == 1 && flag == -1 && fl == 1) ||
26              (minuses_a == 1 && flag == 1 && fl == 1) ||
27              ((minuses_a + minuses_b) == 0 && flag == -1 && fl == 0) ||
28              ((minuses_a + minuses_b) == 2 && flag == 1 && fl == 0))) {
29                answer = '-' + answer;
30              }
31          }
32          else {
33            answer = ToSumm(a, b);
34            if (answer.size() > 8)
35            answer = "overflow";
36            if (answer != "overflow" &&
37            ((minuses_a == 1 && flag == 1 && fl ==0) ||
38            (minuses_a == 1 && flag == -1 && fl == 0) ||
39            ((minuses_a + minuses_b) == 2 && flag == -1 && fl == 1) ||
40            ((minuses_a + minuses_b) == 2 && flag == 1 && fl == 1)))
41            answer = '-' + answer;
42          }
43          return answer;
44        }
45
```

Листинг 19. Метод ToSumm_or_ToSub

2.1.13 Метод ToMult_Base

Вход: два числа, которые необходимо перемножить.

Выход Результат произведения двух чисел.

В данном методе прописанна оснавноая логика произведения чисел. Умножение производится в столбик: один разряд второго числа,начиная с младшего, перемножается с каждым разрядом первого числа, при этом если происходит перенос по таблице переносов по умножению, то он складывеется в более старший разряд. произведение символов находится по таблице умножения. Затем происходит возвращение результата(листинг 20).

```
1
2         string Ar::ToMult_Base(string a, string b) {
3           string answer = "a";
4           string answer_buf = "";
5           char shift1 = 'a';
6           char shift2 = 'a';
7           char shift3 = 'a';
8           char solution;
9           int flag;
10
```

```
11
12        int k = 0;
13        for (int i = b.size() - 1; i >= 0; i--) {
14          for (int j = a.size() - 1; j >= 0; j--) {
15            solution = SearchInMult(a[j], b[i]);
16            shift2 = SearchInMult_Shift(a[j], b[i]);
17            shift3 = SearchInPlus_Shift(solution, shift1);
18            solution = SearchInPlus(solution, shift1);
19            shift1 = SearchInPlus(shift2, shift3);
20            answer_buf = solution + answer_buf;
21            shift3 = 'a';
22            shift2 = 'a';
23            if (j == 0 && shift1 != 'a')
24            answer_buf = shift1 + answer_buf;
25          }
26
27          shift1 = 'a';
28          int s = k;
29          while (s > 0) {
30            answer_buf += 'a';
31            s--;
32          }
33          k++;
34          answer = ToSumm(answer, answer_buf);
35          answer_buf = "";
36        }
37
```

Листинг 20. Метод ToMult_Base

2.1.14 Метод ToMult

Вход: два числа,введенные пользователем, которые необходимо перемножить.

Выход Конечный результат произведения двух чисел.

Данный метод оценивает входные данные, вызывает метод ToMult_Base, для перемножения чисел без минусов, затем на основании собранных данных, либо оставляет ответ без минусов, либо добавляет минуст в ответ(листинг 21).

```
1       string Ar::ToMult(string a, string b) {
2         string answer = "a";
3         int minuses_a = 0;
4         int minuses_b = 0;
5         if (a[0] == '-') {
6           minuses_a++;
7           a.erase(0, 1);
8         }
9         if (b[0] == '-') {
```

```
10            minuses_b++;
11            b.erase(0, 1);
12          }
13          while (a.size() != b.size()) {
14            a.size() < b.size() ? a = 'a' + a : b = 'a' + b;
15          }
16
17          answer = ToMult_Base(a, b);
18
19          while (answer.size() > 1 && answer[0] == 'a')
20          answer.erase(0, 1);
21
22          if (answer.size() > 8)
23          answer = "overflow";
24
25          if (answer[0] != 'a' && answer != "overflow" && (minuses_a + minuses_b) == 1)
26          answer = '-' + answer;
27          return answer;
28
```

Листинг 21. Метод ToMult

2.1.15 Метод ToDiv

Вход: два числа,введенные пользователем, которые необходимо резделить.

Выход: Конечный результат деления двух чисел.

Данный метод вначале проверяет, не подается ли на вход нулевое значение (аддитивная единица), если да, то обрабатываются случаи: если делимое – нулевое (аддитивная единица), то в ответе получаем нулевое значение(аддитивная единица), если делитель нулевой (аддитивная единица), то ответа не существет, если и делимое, и делитель – аддитивные единицы, то ответ – весь возможный промежуток. При четвертом случае, происходит деление модулей чисел, а именно частное увеличивается пока произведение частного на делитель, меньше или равно делимому. Когда нахоится максимальное возможное частное подходящее под условие, то после находится остаток – разность делимого с произведением частного на делителя. Также если делимое и делитель положительные, то часное тоже положительно. Если делимое положительное и делитель отрицательное, то находится частное по модулям делителя и делимого, после к частному прибавляется минус. Если делимое отрицательное, делитель положительный, то также находится частное их модулей, затем к частному прибавляется минус и отнимается мультипликативная единица. Если оба отрицательные, то находится частное модулей, прибавляется мультипликативная единица. Частное всегда положительное и находится по формуле: разность делимого с произведением частного на делителя. Операция деления описана в листинге 22.

```
1       vector <string> Ar::ToDiv(string a, string b, vector<string> result) {
2         //string buf = a;
```

```
3         string answer = "a";// делитель
4         string answer_buf = "a";//частное
5         int minuses_a = 0;
6         int minuses_b = 0;
7         char shift1 = 'a';
8         char shift2 = 'a';
9         char shift3 = 'a';
10        char solution;
11        int flag;
12
13        if (a[0] == '-') {
14          minuses_a++;
15          a.erase(0, 1);
16        }
17        if (b[0] == '-') {
18          minuses_b++;
19          b.erase(0, 1);
20        }
21
22        if (compare(a, "a") == 0 && compare(b, "a") == 0) { //две переменные равны "a" ->
      запысываем в вектор "1"
23          result.push_back(answer);
24          result.push_back(answer_buf);
25          result.push_back("1");
26        }
27        else if (compare(a, "a") != 0 && compare(b, "a") == 0) {//только делитель равен "
    a" -> записываем "2"
28          result.push_back(answer);
29          result.push_back(answer_buf);
30          result.push_back("2");
31        }
32        else if (compare(a, "a") == 0 && compare(b, "a") != 0) {//только делимое равено "
    a" -> записываем "0",
33          //то есть получаем какоето число, а именно "a"
34          result.push_back(answer);
35          result.push_back(answer_buf);
36          result.push_back("0");
37        }
38        else {
39
40          while (a.size() != b.size()) {
41            a.size() < b.size() ? a = 'a' + a : b = 'a' + b;
42          }
43
44          while (true) {
45            if (compare(ToMult_Base(b, ToSumm(answer, "b")), a) != 1) {
46              if (compare(ToMult_Base(b, ToSumm(answer, "ba")), a) != 1) {
47                if (compare(ToMult_Base(b, ToSumm(answer, "baa")), a) != 1) {
```

```
                if (compare(ToMult_Base(b, ToSumm(answer, "baaa")), a) != 1) {
                  if (compare(ToMult_Base(b, ToSumm(answer, "baaaa")), a) != 1) {
                    if (compare(ToMult_Base(b, ToSumm(answer, "baaaaa")), a) != 1) {
                      if (compare(ToMult_Base(b, ToSumm(answer, "baaaaaa")), a) != 1) {
                        answer = ToSumm(answer, "baaaaaaa");
                      }
                      else
                      answer = ToSumm(answer, "baaaaaa");
                    }
                    else
                    answer = ToSumm(answer, "baaaaa");
                  }
                  else
                  answer = ToSumm(answer, "baaaa");
                }
                else
                answer = ToSumm(answer, "baaa");
              }
              else

              answer = ToSumm(answer, "baa");
            }
            else
            answer = ToSumm(answer, "ba");
          }
          else
          answer = ToSumm(answer, "b");
        }
        else
        break;
      }
      while (a.size() > 1 && a[0] == 'a')
      a.erase(0, 1);

      a = minuses_a == 1 ? '-' + a : a;

      while (b.size() > 1 && b[0] == 'a')
      b.erase(0, 1);

      b = minuses_b == 1 ? '-' + b : b;

      if (minuses_a == 0 && minuses_b == 1) {
        answer = '-' + answer;
        //answer_buf = ToSub(a, ToMult(b, answer));
      }
      else if (minuses_a == 1 && minuses_b == 0) {
```

```
95              answer = '-' + answer;
96              string m =  ToMult(b, answer);
97              if (compare(ToMult(b, answer), a) != 0)
98              answer = ToSumm_or_ToSub(answer, "b", 0); //вычитаем один
99              //answer_buf = ToSub(a, ToMult(b, answer));
100           }
101           else if (minuses_a + minuses_b == 2) {
102             if (compare(ToMult(b, answer), a) != 0)
103             answer = ToSumm_or_ToSub(answer, "b", 1);// прибавляем 1
104           }
105
106           answer_buf = ToSumm_or_ToSub(a, ToMult(b, answer), 0);
107
108           result.push_back(answer);
109           result.push_back(answer_buf);
110           result.push_back("0");
111         }
112         return result;
113
114       }
115
```

Листинг 22. Метод ToDiv

2.1.16 Метод Menu

Вход: ожидание вывести возможные действия работы программы пользователю.

Выход: выводится меню действий на консоль, позьзователь выбирает нужное для него действие.

В методе Menu происходит вывод на экран всех возможных действий. Пользователь выбирает нужное ему действие, при этом вызывается метод для проверки пользовательскtго ввода, дальше вызывается метот соответствующий выбору пользователя.

```
1  void Ar::Menu() {
2    cout << "\n\tДобро пожаловать в калькулятор \"большой\" конечной арифметики\n";
3    while (true) {
4      bool out = 0;
5      printf("\n\nВот что вы можете сделать:");
6      printf("\n Вывести ... ");
7      printf("\n [1] Правило \"+1\"");
8      printf("\n [2] Таблицу сложения \"+\"");
9      printf("\n [3] Таблицу умножения \"*\"");
10     printf("\n [4] Отношение порядка");
11     printf("\n Перейти в... ");
12     printf("\n [5] Калькулятор сложения\"+\"");
13     printf("\n [6] Калькулятор вычитания\"-\"");
14     printf("\n [7] Калькулятор умножения\"*\"");
```

```
15      printf("\n [8] Калькулятор деления\"/\"");
16      printf("\n Выйти из... ");
17      printf("\n [0] Программы\n\n");
18
19      while (true) {
20        cin >> buf;
21        try {
22          if (stoi(buf) >= 0 && stoi(buf) <= 8) {
23            break;
24          }
25          else {
26            std::cout << "Ошибка ввода! Ведите число от 0 до 8\n";
27          }
28        }
29        catch (const std::invalid_argument& e) {
30          std::cout << "Ошибка ввода! Неверный формат числа\n";
31        }
32
33      }
34      int n = stoi(buf);
35
36      switch (n) {
37        case 0:
38        out = 1;
39        break;
40
41        case 1:
42        PrintPlusOneChar();
43        break;
44
45        case 2:
46        PrintOperator(OperatorPlusChar);
47        PrintOperator(OperatorPlusChar_Shift);
48        break;
49
50        case 3:
51        PrintOperator(OperatorMultChar);
52        PrintOperator(OperatorMultChar_Shift);
53        break;
54        case 4:
55        PringGraph();
56        break;
57
58        case 5:
59        SummCalculator();
60        break;
61
62        case 6:
```

```
63        SubCalculator();
64        break;
65
66        case 7:
67        MultCalculator();
68        break;
69
70        case 8:
71        DivCalculator();
72        break;
73
74
75
76      }
77      if (out)
78      break;
79      system("pause");
80    }
81  }
82
```

Листинг 23. Метод Menu

2.1.17 Метод DivCalculator

Вход: запрашивается у пользователя два числа для операции деления.

Выход: результат операции деления на экран с введенными числами.

Метод вызывает функцию запроса числа от пользователя два раза, после вызывает функцию деления для данных чисел. Потом вывод результата на экран.

```
1       void Ar::DivCalculator() {
2         vector<string> answer;
3         printf("\n ---------------------------------------");
4         printf("\n Вы перешли в калькулятор Деления.");
5         printf("\n Для выхода в главное меню введите [0].\n\n");
6         while (true) {
7           printf("Что вы хотите поделить?\n");
8           a = Check();
9           if (a == "0")
10          break;
11          printf(" \"/\"\n");
12          b = Check();
13          if (b == "0")
14          break;
15          answer=ToDiv(a, b, answer);
16          if (answer[2] == "0") {
17            if (b[0] == '-')
18            std::cout << "\nОтвет: " << a << " / (" << b << ") = " << answer[0] << "\n";
```

```
            else
            std::cout << "\nОтвет: " << a << " / " << b << " = " << answer[0] << "\n";
            std::cout << "       остаток = " << answer[1] << "\n\n";
            std::cout << "Проверка остатка:\n";

            std::cout<< a << " - ";

            if (b[0] == '-')
            std::cout << "(";
            std::cout << b;
            if (b[0] == '-')
            std::cout << ")";

            std::cout << " * ";

            if (answer[0][0] == '-')
            std::cout << "(";
            std::cout << answer[0];
            if (answer[0][0] == '-')
            std::cout << ")";

            std::cout<<" = "<< a << " - ";

            if (ToMult(b,answer[0])[0] == '-')
            std::cout << "(";
            std::cout << ToMult(b, answer[0]);
            if (ToMult(b,answer[0])[0] == '-')
            std::cout << ")";

            std::cout<<" = "<< ToSumm_or_ToSub(a,ToMult(b,answer[0]),0)<<"\n\n";


          }
          else if (answer[2] == "1") {
            std::cout << "\nОтвет: [-cccccccc; cccccccc]"  << "\n\n";
          }
          else {
            std::cout << "\nОтвет: пустое множество" << "\n\n";
          }
          answer.clear();
        }
      }

```

Листинг 24. Метод DivCalculator

2.1.18 Метод MultCalculator

Вход: запрашивается у пользователя два числа для операции умножения.

Выход: результат операции умножения на экран с введенными числами.

Метод вызывает функцию запроса числа от пользователя два раза, после вызывает функцию умножения для данных чисел. Потом вывод результата на экран.

```
1   void Ar::MultCalculator() {
2     string answer = "";
3     printf("\n ---------------------------------------");
4     printf("\n Вы перешли в калькулятор умножения.");
5     printf("\n Для выхода в главное меню введите [0].\n\n");
6     while (true) {
7       printf("Что вы хотите перемножить?\n");
8       a = Check();
9       if (a == "0")
10      break;
11      printf(" \"*\"\n");
12      b = Check();
13      if (b == "0")
14      break;
15      answer=ToMult(a, b);
16      if (answer == "overflow") {
17        std::cout << "\nОтвет: " << answer << "\n";
18      }
19      else if (b[0] == '-')
20      std::cout << "\nОтвет: " << a << " * (" << b << ") = " << answer << "\n\n";
21      else
22      std::cout << "\nОтвет: " << a << " * " << b << " = " << answer << "\n\n";
23    }
24  }
25
```

Листинг 25. Метод MultCalculator

2.1.19 Метод SubCalculator

Вход: запрашивается у пользователя два числа для операции вычитания.

Выход: результат операции вычитания на экран с введенными числами.

Метод вызывает функцию запроса числа от пользователя два раза, после вызывает функцию вычитания для данных чисел. Потом вывод результата на экран.

```
1   void Ar::SubCalculator() {
2     string answer = "";
3     printf("\n ---------------------------------------");
4     printf("\n Вы перешли в калькулятор вычитания.");
5     printf("\n Для выхода в главное меню введите [0].\n\n");
6     while (true) {
```

```
7           printf("Что вы хотите вычесть?\n");
8           a = Check();
9           if (a == "0")
10          break;
11          printf(" \"-\"\n");
12          b = Check();
13          if (b == "0")
14          break;
15          answer=ToSumm_or_ToSub(a, b,0);
16          if (answer == "overflow") {
17            std::cout << "\nОтвет: " << answer << "\n";
18          }
19          else if (b[0] == '-')
20          std::cout << "\nОтвет: " << a << " - (" << b << ") = " << answer << "\n\n";
21          else
22          std::cout << "\nОтвет: " << a << " - " << b << " = " << answer << "\n\n";
23        }
24      }
25
```

Листинг 26. Метод SubCalculator

2.1.20 Метод SummCalculator

Вход: запрашивается у пользователя два числа для операции сложения.

Выход: результат операции сложения на экран с введенными числами.

Метод вызывает функцию запроса числа от пользователя два раза, после вызывает функцию сложения для данных чисел. Потом вывод результата на экран.

```
1       void Ar::SummCalculator() {
2         string answer = "";
3         printf("\n ----------------------------------------");
4         printf("\n Вы перешли в калькулятор сложения.");
5         printf("\n Для выхода в главное меню введите [0].\n\n");
6         while (true) {
7           printf("Что вы хотите сложить?\n");
8           a = Check();
9           if (a == "0")
10          break;
11          printf("\"+\"\n");
12          b = Check();
13          if (b == "0")
14          break;
15          answer=ToSumm_or_ToSub(a, b,1);
16          if (answer == "overflow") {
17            std::cout << "\nОтвет: "  << answer << "\n";
18          }
19          else if (b[0] == '-')
```

```
20          std::cout << "\nОтвет: " << a << " + (" << b << ") = " << answer << "\n";
21          else
22          std::cout << "\nОтвет: " << a << " + " << b << " = " << answer << "\n";
23          printf(" ---------------------------------------\n");
24
25        }
26      }
27
```

Листинг 27. Метод SummCalculator

2.1.21 Метод PringGraph

Вход: ожидание вывода на экран отношения порядка.

Выход: Вывод отношения подрядка на экран

Метод проходит по массиву, хранящем отношение порядка и выводт его на консоль.

```
1     void Ar::PringGraph() {
2       cout << "\nОтношение порядка\n\n";
3       for (int i = 0; i < 8; i++) {
4         cout << Graph[i];
5         if (i != 7)
6         cout << " < ";
7       }
8       cout << endl<<endl;
9     }
10
```

Листинг 28. Метод PringGraph

2.1.22 Метод PringGraph

Вход: ожидание корректного ввода числа пользователем

Выход: Корректный пользовательский ввод и число пригодное для работы в данной программе

Метод проверяет ввод с регулярным выражением и со случаем,е сли он введет «0».

```
1   string Ar::Check() {
2     while (true) {
3       cin >> buf;
4       if (regex_match(buf, power) && buf != "-" || buf == "0") {
5         break; // Ввод соответствует регулярному выражению, выходим из цикла
6       }
7       else {
8         std::cout << "Ошибка ввода! Ведите число в диапозоне [-cccccccc;cccccccc]\n";
9       }
10    }
11    return buf;
```

```
12      }
13
```

Листинг 29. Метод Check

3 Результаты работы программы

При запуске программы на консоль выводится главное меню с перечнем возможных действий, которые может выбрать пользователь (рисунок 2).

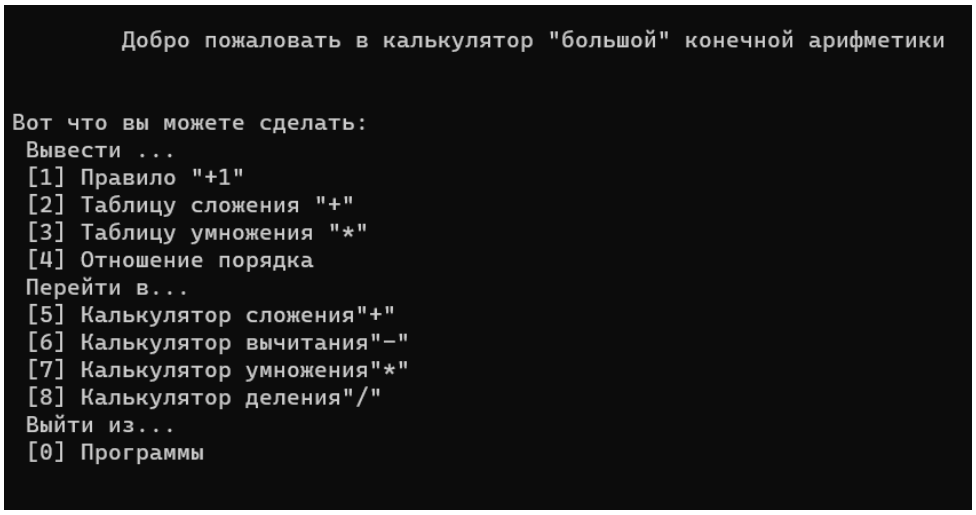

Рис. 2. Главное меню программы

После нажатия клавиши «1», либо «2», либо «3», либо «4», на консоль выводится таблица правила «+1», сложения с переносом по сложению, умножения с переносом по умножению и отношение порядка соответственно (рисунки 3-8)

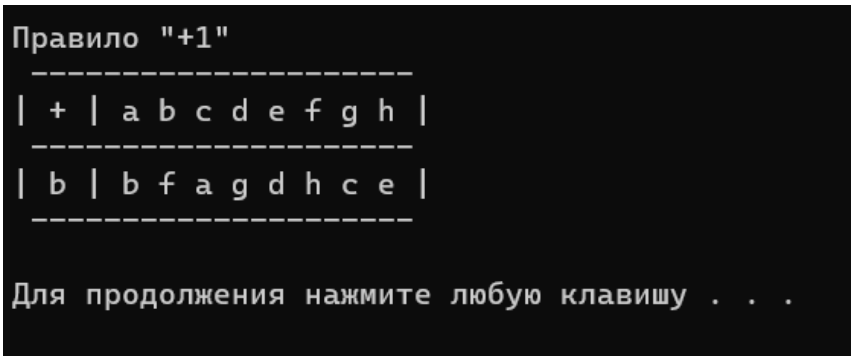

Рис. 3. Правило «+1»

Рис. 4. Таблица сложения

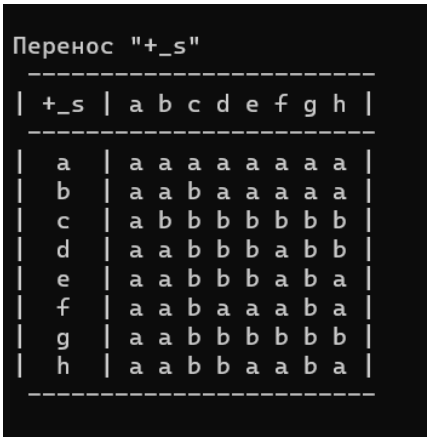

Рис. 5. Таблица переноса по сложению

Рис. 6. Таблица умножения

Рис. 7. Таблица переноса по умножению

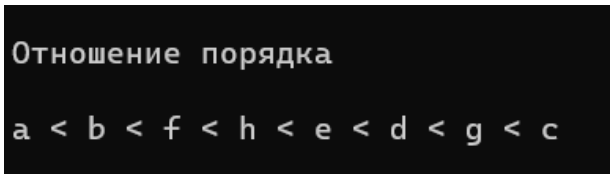

Рис. 8. Отношение порядка

После нажатия клавиши «5», либо «6», либо «7», либо «8», переход в калькулятор сложения, вычитания, умножения или деления соответственно, затем программа запрашивает числа, после корректного ввода выводится ответ. Примеры ввода и вычисления в 4 калькулятторах изображены на рисунках 9-13.

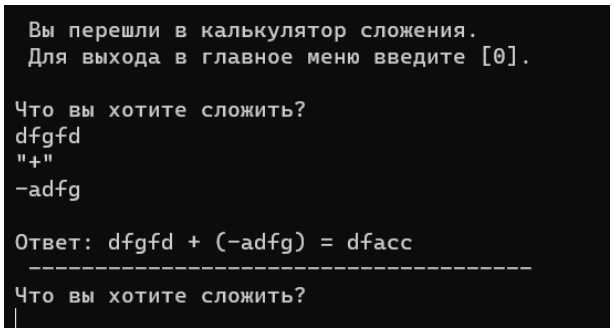

Рис. 9. Калькулятор сложения

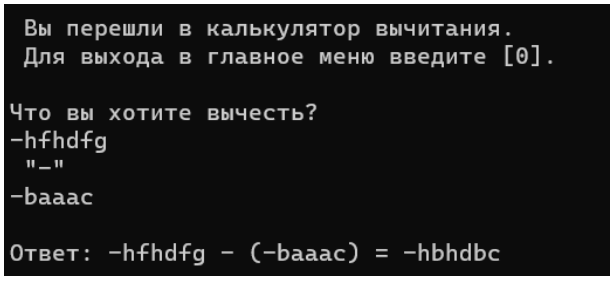

Рис. 10. Калькулятор вычитания

```
 Вы перешли в калькулятор умножения.
 Для выхода в главное меню введите [0].

Что вы хотите перемножить?
dfg
 "*"
-cbcb

Ответ: dfg * (-cbcb) = -egdfgeg
```

Рис. 11. Калькулятор умножения

```
 Вы перешли в калькулятор Деления.
 Для выхода в главное меню введите [0].

Что вы хотите поделить?
ccaaff
 "/"
fdddd

Ответ: ccaaff / fdddd = fc
       остаток = edac

Проверка остатка:
ccaaff - fdddd * fc = ccaaff - cghhbh = edac
```

Рис. 12. Калькулятор деления

Заключение

В результате выаолнения курсовой работы был реализован калькулятор для выполнения четырех арифметических операций $(+, -, *, \div)$ с «большими» конечными числами $< M_8^8; +, * >$Он основан на «малой» конечной арифметике $< M_8; +, * >$, где действует правило « $+1$» и выполняются обычные свойства арифметических операций, а именно коммутативности $(+, *)$, ассоциативности $(+, *)$, дистрибутивности $*$ относительно $+$, определены аддитивная единица «a» и мультипликативная единица «b». Кроме того, программа имеет защиту от неправильного ввода данных пользователем.

Одним из преимуществ разработанной программы является то, что с помощью объектно-ориентированного подхода класс Ar можно расширить битовыми операциями, такими как «И», «ИЛИ», «исключающее ИЛИ». Можно также использовать этот класс с любыми другими классами. Помимо этого, преимущество программы состоит в том, мало повторяется код, реализованы функции которые вызываются в программе в нескольких местах или являются общими для некоторых действий, тем самым не повторяя уже написанный код гдето в других местах. Например метод проверки пользовательского ввода или заполнение таблицы умнижения или деления. Вместо того чтобы написать две такие функции с практически идеальным фрагментом кода, используется одна функция, в которой меняются небольшие различия действий, взависимости от параметра

Одним из недостатков является то, что пришлось использовать оптимизацию деления вручную, без нее уже для шести разрядных чисел решение находилось очень долго. Поэтому если увеличивать разрядность чисел, придется самим прописывать оптимизацию, так как при увеличении разрядностей, деление будет достаточно долго считает.

В данной программе можно реализовть эффективную функцию деления, а также реализовать функции поиска НОД и НОК.

Лабораторная реализована на языке C++, в среде разработки Visual Studio 2022.

Источники

1. Дискретная математика для программистов. 3-е издание. Новиков Ф.А. СПБ:.Питер,2009.
2. Секция «Телематика». -https://tema.spbstu.ru/dismath/
3. Полубенцева М.И. C/C++. Процедурное программирование. БХВ-Петербург, 2008.