МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»

Институт прикладной математики и механики

Высшая школа прикладной математики и вычислительной физики

Секция «Телематика»

Дисциплина: Дискретная математика

Отчет по курсовой работе

«Калькулятор большой «конечной» арифметики»

Вариант № 9

Студент:

Верещагина С. Е.
группа: 3630201/90002

Проверил:

старший преподаватель,
Востров А. В.

Санкт-Петербург – 2021

Содержание

Введение

В данной курсовой работе необходимо реализовать калькулятор большой конечной арифметики для четырех действий (+,-,*,/) на основе малой конечной арифметики, где задано правило «+1» (см. Рис. 1) и выполняются свойства коммутативности (+, *), ассоциативности (+), дистрибутивности * относительно +, заданы аддитивная еденица «0» и мультипликативная еденица «1», x*0=0.

№	Z_i	«+1»							
		0	1	2	3	4	5	6	7
9	Z_8	1	3	0	6	2	4	7	5

Рис. 1: Правило «+1» для варианта №9

1 Математическое описание

1.1 Основные определения

Множество M вместе с $\sum = \phi_1, ..., \phi_m, .. : M^{n_i} \to M$, где n_i - арность операции ϕ называется *алгебраичной структурой* или просто *алгеброй*. Множество M называется носителем, а $\sum$ - сигнатурой.
Кольцо - это множество M с двумя бинарными операциями $+$ и * (сложение и умножение соответственно), в котором:
$a, b, c \in M$

- $(a + b) + c = a + (b + c)$
- $0 \in M(a(a + 0 = 0 + a = a))$
- $a + b = b + a$
- $a * (b * c) = (a * b) * c$
- $a * (b + c) = (a * c) + (a * c), (a + b) * c = (a * c) + (b * c)$

Если в кольце умножение коммутативно, то такое кольцо называется *коммутативным*. Если в кольце существует нейтральный элемент по умножению, то такое кольцо называется *кольцом с еденицей*.
«Малая» конечная арифметика - это коммутативное кольцо $< Z_8; +; * >$. $Z_8 = 0, 1, 2, 3, 4, 5, 6, 7$ с еденицей, на котором определены операции сложения, умножения.
«Большая» конечная арифметика также является коммутативным кольцом $< Z_8^8; +; * >$ с еденицей, на котором определены операции вычитания и деления с остатком.
В большой конечной арифметике сигнатура определена посредством операций малой конечной арифметики и операций переполнения.[1]

1.2 Свойства операций

Бинарная операция $*(+)$ называется *коммутативной*, если ее результат не зависит от перестановки операндов, то есть:

$$x * y = y * x, \forall x, y \in M$$
$$x + y = y + x, \forall x, y \in M$$

Бинарная операция $*(+)$ называется *ассоциативной*, если:

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

Бинарная операция $*$ *дистрибутивна* относительно бинарной операции $+$, если для любых трех элементов x, y, z выполнено:

$$x * (y + z) = (x * y) + (x * z)$$
$$(y + z) * x = (y * x) + (z * x)$$

Аддитивная еденица «0»:

$$\forall x \in M x + 0 = 0 + x = x$$

Мультипликативная еденица «1»:

$$\forall x \in M x * 0 = 0 * x = 0$$

Операция сложения в «большой» конечной арифметике - коммутативная и ассоциативная операция с аддитивной еденицей. Она реализована как поразрядное сложение в малой конечной арифметике с переносом еденицы в следующий разряд при появлении нового разряда.
Операция умножения в «большой» конечной арифметике - коммутативная с мультипликативной еденицей. Она реализована аналогично сложению, но перенос при переполнении может быть больше еденицы, кроме того, необходимо учесть перенос по сложению.[3]
Для «малой» конечной арифметики отношение строгого порядка строится по правилу:

$$0 < 0 + 1 < 0 + 1 + 1 < ... < 0 + 1 + 1, ..., +1$$

Отношение строгого порядка для заданного правила «$+1$»:

$$0 \prec 1 \prec 3 \prec 6 \prec 7 \prec 5 \prec 4 \prec 2$$

1.3 Примеры вычисления выражений

Примеры вычисления выражений:

1. Сложение: $2 + 4 = 2 + 1 + 5 = 0 + 5 = 5$
2. Умножение: $3 * 3 = (1 + 1) * 3 = 3 + 3 = 7$

1.4 Построение таблиц

1. Таблицы для «малой» конечной арифметики:

+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	3	0	6	2	4	7	5
2	2	0	4	1	5	7	3	6
3	3	6	1	7	0	2	5	4
4	4	2	5	0	7	6	1	3
5	5	4	7	2	6	3	0	1
6	6	7	3	5	1	0	4	2
7	7	5	6	4	3	1	2	0

Рис. 2: Таблица операции сложения малой конечной арифметики

*	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	1	4	3	6	5	7
3	0	3	4	7	7	3	4	0
4	0	4	3	7	7	4	3	0
5	0	5	6	3	4	1	2	7
6	0	6	5	4	3	2	1	7
7	0	7	7	0	0	7	7	0

Рис. 3: Таблица операции сложения малой конечной арифметики

2. Для перехода к большой конечной арифметике необходимы таблицы переполнения по сложению и умножению:

+s	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0
2	0	1	1	1	1	1	1	1
3	0	0	1	0	1	0	0	0
4	0	0	1	1	1	1	1	1
5	0	0	1	0	1	1	1	1
6	0	0	1	0	1	1	0	0
7	0	0	1	0	1	1	0	1

Рис. 4: Таблица переполнения по сложению

*s	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	4	1	5	7	3	6
3	0	0	1	0	1	1	0	1
4	0	0	5	1	7	6	3	6
5	0	0	7	1	6	6	1	3
6	0	0	3	0	3	1	1	1
7	0	0	6	1	6	3	1	3

Рис. 5: Таблица переполнения по умножению

2 Особенности реализации

Алгоритм программы содержит в себе построение таблиц (в виде двумерных массивов, по строго заданному правилу «+1») сложения, умножения, переноса по сложению и переноса по умножению.
Программа выполняет стандартные бинарные операции (+, -, *, /). На вход берется два числа и одна из данных операций.
Программа поддерживает работу с отрицательными числами.

2.1 Структура данных

Программа включает в себя следующие поля данных:

N - размер заданной арифметики;
int rez - переменная для хранения результата;
const int razr - переменная для хранения разрядности;
int RulePlusOne[N] - двумерный массив с малой «конечной» арифметикой с заданным правилом «+1»;
int Plus[N][N] - двумерный массив, содержащий таблицу по сложению;
int PlusP[N][N] - двумерный массив, содержащий таблицу переноса по сложению;
int Mult[N][N] - двумерный массив, содержащий таблицу по умножению;
int MultP[N][N] - двумерный массив, содержащий таблицу переноса по умножению.

2.2 Реализация построения таблиц сложения, умножения и переполнения

Таблицы малой и большой конечной арифметики по сложению и умножению заданы при помощи двумерных массивов.
Функция SetPlus() генерирует таблицы по сложению и переносу по сложению и записывает результат в переменные Plus[][] и PlusP[][]. Заполнение таблицы сложения и его дополнения идет одновременно.

```
void SetPlus()
{
for (int i = 0; i < N; i++)
{
Plus[0][i] = Plus[i][0] = i;
Plus[1][i] = Plus[i][1] = Rule[i];
if (Rule[i] == 0) PlusP[1][i] = PlusP[i][1] = 1;
}
for (int i = 2; i < N; i++)
for (int k = i; k < N; k++)
{
PlusP[i][k] = PlusP[k][i] = (Count[i] + Count[k]) / N;
Plus[i][k] = Plus[k][i] = Order[(Count[i] + Count[k]) \% N];
}
}
```

Функция SetMult() генерирует таблицы по сложению и переносу по сложению и записывает результат в переменные Mult[][] и MultP[][]. Заполнение таблицы умножения и его дополнения идет одновременно.

```
void SetMult()
7
{
for (int i = 0; i < N; i++)
Mult[1][i] = Mult[i][1] = i;
for (int i = 2; i < N; i++)
for (int k = i; k < N; k++)
{
int p = 0;
int t = i;
for (int e = 0; e < Count[k] - 1; e++)
{
if (PlusP[i][t]) p = Plus[p][1];
t = Plus[i][t];
}
Mult[i][k] = Mult[k][i] = t;
MultP[i][k] = MultP[k][i] = p;
}
}
```

2.3 Реализация сложения и вычитания

- Функция Sum() реализует сложение двух чисел.

  На вход функция получает два числа. Сложение происходит поразрядно, начиная с младшего разряда, по таблице сложения. К результату прибавляется дополнение суммы предыдущего разряда (изначально равное 0). Само дополнение вычисляется как сумма дополнений чисел данного и предыдущего разряда.

```
int Sum(int A, int B)
{
int n = 1;
int res = 0;
int over = 0;
int over1 = 0;
bool neg = 0;
if (((A >= 0) && (B >= 0)) || ((A <= 0) && (B <= 0)))
{
if ((A < 0) && (B < 0))
{
neg = 1;
A *= -1;
B *= -1;
}
while (A || B)
{
int t = Plus[A \% 10][B \% 10];
over1 = Plus[PlusP[t][over]][PlusP[A \% 10][B \% 10]];
t = Plus[t][over];
over = over1;
res = res + n * t;
n *= 10;
A = A / 10;
B = B / 10;
}
res = res + n * over;
if (neg) res *= -1;
}
else if (A < 0) res = Sub(B, -1 * A);
else if (B < 0) res = Sub(A, -1 * B);
return res;
}
```

- Функция Sub() реализует вычитание двух чисел.

  На вход функция получает два числа. Если числа одинакового знака, то функция определяет знак и записывает флаг в переменную neg (1, если числа отрицательные, 0, если числа положительные). Далее функция производит вычитание и результат домножает на -1 если neg = 1.
  Если числа разного знака, то вызывается функция сложения, в которую передается два этих числа, причем отрицательное число домножается на (-1). Если отрицательным было первое число, то результат тоже домножается на (-1).
  Если обнаруживается наличие переполнения, программа выводит в консоль сообщение.

```
int Sub(int A, int B)
{
if ((A > 0) && (B < 0)) return Sum(A, -1 * B);
else if ((A < 0) && (B > 0)) return (-1 * Sum(-1 * A, B));
else if ((A < 0) && (B < 0))
{
int tmp = A;
A = abs(B);
B = abs(tmp);
}
int n = 1;
int t, res = 0;
bool loan = 0, neg = 0;
if (Cmp(A, B) == 0)
{
int tmp = A;
A = B;
B = tmp;
neg = 1;
}
while (A || B)
{
if (loan)
{
t = Count[A \% 10] - 1 - Count[B \% 10];
loan = 0;
}
else t = Count[A \% 10] - Count[B \% 10];
if (t < 0)
{
loan = 1;
t = Order[N + t];
}
else t = Order[t];
res = res + n * t;
n *= 10;
A = A / 10;
B = B / 10;
}
if (neg) res *= -1;
return res;
}
```

2.4 Реализация умножения

Функция Mul() реализует умножение двух чисел, получая на вход два числа типа int. Если числа разных знаков, то функция записывает в переменную neg еденицу. Умножение происходит поразрядно по модулю с помощью таблиц умножения и переноса по умножению. Полученный результат прибавляется к

локальной переменной, которая возвращается в конце функции. Если neg = 1, то возвратное значение домножается на (-1). Если обнаруживается наличие переполнения, программа выводит в консоль сообщение.

```
int Mul(int A, int B)
{
int n = 1;
int res = 0;
int over = 0;
bool neg = 0;
if ((A < 0) && (B < 0)) neg = 0;
else if ((A > 0) && (B < 0)) neg = 1;
else if ((A < 0) && (B > 0)) neg = 1;
A = abs(A);
B = abs(B);
while (B)
{
int tmp = 0;
int ntmp = 1;
int Atmp = A;
while (Atmp)
{
int t = Mult[Atmp \% 10][B \% 10] + MultP[Atmp \% 10][B \% 10] * 10;
tmp = Sum(tmp, t * ntmp);
Atmp = Atmp / 10;
ntmp *= 10;
}
B = B / 10;
res = Sum(res, tmp * n);
n *= 10;
}
if (neg) res *= -1;
return res;
}
```

2.5 Реализация деления

Функция Div() реализует деление двух чисел На вход функция получает два числа (первое - делимое, второе - делитель). Прописаны все частные случаи деления. Если первое число меньше второго - то сразу выводится результат деления 0, остаток - первое число целиком. Если пользователь вводит деление 0 на 0, выводится сообщение: "Неопределенность. Любое число в промежутке [-222222222;222222222],"т.е. интервал со всеми вариантами. В интервале число 2, потому что оно максимальное в данной арифметике. Если пользователь вводит ноль только в качестве делителя, то выводится следующее системное сообщение: "На ноль делить нельзя! (искл. 0/0)". Само деление реализовано через вычитание. Пока делимое больше делителя, мы вычитаем из первого второе и записываем в делимое. К временному результату добавляем единицу. После выхода из цикла, когда делимое меньше делителя, делимое записывается в остаток.

```
int Div(int A, int B)
{
if ((A == 0) && (B == 0)) return (razr * Order[N - 1] + 1);
else if (B == 0) return (razr * Order[N - 1] + 2);
else if (A == 0)
{
rem = 0;
return 0;
}
rem = 0;
int n = 1;
```

```
int res = 0;
int ost = 0;
bool one = 0;
bool neg = 0;
if ((A < 0) && (B < 0)) neg = 0;
else if ((A > 0) && (B < 0)) neg = 1;
else if ((A < 0) && (B > 0)) { neg = 1; one = 1; }
A = abs(A);
B = abs(B);
A = Sub(A, B);
while (A >= 0)
{
res = Sum(res, 1);
rem = A;
A = Sub(A, B);
}
if (one)
13
{
res = Sum(res, 1);
rem = abs(A);
}
if (neg) res *= -1;
return res;
}
```

2.6 Реализация сравнения

На вход функция принимает два числа, которые необходимо сравнить. Числа сравниваются по каждому разряду, начиная с меньшего. Для этого две цифры одного разряда раскладываются на сумму единиц, и сравнивается их количество. Если больше у первого числа - возвращается истина, в противном случае - ложь.

```
bool Cmp(int A, int B)  // > == 1
{
bool f = 0;
int n = 1;
for (int i = 0; i < N - 1; i++)
n *= 10;
for (int i = n; i > 0; i /= 10)
{
int d = A / i;
if (Count[A / i] > Count[B / i])
{
f = 1;
break;
}
else if (Count[A / i] < Count[B / i])
{
f = 0;
break;
}
A = A % i;
B = B % i;
}
return f;
}
```

3 Результат работы программы

На Рис. 6 представлены результаты сложения, вычитания, умножения и деления. На Рис. 7 представлены обработка некорректного пользовательского ввода, а так же крайние случаи для деления (деление на 0 и 0/0) и переполнение. Программа работает верно.

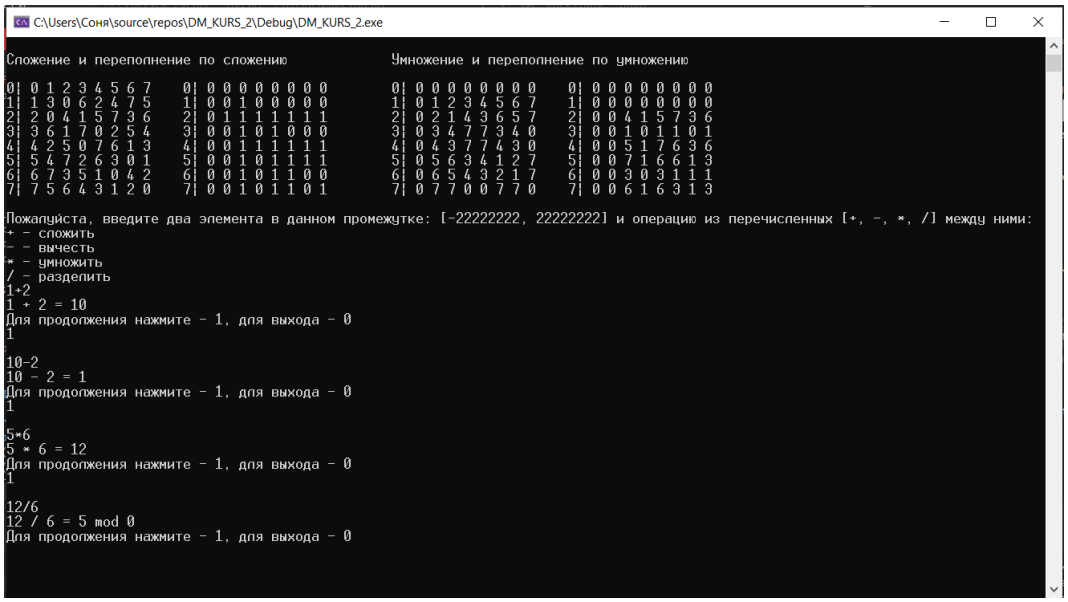

Рис. 6: Результаты сложения, вычитания, умножения и деления

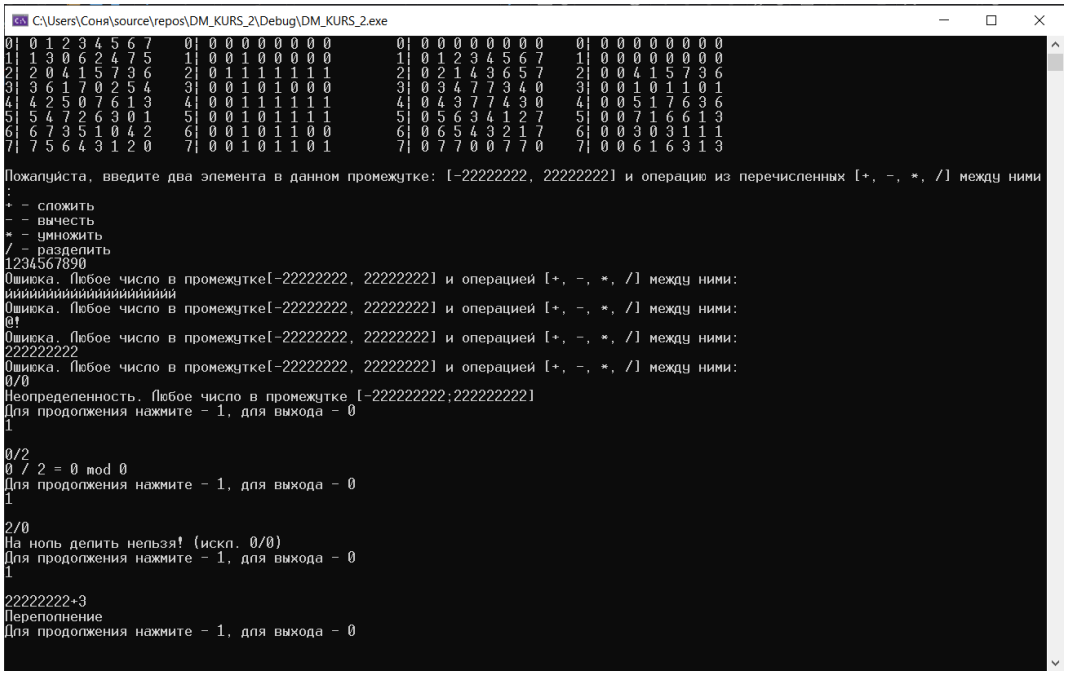

Рис. 7: Обработка некорректного пользовательского ввода, деление на 0 и 0/0, переполнение

Заключение

В ходе выполнения данной работы был сформирован калькулятор большой «конечной» арифметики $< Z_8; +; * >$ для четырех действий $(+, -, *, /)$ на основе малой «конечной» арифметики, с заданным правилом, и выполняются свойства коммутативности $(+, *)$, ассоциативности $(+)$, дистрибутивности $*$ относительно $+$, заданы аддитивная единица «0» и мультипликативная единица «1», $* 0 = 0$. Калькулятор поддерживает работу с отрицательными числами, защищен от некорректного ввода данных. Достоинства программы:

- минимальное использование сторонних библиотек;
- код программы можно легко приспособить для любой конечной арифметики.

Недостатки программы:

- возможность проводить действия только над двумя операндами;
- возможность работать только с числами состоящих из цифр;
- вычисление результата только для одного действия.

Программу можно масштабировать следующим образом: можно сделать так, чтобы пользователь сам вводил необходимую ему конечную арифметику, а так же добавить операцию возведения в степень.

Список литературы

[1] Новиков Ф. А. «Дискретная математика для программистов: Учебник для вузов». Санкт-Петербург: Питер, 2008 г. – 384 стр.

[2] Полубенцева М. И. «C/C++. Процедурное программирование». Санкт-Петербург: БХВ, 2015 г. – 448 стр.

[3] Востров А. В. Лекции по курсу «Дискретная математика».