МИНОБРНАУКИ РОССИИ

«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 о выполнении курсовой работы
Дискретная математика
Калькулятор «большой» конечной арифметики
Вариант №14

Студент,
группа 5130201/30002 _____ Овчинников Т. А.

Преподаватель

_____ Востров А. В.

«_____»_____ 2024г.

Санкт-Петербург, 2024

Содержание

Введение

Задача курсовой работы заключается в разработке калькулятора «большой» конечной арифметики $< Z_8^8; +, * >$ для четырех действий $(+, -, *, \div)$ 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, поиск НОД и НОК для двух чисел.

Правило «+1» задано в Таблице 1.

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

Таблица 1. Правило «+1»

1 Математическое описание

1.1 Алгебраические структуры

Множество M вместе с набором операций $\Sigma = \{\varphi_1, \ldots, \varphi_m\}$, $\varphi_i : M^{n_i} \to M$, где n_i — арность операции φ_i, называется алгебраической структурой, универсальной алгеброй или просто алгеброй

Коммутативное кольцо с единицей — это алгебраическая структура $< M; +, * >$, в которой:

1. $(a + b) + c = a + (b + c)$
2. $\exists 0 \in M \ (a + 0 = 0 + a = a)$
3. $\forall a \in M \ (a + (-a) = 0)$
4. $a + b = b + a$
5. $(a * b) * c = a * (b * c)$
6. $a * (b + c) = a * b + a * c$
7. $a * b = b * a$
8. $\exists 1 \in M \ (a * 1 = 1 * a = a)$

Малая конечная арифметика — конечное коммутативное кольцо с единицей $< M_i; +, * >$, на котором определены операции вычитания «-» и деления «÷», но деление определено частично.

Большая конечная арифметика — конечное коммутативное кольцо с единицей $< M_i^n; +, * >$, на котором определены операции переноса, вычитания «-» и деления «÷» , причем деление определено с остатком.

В данной работе большая конечная арифметика имеет вид $< M_8^8; +, * >$

1.2 Таблицы операций

Правило для «+» можно увидеть в Таблице 2, для «$+_S$» в Таблице 3, для «*» в Таблице 4 и для «$*_S$» в Таблице 5.

Таблица 2. Правило «+»

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

Таблица 3. Правило «$+_S$»

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

Таблица 4. Правило «$*$»

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

Таблица 5. Правило «$*_S$»

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

Также введем отношение строгого линейного порядка в малой конечной арифметике, которое определяется правилом «$+1$»:

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

1.3 Примеры вычислений для построения таблиц

1. $e + c = e + (e + b) = e + (b + b) + b = f$
2. $f + e = f + b + b = d$
3. $c * e = c * (b + b) = c * b + c * b = c + c = c + e + b = c + b + b + b = h$

4. $g * a = a$

2 Особенности реализации программы

В данном разделе будут рассмотренны спроектированные классы, реализующие работу программы.

Работа выполнялась с использованием графического фреймворка Qt для C++.

2.1 SymbolNumber

SymbolNumber - класс реализущий операнды символьных операций.

SymbolNumber имеет одно поле: строка QString с символьным числом внутри. Также для реализации вычислений используются таблицы сложения и умножения, а также таблицы сдвигов, они определены в коде как глобальные объекты QVector<QVector<QChar>>. Помимо таблиц в методах данного класса используется отношение порядка и перевод из символа в колонку/строку таблиц и обратно, они определены как глобальные объекты QHash. Код заголовочного файла данного класса можно увидеть в «Листинг 1».

Листинг 1. Класс SymbolNumber

```
QHash<QChar,int> rowCol = {{'a', 0},
                           {'b', 1},
                           {'c', 2},
                           {'d', 3},
                           {'e', 4},
                           {'f', 5},
                           {'g', 6},
                           {'h', 7}};

QHash<int,QChar> rowSymb = {{0, 'a'},
                            {1, 'b'},
                            {2, 'c'},
                            {3, 'd'},
                            {4, 'e'},
                            {5, 'f'},
                            {6, 'g'},
                            {7, 'h'}};

QHash<QChar,int> poradok = {{'a', 0},
                            {'b', 1},
                            {'e', 2},
                            {'c', 3},
                            {'g', 4},
                            {'f', 5},
                            {'h', 6},
                            {'d', 7}};


QVector<QVector<QChar>> SumTable =
{{'a','b','c','d','e','f','g','h'},
```

```
    {'b','e','g','a','c','h','f','d'},
    {'c','g','h','e','f','a','d','b'},
    {'d','a','e','h','b','g','c','f'},
    {'e','c','f','b','g','d','h','a'},
    {'f','h','a','g','d','e','b','c'},
    {'g','f','d','c','h','b','a','e'},
    {'h','d','b','f','a','c','e','g'}};

    QVector<QVector<QChar>> MulTable =
     {{'a','a','a','a','a','a','a','a'},
    {'a','b','c','d','e','f','g','h'},
    {'a','c','b','f','h','d','g','e'},
    {'a','d','f','b','h','c','g','e'},
    {'a','e','h','h','g','e','a','g'},
    {'a','f','d','c','e','b','g','h'},
    {'a','g','g','g','a','g','a','a'},
    {'a','h','e','e','g','h','a','g'}};

    QVector<QVector<QChar>> SumSTable=
    {{'a','a','a','a','a','a','a','a'},
    {'a','a','a','b','a','a','a','a'},
    {'a','a','a','b','a','b','a','b'},
    {'a','b','b','b','b','b','b','b'},
    {'a','a','a','b','a','a','a','b'},
    {'a','a','b','b','a','b','b','b'},
    {'a','a','a','b','a','b','b','b'},
    {'a','a','b','b','b','b','b','b'}};

    QVector<QVector<QChar>> MulSTable =
    {{'a','a','a','a','a','a','a','a'},
    {'a','a','a','a','a','a','a','a'},
    {'a','a','b','e','a','b','b','e'},
    {'a','a','e','h','b','g','c','f'},
    {'a','a','a','b','a','b','b','b'},
    {'a','a','b','g','b','c','e','g'},
    {'a','a','b','c','b','e','e','c'},
    {'a','a','e','f','b','g','c','g'}};


    class SymbolNumber
    {
      QString number;

      QString sum(const QString& l, const QString& r);
      QString sub(const QString& l, const QString& r);
      QString mul(const QString& l, const QString& r);
      QString divPoloj(const QString& l, const QString& r);
      QString divOtric(const QString& l, const QString& r);
```

```
79      public:
80      SymbolNumber();
81      SymbolNumber(const QString& num);
82
83      const QString getNum() const;
84      SymbolNumber pow(const SymbolNumber& another) ;
85      SymbolNumber rem(SymbolNumber& another) ;
86
87      SymbolNumber operator+(const SymbolNumber& another) ;
88      SymbolNumber operator-(const SymbolNumber& another) ;
89      SymbolNumber operator*(const SymbolNumber& another) ;
90      SymbolNumber operator/(const SymbolNumber& another) ;
91
92  };
```

2.1.1 Конструктор класса SymbolNumber

Вход: строка QString с введенным пользователем числом
Выход: сформированный объект, готовый к работе

Внутри конструктора класса SymbolNumber удаление незначащих нулей, также избавление от лишнего - при вводе числа «-a». Код данного метода можно увидеть в «Листинг 2».

Листинг 2. Конструктор SymbolNumber

```
1   SymbolNumber::SymbolNumber(const QString &num)
2   {
3     number = num;
4
5     if(number.length() == 0) number = "a";
6
7     if(num[0] == '-'){
8       while(number.length() > 2 && number[1] == 'a'){
9         number.remove(1,1);
10      }
11    } else {
12      while(number.length() > 1 && number[0] == 'a'){
13        number.removeFirst();
14      }
15    }
16
17    if(number == "-a") number = "a";
18  }
```

2.1.2 sum

Вход: две строки QString с левым и правым слагаемыми

Выход: строка QString с результатом сложения

Метод sum выполняет поразрядное суммирование двух символьных чисел путем прохода строк с числами от первого до последнего разряда, высчитывая на каждом шаге сумму двух символов на соответсвующих позициях, также сохраняя сумму смещения. Код данного метода можно увидеть в «Листинг 3».

Листинг 3. sum

```
1  QString SymbolNumber::sum(const QString& l, const QString& r)
2  {
3    QString result;
4    QChar ostatok = 'a';
5
6    int lenTh = l.length(), lenAn = r.length();
7
8    while(lenTh-1 >= 0 || lenAn-1 >= 0){
9      QChar leftOperand = (lenTh > 0) ? l[lenTh-1] : 'a';
10     QChar rightOperand = (lenAn > 0) ? r[lenAn-1] : 'a';
11
12     QChar bezOstatka = sumResult(leftOperand, rightOperand);
13
14     result.prepend(sumResult(ostatok, bezOstatka));
15
16     ostatok = sumSResult(ostatok, bezOstatka);
17     ostatok = sumResult(ostatok,
18      sumSResult(leftOperand, rightOperand));
19
20     lenAn--;
21     lenTh--;
22   }
23
24   if(ostatok!='a') result.prepend(ostatok);
25
26   while(result[0] == 'a' && result.length() > 1){
27     result.removeFirst();
28   }
29
30   return result;
31 }
```

2.1.3 sub

Вход: две строки QString с левым и правым операндами
Выход: строка QString с результатом вычитания

Метод sub выполняет поразрядное вычитание двух символьных чисел путем прохода строк с числами от первого до последнего разряда, высчитывая на каждом шаге разность двух символов на соответсвующих позициях, также при необходимости

занимая единицу у большего разряда. Если левый операнд меньше правого метод меняет их местами и записывает результат с минусом . Код данного метода можно увидеть в «Листинг 4».

Листинг 4. sub

```
QString SymbolNumber::sub(const QString& l, const QString& r)
{
  QString result;
  QString left = l;
  QString right = r;

  bool zaim = 0, otric = 0;

  int lenTh = left.length(), lenAn = right.length();

  if(lenTh < lenAn || (lenTh==lenAn && checkLess(l, r))){
    otric = 1;
    left = r;
    right = l;
    lenTh = lenAn;
    lenAn = l.length();
  }

  while(lenTh-1 >= 0 || lenAn-1 >= 0){
    QChar leftOperand = (lenTh > 0) ? left[lenTh-1] : 'a';
    QChar rightOperand = (lenAn > 0) ? right[lenAn-1] : 'a';
    QChar newOperand;

    if(zaim){
      newOperand = subResult(leftOperand, 'b');
    } else{
      newOperand = leftOperand;
    }

    if(checkLess(leftOperand, rightOperand)){
      zaim = 1;
    } else {
      zaim = 0;
    }

    result.prepend(subResult(newOperand, rightOperand));

    lenAn--;
    lenTh--;
  }

  while(result[0] == 'a'&& result.length() > 1){
    result.removeFirst();
```

```
44      }
45
46      if(otric) result.prepend('-');
47
48      return result;
49    }
```

2.1.4 mul

Вход: две строки QString с левым и правым множителями
Выход: строка QString с результатом умножения

Метод mul выполняет поразрядное вычитание умножение символьных чисел путем прохода строк с числами от первого до последнего разряда, высчитывая на каждом шаге результат умноженя двух символов на соответсвующих позициях, также учитвая смещение умножения. Код данного метода можно увидеть в «Листинг 5».

Листинг 5. mul

```
1   QString SymbolNumber::mul(const QString &l, const QString &r)
2   {
3     QString result = "a";
4
5     int lenTh = l.length(), lenAn = r.length();
6
7     for (int i = lenAn-1; i >= 0; i--){
8       QString temp;
9       QChar ost = 'a', leftOperand = r[i], bezOst, rightOperand;
10      for(int j = lenTh-1; j >= 0; j--){
11        rightOperand = l[j];
12        bezOst = mulResult(leftOperand, rightOperand);
13
14        temp.prepend(sumResult(bezOst, ost));
15
16        ost = sumSResult(bezOst, ost);
17        ost = sumResult(ost, mulSResult(leftOperand,
18         rightOperand));
19      }
20      if(ost!='a') temp.prepend(ost);
21      temp.append(QString((lenAn-i-1),'a'));
22      result = sum(result, temp);
23    }
24
25    return result;
26  }
```

2.1.5 divPoloj

Вход: две строки QString с делимыми и делителем
Выход: строка QString с положительным результатом деления

Метод divPoloj выполняет поиск подходящего множетеля, на который нужно умножить делитель, чтобы получить делимое. Процесс останавливается до того, когда при увелечении множителя на единицу результат умножения станет больше делителя .Код данного метода можно увидеть в «Листинг 6».

Листинг 6. divPoloj

```
1  QString SymbolNumber::divPoloj(const QString &l,
2  const QString &r)
3  {
4    if(l == "a") return "a";
5
6    QString mnojitel = "a";
7
8    QString resUmnoj = r;
9
10   while(checkLess(resUmnoj, l)){
11     mnojitel = sum(mnojitel, "b");
12     resUmnoj = sum(resUmnoj, r);
13   }
14
15   if(resUmnoj == l) mnojitel = sum(mnojitel, "b");
16
17   return mnojitel;
18 }
```

2.1.6 divOtric

Вход: две строки QString с делимыми и делителем
Выход: строка QString с отрицательным результатом деления

Метод divOtric выполняет поиск подходящего множетеля, на который нужно умножить делитель, чтобы получить делимое. Процесс останавливается после того, когда при увелечении множителя на единицу результат умножения станет больше делителя. Код данного метода можно увидеть в «Листинг 7».

Листинг 7. divOtric

```
1  QString SymbolNumber::divOtric(const QString &l,
2  const QString &r)
3  {
4    if(l == "a") return "a";
5
6    QString mnojitel = "a";
7
```

```
8       QString resUmnoj = r;
9
10      while(checkLess(resUmnoj, l)){
11        mnojitel = sum(mnojitel, "b");
12        resUmnoj = sum(resUmnoj, r);
13      }
14
15      mnojitel = sum(mnojitel, "b");
16
17      if(checkLess(resUmnoj, l)) mnojitel = sum(mnojitel, "b");
18
19      return mnojitel;
20    }
```

2.1.7 Оператор +

Вход: два объекта SymbolNumber с двумя слагаемыми
Выход: объект SymbolNumber с результатом суммы

Оператор + определяет знаки слагаемых и в зависимости от них отправляет символьные числа в соответствующий метод. Код данного оператора можно увидеть в «Листинг 8».

Листинг 8. Оператор +

```
1   SymbolNumber SymbolNumber::operator+
2   (const SymbolNumber &another)
3   {
4     QString left = number, right = another.number;
5
6     if(left[0] == '-' && right[0] == '-'){
7       left.removeFirst();
8       right.removeFirst();
9       return sum(left,right).prepend('-');
10    } else if(left[0] == '-'){
11      left.removeFirst();
12      return sub(right, left);
13    } else if(right[0] == '-'){
14      right.removeFirst();
15      return sub(left, right);
16    }
17
18    return sum(left,right);
19  }
```

2.1.8 Оператор -

Вход: два объекта SymbolNumber с двумя операндами

Выход: объект SymbolNumber с результатом разности

Оператор + определяет знаки операндов и в зависимости от них отправляет символьные числа в соответствующий метод. Код данного оператора можно увидеть в «Листинг 9».

Листинг 9. Оператор -

```
SymbolNumber SymbolNumber::operator-
(const SymbolNumber &another)
{
  QString left = number, right = another.number;

  if(left[0] == '-' && right[0] == '-'){
    left.removeFirst();
    right.removeFirst();
    return sub(right,left);
  } else if(left[0] == '-'){
    left.removeFirst();
    return sum(left,right).prepend('-');
  } else if(right[0] == '-'){
    right.removeFirst();
    return sum(left, right);
  }

  return sub(left, right);
}
```

2.1.9 Оператор *

Вход: два объекта SymbolNumber с двумя множителями
Выход: объект SymbolNumber с результатом умножения

Оператор * определяет знаки множителей и в зависимости от них определяет знак результата умножения, после чего отправляет символьные числа в метод mul. Код данного оператора можно увидеть в «Листинг 10».

Листинг 10. Оператор *

```
SymbolNumber SymbolNumber::operator*
(const SymbolNumber &another)
{
  QString left = number, right = another.number;

  if(left[0] == '-' && right[0] == '-'){
    left.removeFirst();
    right.removeFirst();
    return mul(left,right);
  } else if(left[0] == '-'){
    left.removeFirst();
```

```
12        return mul(right, left).prepend('-');
13      } else if(right[0] == '-'){
14        right.removeFirst();
15        return mul(left, right).prepend('-');
16      }
17
18      return mul(number, another.number);
19    }
```

2.1.10 Оператор /

Вход: два объекта SymbolNumber с делимым и делителем
Выход: объект SymbolNumber с результатом деления

Оператор / определяет знаки делимого и делителя и в зависимости от них определяет знак результата деления, после чего отправляет символьные числа в метод divPoloj либо divOtric. Также оператор контролирует деление на 0. Код данного оператора можно увидеть в «Листинг 11».

Листинг 11. Оператор /

```
1   SymbolNumber SymbolNumber::operator/
2   (const SymbolNumber &another)
3   {
4     QString left = number, right = another.number;
5
6     if(left == "a" && right == "a"){
7       return SymbolNumber("[-dddddddd; dddddddd]");
8     } else if(right == "a"){
9       return SymbolNumber("ц0");
10    }
11
12    if(left[0] == '-' && right[0] == '-'){
13      left.removeFirst();
14      right.removeFirst();
15      return divPoloj(left,right);
16    } else if(left[0] == '-'){
17      left.removeFirst();
18      return divOtric(left, right).prepend('-');
19    } else if(right[0] == '-'){
20      right.removeFirst();
21      return divOtric(left, right).prepend('-');
22    }
23    return divPoloj(left, right);
24  }
```

2.1.11 sumResult

Вход: два символа QChar с левым и правым слагаемыми
Выход: символ QChar с результатом сложения без остатка

Функция sumResult с помощью таблицы SumTable определяет результат сложения двух символов без остатка. Код данной функции можно увидеть в «Листинг 12».

Листинг 12. sumResult

```
1  QChar sumResult(const QChar& left, const QChar& right){
2    return SumTable[rowCol[left]][rowCol[right]];
3  }
```

2.1.12 sumSResult

Вход: два символа QChar с левым и правым слагаемыми
Выход: символ QChar с отстатком от результата сложения

Функция sumSResult с помощью таблицы SumSTable определяет остаток от результата сложения двух символов . Код данной функции можно увидеть в «Листинг 13».

Листинг 13. sumSResult

```
1  QChar sumSResult(const QChar& left, const QChar& right){
2    return SumSTable[rowCol[left]][rowCol[right]];
3  }
```

2.1.13 mulResult

Вход: два символа QChar с левым и правым множителями
Выход: символ QChar с результатом умножения без остатка

Функция mulResult с помощью таблицы MulTable определяет результат умножения двух символов без остатка. Код данной функции можно увидеть в «Листинг 14».

Листинг 14. mulResult

```
1  QChar mulResult(const QChar& left, const QChar& right){
2    return MulTable[rowCol[left]][rowCol[right]];
3  }
```

2.1.14 mulSResult

Вход: два символа QChar с левым и правым множителями
Выход: символ QChar с отстатком от результата умножения

Функция mulSResult с помощью таблицы MulSTable определяет остаток от результата умножения двух символов . Код данной функции можно увидеть в «Листинг 15».

Листинг 15. mulSResult

```
1  QChar mulSResult(const QChar& left, const QChar& right){
2    return MulSTable[rowCol[left]][rowCol[right]];
3  }
```

2.1.15 subResult

Вход: два символа QChar с левым и правым операндами
Выход: символ QChar с результатом вычитания

Функция subResult с помощью таблицы SumTable определяет результат вычитания двух символов. Код данной функции можно увидеть в «Листинг 16».

Листинг 16. subResult

```
1  QChar subResult(const QChar& left, const QChar& right){
2    return rowSymb[SumTable[rowCol[right]].indexOf(left)];
3  }
```

2.1.16 chetnoe

Вход: строка QString с символьным числом
Выход: 1 - если число четное, 0 - если нечетное

Функция chetnoe проверяет символьное число на четность путем провеки последнего символа. Код данной функции можно увидеть в «Листинг 17».

Листинг 17. subResult

```
1    QChar subResult(const QChar& left, const QChar& right){
2      return rowSymb[SumTable[rowCol[right]].indexOf(left)];
3    }
```

2.1.17 checkLess

Вход: две строки QString с двумя символьными числами
Выход: 1 - если число слева меньше правого , 0 - в обратном случае

Функция checkLess поразрядно сравнивает числа с помощью таблицы отношения порядка poradok. Код данной функции можно увидеть в «Листинг 18».

Листинг 18. checkLess

```
1  bool checkLess(const QString& left, const QString& right){
2    if (left.length() < right.length()) {
```

```
3         return true;
4       }
5       if (left.length() > right.length()) {
6         return false;
7       }
8
9       for (int i = 0; i < left.length(); i++) {
10        if (poradok[left[i]] < poradok[right[i]]) {
11          return true;
12        } else if (poradok[left[i]] > poradok[right[i]]) {
13          return false;
14        }
15      }
16
17      return false;
18    }
```

2.2 Calculator

Calculator - класс реализущий интерфейс калькулятора. Он унаследован от класса QWidget, для того чтобы приложение работало в оконном режиме.

Calculator имеет следующие поля: 2 указателя на QLineEdit, для реализации полей ввода чисел; указатель на QComboBox, для выбора операции над введенными числами; указатель на QLabel для отображения результатов вычисления. Код заголовочного файла данного класса можно увидеть в «Листинг 19».

Листинг 19. Класс Calculator

```
1   class Calculator : public QWidget
2   {
3     Q_OBJECT
4
5     QLineEdit *input1;
6     QLineEdit *input2;
7     QComboBox *operationCombo;
8     QLabel *resultLabel;
9
10    public slots:
11    void checkInput();
12
13    void calculate();
14
15    public:
16    explicit Calculator(QWidget *parent = nullptr);
17  };
```

2.2.1 Конструктор класса Calculator

Вход: указатель на родительский элемент класса QWidget
Выход: сформированный объект, готовый к работе

Внутри конструктора класса Calculator инициализируются все объекты, находящиеся внутри класса, создаются и заполняются layout объекты, для размещения необходимых полей, кнопок и текстовых блоков и выполняется привязка сигналов кнопок и некоторых объектов к соответствующим слотам данного класса.. Код данного метода можно увидеть в «Листинг 20».

Листинг 20. Конструктор Calculator

```
1  Calculator::Calculator(QWidget *parent)
2  : QWidget{parent}
3  {
4    setWindowTitle("Калькулятор большойконечнойарифметики  ");
5
6    input1 = new QLineEdit(this);
7    input1->setPlaceholderText("a");
8    input2 = new QLineEdit(this);
9    input2->setPlaceholderText("a");
10   operationCombo = new QComboBox(this);
11   QPushButton *calculateButton = new QPushButton("Рассчитать",
12    this);
13   resultLabel = new QLabel("Результат: ", this);
14   QLabel* mapText = new QLabel("Порядок символовдля ( проверки):
15   \na:0 , b:1 , e:2 , c:3 , g:4 , f:5 , h:6 , d:7");
16   QFrame *line = new QFrame();
17   line->setFrameShape(QFrame::HLine);
18   line->setFrameShadow(QFrame::Sunken);
19
20   operationCombo->addItem("+");
21   operationCombo->addItem("-");
22   operationCombo->addItem("*");
23   operationCombo->addItem("/");
24   operationCombo->addItem("^");
25
26   QVBoxLayout *mainLayout = new QVBoxLayout(this);
27   QHBoxLayout *inputLayout = new QHBoxLayout();
28   inputLayout->addWidget(input1);
29   inputLayout->addWidget(operationCombo);
30   inputLayout->addWidget(input2);
31
32   mainLayout->addLayout(inputLayout);
33   mainLayout->addWidget(calculateButton);
34   mainLayout->addWidget(resultLabel);
35   mainLayout->addWidget(line);
36   mainLayout->addWidget(mapText);
37
```

```
38      connect(calculateButton, &QPushButton::clicked, this,
39      &Calculator::calculate);
40      connect(input1, &QLineEdit::textChanged, this,
41      &Calculator::checkInput);
42      connect(input2, &QLineEdit::textChanged, this,
43      &Calculator::checkInput);
44    }
```

2.2.2 Слот checkInput

Вход: 2 указателя на QLineEdit
Выход: QMessageError при ошибке ввода

Слот checkInput вызывается при изменении одного из объектов QLineEdit и проверяет ввод пользователя на количество знаков и лишние символы. Код данного метода можно увидеть в «Листинг 21».

Листинг 21. Слот checkInput

```
1  void Calculator::checkInput()
2  {
3    QString numbers = "abcdefhg";
4
5    QString in1 = input1->text().remove(' ');
6    input1->setText(in1);
7    QString in2 = input2->text().remove(' ');
8    input2->setText(in2);
9
10   if(in1.length() > 0){
11     if((in1[0]!='-'&&in1.length() > 8) ||
12      (in1[0]=='-'&&in1.length() > 9)){
13       QMessageBox::warning(this, "Ошибка",
14        "Слишком длинноечисло !");
15       in1.removeLast();
16       input1->setText(in1);
17       return;
18     }
19   }
20
21   if(in2.length()>0){
22     if((in2[0]!='-'&&in2.length() > 8) ||
23      (in2[0]=='-'&&in2.length() > 9)){
24       QMessageBox::warning(this, "Ошибка",
25        "Слишком длинноечисло !");
26       in2.removeLast();
27       input2->setText(in2);
28       return;
29     }
30   }
```

```
31    for(int i = 0; i < in1.length(); i++){
32      if(!numbers.contains(in1[i])){
33        if(i==0&&in1[i]==QChar('-')) continue;
34        QString errorText("Недопустимый символ {");
35          errorText.append(in1[i]);
36          errorText.append("} влевомоперанде  .\nСписокдопустимых
37            : a,b,c,d,e,h,g,f,- если( вначале ).");
38        QMessageBox::warning(this, "Ошибка", errorText);
39        input1->setText(in1.remove(i,1));
40      }
41    }
42
43    for(int i = 0; i < in2.length(); i++){
44      if(!numbers.contains(in2[i])){
45        if(i==0&&in2[i]==QChar('-')) continue;
46        QString errorText("Недопустимый символ {");
47          errorText.append(in2[i]);
48          errorText.append("} вправомоперанде  .\nСписокдопустимых
49            : a,b,c,d,e,h,g,f,- если( вначале ).");
50        QMessageBox::warning(this, "Ошибка", errorText);
51        input2->setText(in2.remove(i,1));
52      }
53    }
54  }
```

2.2.3 Слот calculate

Вход: 2 указателя на QLineEdit и указатель на QComboBox
Выход: результат вычисления

Слот calculate вызывается при нажатии на кнопку «Рассчитать». Данный слот берет числа введенные пользователем в объекты QLineEdit, приводит их к объектам SymbolNumber, производит выбранное вычисление и меняет QLabel с результатом. Код данного метода можно увидеть в «Листинг 22».

Листинг 22. Слот calculate

```
1  void Calculator::calculate()
2  {
3    QString left = input1->text();
4    QString right = input2->text();
5
6    SymbolNumber symbNum1(left);
7    SymbolNumber symbNum2(right);
8
9    SymbolNumber symbRes;
10   SymbolNumber remainder;
11
12   QString operation = operationCombo->currentText();
```

```
13
14    if (operation == "+") {
15      symbRes = symbNum1+symbNum2;
16    } else if (operation == "-") {
17      symbRes = symbNum1-symbNum2;
18    } else if (operation == "*") {
19      symbRes = symbNum1*symbNum2;
20    } else if (operation == "/") {
21      symbRes = symbNum1 / symbNum2;
22      remainder = symbNum1.rem(symbNum2);
23      if(symbRes.getNum() != "0" && symbRes.getNum() !=
24       "[-dddddddd; dddddddd]"){
25        resultLabel->setText("Результат: " + symbRes.getNum() +
26         '\n' + "Остаток: " + remainder.getNum());
27      } else {
28        resultLabel->setText("Результат: " + symbRes.getNum());
29      }
30      return;
31    } else if (operation == "^") {
32      symbRes = symbNum1.pow(symbNum2);
33    } else {
34      QMessageBox::warning(this, "Ошибка", "Выберите операцию!");
35      return;
36    }
37
38    if((symbRes.getNum()[0]!='-'&&symbRes.getNum().length() > 8)
39    || (symbRes.getNum()[0]=='-'&&symbRes.getNum().length() > 9)){
40      QMessageBox::warning(this, "Ошибка",
41      "Результат операциивыходитзаграницыдопустимыхзначений      :\n
42      [-dddddddd; dddddddd]");
43      return;
44    }
45
46
47    resultLabel->setText("Результат: " + symbRes.getNum());
48  }
```

3 Результаты работы программы

После запуска программы пользователь видит начальное меню (Рис.1).

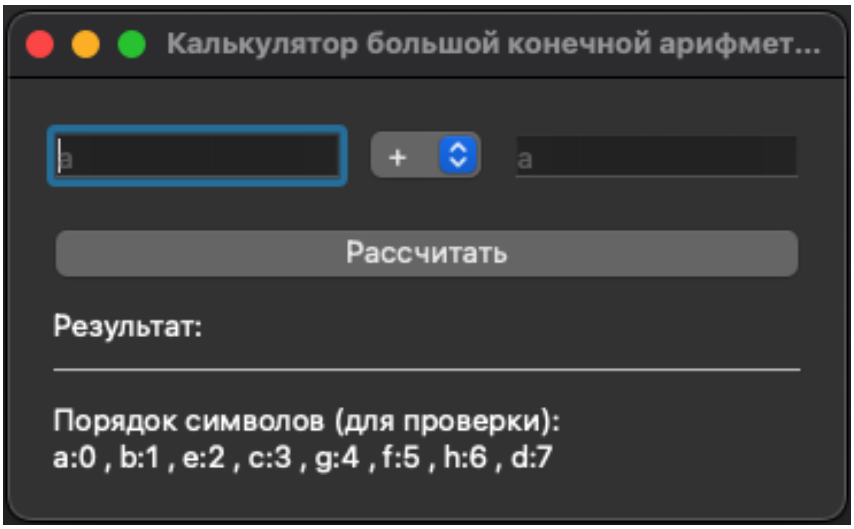

Рис. 1. Начальное меню

При нажатии на кнопку выбора операции пользователь видит список операций (Рис.2).

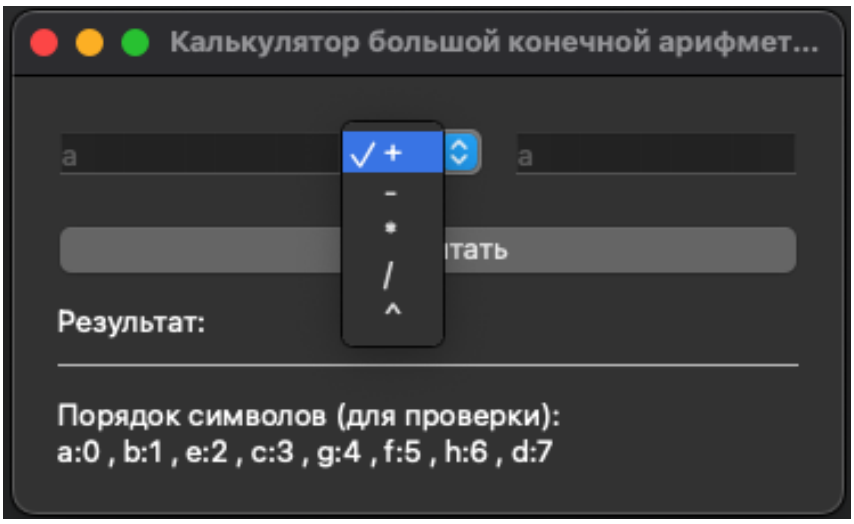

Рис. 2. Список операций

При вводе некоторых значений, выборе операции и нажатии на кнопку «Рассчитать» в приложении появиться текст в поле «Результат: »(Рис.3).

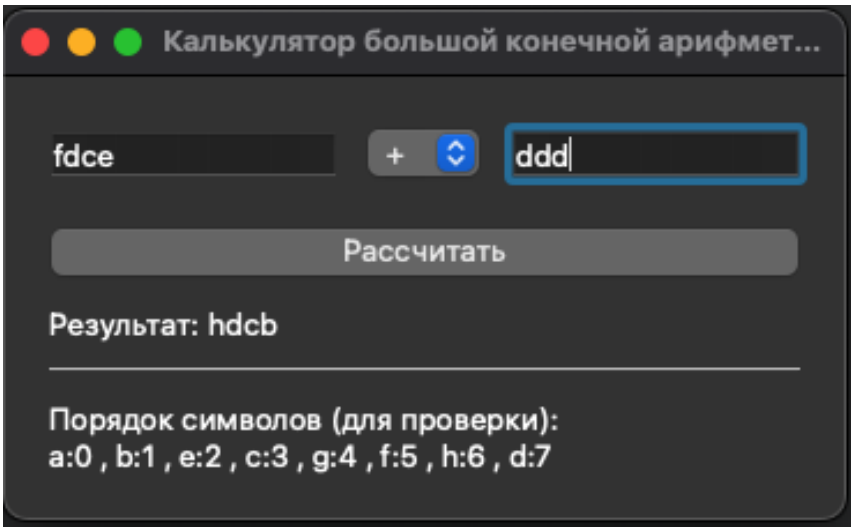

Рис. 3. Результат вычисления

При вводе некорректных символов, всплывет окно, сообщающее об ошибке и некорректный символ удаляется (Рис.4).

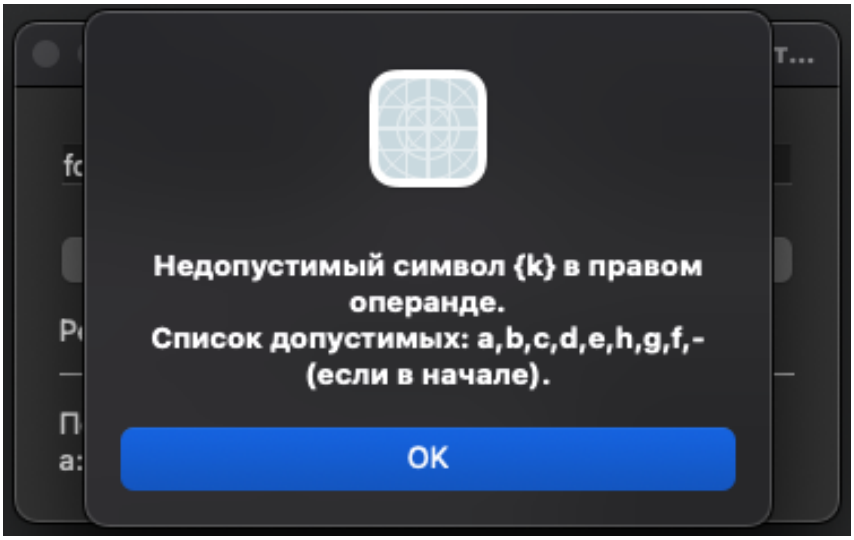

Рис. 4. Некорректный символ

При вводе числа, выходящего за границы допустимых значений, пользователь увидит окно, сообщающее о том, что операнд выходит за границы допустимых значений, после этого число обрежеться справа до допустимого значения (Рис.5).

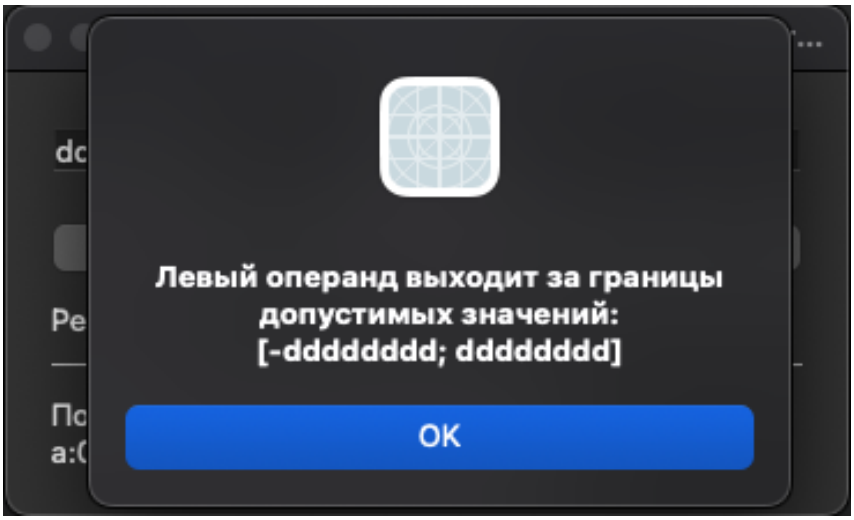

Рис. 5. Операнд выходит за границы допустимых значений

При попытке вычисления операции, результат который выходит за пределы допустимых значений пользователь получить сообщение о том, что результат выходих за границы допустимых значений (Рис.6).

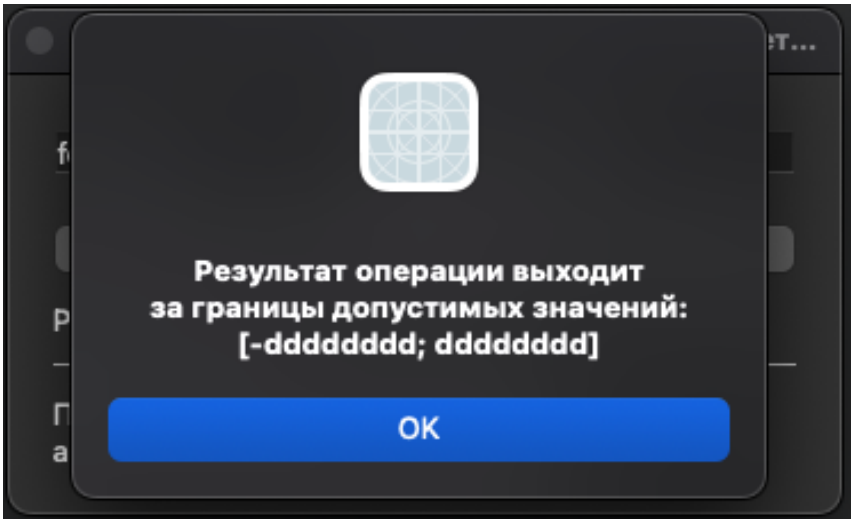

Рис. 6. Результат выходит за границы допустимых значений

При делении на нулевой элемент любого числа, кроме нулевого элемента, пользователь увидит в результате пустое множество (Рис.7).

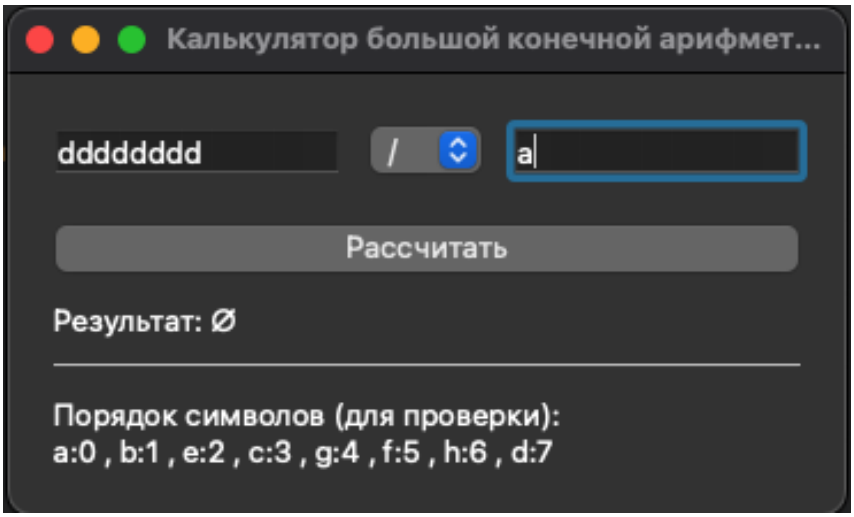

Рис. 7. Деление на нулевой элемент

При делении нулевой элемент на нулевой элемент пользователь увидит в результате множество всех возможных значений (Рис.8).

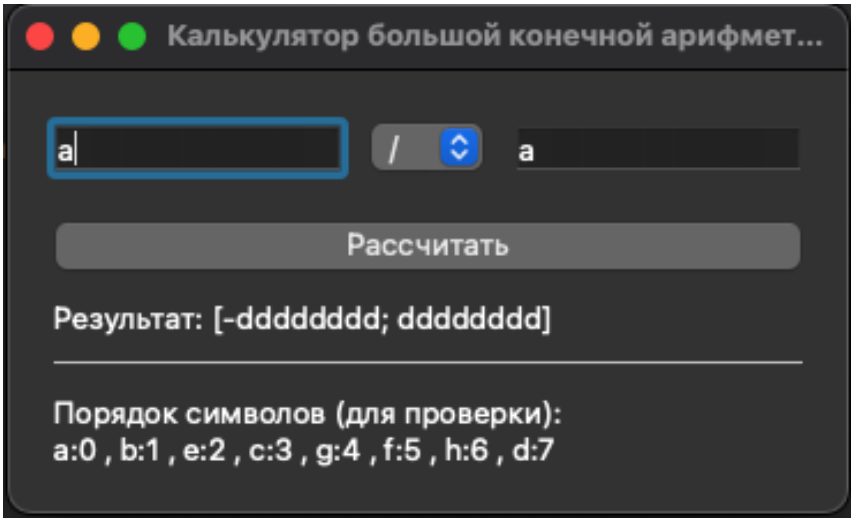

Рис. 8. Деление нулевой элемент на нулевой элемент

Заключение

В результате выполнения курсовой работы был реализован калькулятора «большой» конечной арифметики $<Z_8^8; +, *>$ для четырех действий $(+, -, *, \div)$. Спроектированный калькулятор может выполнять следующие действия: сложение, вычитание, умножение, деление, возведение в степень. Программа контролирует некорректный ввод, выходы за границы допустимых значений и деление на нулевой элемент.

Достоинства программы:

- Готовые классы фрейморка Qt гарантируют быструю и эффективную работу интерфейса;
- Графический интерфейс позволяет удобнее вычислять новые результаты операций;
- Разделение вычислений и контроля результатов дают возможность быстро масштабировать калькулятор до больших разрядностей;

Недостатки программы:

- Сильная привязанность к конкретным символам;

Масштабирование:в программу можно добавить сохранение предыдущих результатов вычисления, возможность добавлять переменные для того чтобы считать формулы с несколькими действиями сразу, а также функционал для управления порядком операций.

Список использованной литературы

[1] Секция "Телематика"/ текст : электронный / — URL: https://tema.spbstu.ru/dismath/ (Дата обращения 08.12.2024)

[2] Qt Documentation / текст : электронный / — URL: https://doc.qt.io/ (Дата обращения 08.12.2024)

[3] Новиков Ф. А. *Дискретная математика для программистов*. 3-е изд. — Санкт-Петербург: Питер Пресс, 2009. — 384 с.