Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «ФУНДАМЕНТАЛЬНЫЕ НАУКИ»

КАФЕДРА «ПРИКЛАДНАЯ МАТЕМАТИКА»

Лабораторная работа № 3

по дисциплине «Типы и структуры данных»

Тема Обратная польская запись

Студент Лямин И.С.

Группа ФН12-31Б

Преподаватели Волкова Л.Л.

Москва, 2024

Содержание

ВВЕДЕНИЕ

В данной лабораторной работе будет реализован алгоритм перевода арифметического выражение из инфиксной записи в постфиксную и последующее вычисление значения выражения. Цель работы — реализовать алгоритм перевода из инфиксной записи в постфиксную, реализовать функцию вычисления значения выражения из постфиксной записи.

Для достижения цели необходимо выполнить следующие задачи:

1) разобрать суть основных понятий используемых для преобразовании и вычислении
2) разработать алгоритмы валидации, токенизации, проверки, перевода и вычисления
3) реализовать алгоритмы
4) провести тестирование, проверить работоспособность реализаций алгоритмов

1 Аналитическая часть

1.1 Инфиксная и постфиксная нотация

Инфиксная нотация – это стандартная форма записи математических выражений, в которой операторы располагаются между операндами. Например, выражение `A + B` представляет операцию сложения двух переменных A и B. Инфиксная нотация требует явного указания порядка операций с использованием скобок.

Это форма записи математических выражений, где операторы следуют за операндами. Например, выражение `A B +` в постфиксной нотации также представляет операцию сложения. Эта форма не требует скобок для определения порядка операций, что упрощает процесс вычисления.

1.2 Преимущество постфиксной нотации

Главное преимущество постфиксной нотации (обратной польской записи) заключается в том, что при её разборе нет необходимости расставления скобок и в связи с этим такую запись легко представить в виде бинарного дерева, а следовательно и составить компьютерный алгоритм. Рассмотрим, на рисунке 1.1, пример: 1,2,+,2,3,*,/.

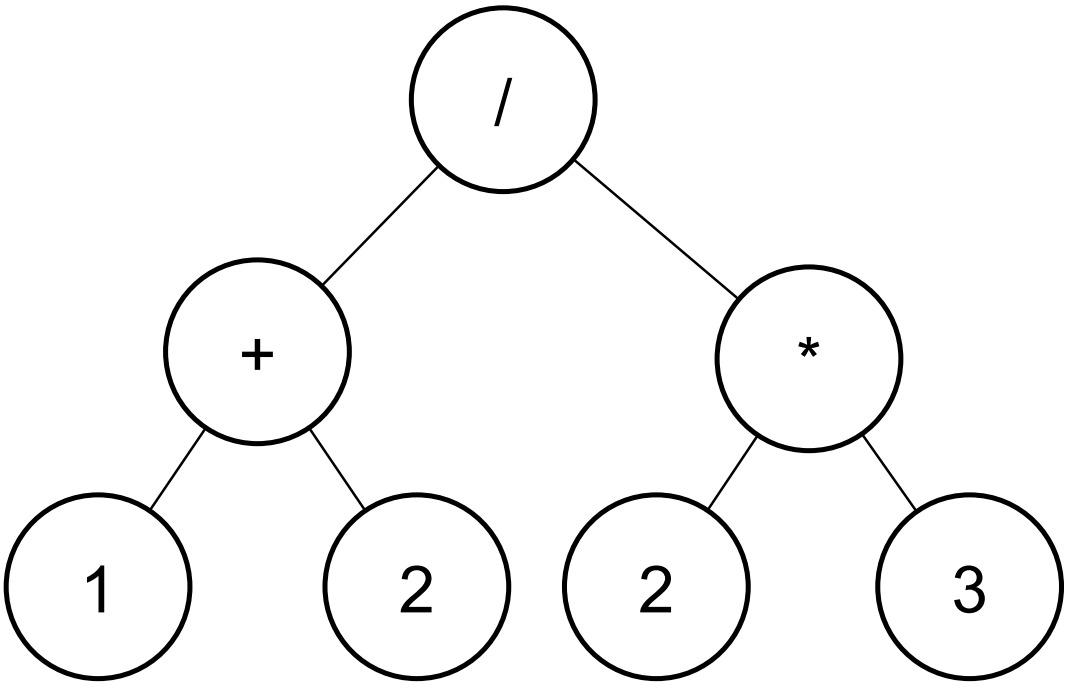

Рисунок 1.1 — Дерево постфиксной записи

2 Конструкторская часть

2.1 Вспомогательные функции

Для программной реализации алгоритма сначала опишем вспомогательные функции

1) `is_num(char elem)`
   - — тип возвращаемого значения: bool,
   - — описание: определяет, принадлежит ли данный элемент набору ’123456789x.’,
2) `is_oper(char elem)`
   - — тип возвращаемого значения: bool,
   - — описание: определяет, принадлежит ли данный элемент набору ’+-/*^’,
3) `is_bark(char elem)`
   - — тип возвращаемого значения: bool,
   - — описание: определяет, принадлежит ли данный элемент набору ’()’,
4) `ord(string elem)`
   - — тип возвращаемого значения: int,
   - — описание: определяет порядок выполнения операции при инфиксной записи в соответствии со следующим словарём,

Листинг 2.1 — словарь с порядками операций

```
map<string, unsigned> operators{{"-u", 2},
  {"cos", 4},
  {"sin", 4},
  {"tg", 4},
  {"ctg", 4},
  {"acos", 4},
  {"asin", 4},
  {"atg", 4},
  {"actg", 4},
  {"^", 3},
  {"*", 2},
  { "/", 2 },
  { "+", 1 },
  { "-", 1 }
};
```

5) `arithm(flaot num_1, flaot num_2, string operator)`
   - — тип возвращаемого значения: float,
   - — описание: находит значение получаемое в результате применения заданного бинарного оператора (operator) к двум операндам (num_1, num_2),

6) `arithm(flaot num, string operator)`

— тип возвращаемого значения: float,

— описание: находит значение получаемое в результате применения заданного унарного оператора (operator) к одному операнду (num),

Для взаимодействия со стеком разработаны следующие функции

1) `IsEmpty`

— описание: проверяет, пуст ли стек,

— возвращает: `true`, если стек пуст; в противном случае `false`,

2) `IsFull`

— описание: проверяет, полон ли стек,

— возвращает: `true`, если стек полон; в противном случае `false`,

3) `Push`

— описание: добавляет новый элемент на верх стека,

4) `Pop`

— описание: удаляет верхний элемент из стека и сохраняет его в параметре по ссылке,

— возвращает: `true`, если операция успешна (т. е. стек не пуст); в противном случае `false`,

5) `Peek`

— описание: извлекает верхний элемент стека без его удаления, сохраняя его в параметре по ссылке,

— возвращает: `true`, если операция успешна (т. е. стек не пуст); в противном случае `false`,

6) `print`

— описание: отображает элементы стека,

2.2 Алгоритм валидации строки

Алгоритм валидации строки направлен на выявление невалидных символов, которые вызовут ошибку в последующей реализации программы. Результат работы этого алгоритма это строка проверенная по следующим критериям

1) отсутствие неизвестных функций,
2) правильная расстановка скобок,
3) правильная расстановка операторов,
4) перевод обычного минуса в бинарный при отсутствии левого операнда,
5) явное деление на ноль,

Этапы работы алгоритма

1) перевод строки в нижний регистр и проверка скобок
   — переводим строку в нижний регистр функцией lower(), встроенной в стандартную библиотеку c++,
   — для отслеживания правильности расстановок скобок с помощью переменной i с изначальным значением 0 проходимся по всей строке через цикл и при наличии открывающейся строки прибавляем 1 к i, а в случае закрывающейся вычитаем 1. Если на протяжении всего цикла значение i ни разу не опускалось ниже 0 и к концу цикла оно равно 0, то в строке скобки расставлены правильно,
2) проверка на правильность введённых функций,
3) проверка на корректность расстановки операторов,

2.3 Алгоритм токенизации строки

Задача данного алгоритма составить массив с данными формата `string`, который содержит операторы, операнды и числа в том порядке, в котором они были поданы пользователем (в виде инфиксной записи).

Поля и начальная настройка

1) `str`
   — тип: `string`,
   — описание: изначальная строка, содержащая все элементы введённые пользователем,
2) `arr`
   — тип: `string*`,
   — описание: результирующий массив, в который добавляются элементы арифметического выражения,
3) `len`
   — тип: `int`
   — описание: количество элементов в изначальной строке,
4) `cur`
   — тип: `string`,
   — описание: локальный элемент сохраняющий терм(токен) и добавляющий его в результирующий массив при выполнении определённых условий, после этого он обнуляется,
5) `ind_arr`
   — тип: `int`
   — описание: Счетчик для определения длины результирующего массива,

Этапы работы алгоритма

Функция `create_arr` последовательно обрабатывает элементы входной строки `str`, выполняя следующие действия(i — индикатор итерации).

1) из строки `str` извлекается первый элемент и сохраняется в переменной `cur`,
   - если первый элемент `str` — оператор
     - если элемент — $'-'$ то `arr[ind] == "u-"`,
     - иначе `arr[ind_arr] = cur`,

     строка `cur` обнуляется , к индикатору `ind_arr` прибавляется 1,
   - если текущий элемент `str` — число
     - если следующий элемент тоже число, то переходим к следующей итерации,
     - иначе `arr[ind_arr] = cur`, прибавляем 1 к `ind_arr` и очищаем `cur`,
   - если текущий элемент `str` — скобка,
     - `arr[ind_arr] = cur`, прибавляем 1 к `ind_arr` и очищаем `cur`,
   - иначе
     - если следующий элемент открывающаяся скобка или текущий элемент последний, то присваиваем `arr[ind_arr] = cur`, прибавляем 1 к `ind_arr` и очищаем `cur`,
     - иначе продолжаем итерацию,
2) цикл продолжается до того момента как не будут перебраны все элементы троки,
3) функция возвращает словарь `arr` с токенами,

2.4 Алгоритм перевода из инфиксной записи в постфиксную

Для преобразования строки, содержащей инфиксную запись математического выражения, в постфиксную (обратную польскую) запись разработана функция `create_postf_arr`. Алгоритм будет использовать стек и контейнер для хранения промежуточных результатов. Ниже приведено описание всех частей алгоритма.

Поля и начальная настройка

1) `str`
   - тип: `string`,
   - описание: входящая строка содержащая инфиксную запись,
2) `stack`
   - тип: `stack<string>`,
   - описание: пустой стек,
3) `postf_arr`
   - тип: `string*`,

— описание: хранит результат в постфиксной записи, который будет сформирован и возвращен функцией,

4) `cur`

— тип: индекс обозреваемого элемента (на нуле),

Этапы работы алгоритма

Алгоритм выполняет обработку каждого элемента инфиксного выражения по очереди до того момента, пока курсор находится в пределе в пределах входной строки`str`.

1) если элемент под курсором число, то поместить его в ответ `posf_arr`,

2) если под курсором открывающаяся скобка `(`, она помещается в стек `stack`,

3) если под курсором закрывающаяся скобка `)`, из стека `a` извлекаются элементы и добавляются в `postf_arr` до тех пор, пока не встретится открывающая скобка `(`, либо пока не `stack` не будет пуст,

4) если `s` является оператором или тригонометрической функцией ,

4.1) если стек `stack` пуст, оператор помещается в стек,

4.2) если верхний знак в стеке это открывающая скобка, `(`, оператор помещается в `stack`,

4.3) если приоритет оператора под курсором выше или равен приоритету оператора на вершине стека (проверяется с помощью функции `ord`), оператор помещается в `stack`,

4.4) в противном случае оператор из вершины стека извлекается и добавляется в `postf_arr`, после чего алгоритм возвращаться к пункту 4,

5) после окончания цикла, все элементы из стека `stack` помещаются в ответ `postf_arr`,

Результат

Функция возвращает динамический массив `postf_arr`, содержащий исходное выражение в постфиксной записи.

2.5 Алгоритм вычисления выражения в постфиксной записи

Функция `calculate` разработана для вычисления значения математического выражения, заданного в строке `str`. Выражение подается в инфиксной записи и преобразуется в постфиксную запись для вычисления, используя стек для хранения промежуточных значений и операндов. Результат вычислений возвращается через ссылку `ans`.

Поля и начальная настройка

1) `postf`

— тип: `string*`,

— описание: динамический массив, содержащий элементы выражения в постфиксной записи, полученный после преобразования `str` с помощью функции `create_post_arr`,

2) `stack`
   — тип: `Stack<float>`,
   — описание: вспомогательный стек для хранения числовых значений и промежуточных результатов операций,
3) `cur`
   — тип: `int`,
   — описание: индекс обозреваемого элемента (на нуле),

Этапы работы алгоритма

Функция выполняет вычисления для каждого элемента постфиксного выражения.

1) если под курсором число то помещаем его в стек `stack`,
2) если под курсором знак операции то,
   — попытаться извлечь из стека необходимое количество аргументов. В случае неудачи, выдать ошибку и прервать цикл,
   — если удалось извлечь необходимое количество аргументов, то произвести соответствующие вычисления и поместить результат в стек,
3) конец цикла,

3 Технологическая часть

3.1 Выбор средств реализации

Для программной реализации алгоритма использовалась среда разработки Visual Studio 2022, язык программирования, на котором была выполнена реализации алгоритмов — C++. Для компиляции кода использовался компилятор MSVC. Исследование проводилось на ноутбуке (64–разрядная операционная система, процессор x64, частота процессора 3.1 ГГц, модель процессора 12th Gen Intel(R) Core(TM) i5-12500H, оперативная память 16 ГБ)

3.2 Реализация алгоритмов

В листингах 3.1, 3.2, 3.3, 3.4, 3.5 представлена программная реализация описанных классов и функций.

Листинг 3.1 — Программная реализация стека и его интерфейса

```
template<typename tp>
struct Stack {
  int capacity = 0;
  int col_elem = 0;
  tp* array = nullptr;
  int end = 0;
  int start = 0;

  Stack(int capacity) {
    this->capacity = capacity;
    array = new tp[capacity];
    for (int i = 0; i < capacity; i++) {
      this->array[i] = tp();
    }
  }
};

template<typename tp>
bool IsEmpty(Stack<tp>* ds) {
  if (ds->col_elem == 0) return true;
  return false;
}

template<typename tp>
bool Pop(Stack<tp>* stack, tp* contain) {
  if (IsEmpty(stack)) return false;
```

```
27      *contain = stack->array[stack->end];
28      stack->array[stack->end] = tp();
29      stack->end -= 1;
30      //cout << "Number " << (*contain) << " extracted from stack" <<
            endl;
31      stack->col_elem -= 1;
32      return true;
33    }
34
35    template<typename DS>
36    bool IsFull(DS* ds) {
37      if (ds->capacity == ds->col_elem) return true;
38      return false;
39    }
40
41    template<typename tp>
42    bool Push(Stack<tp>* st, tp elem) {
43      if (IsFull(st)) {
44        cout << "Array is full" << endl;
45        return 0;
46      }
47      for (int i = 0; i < st->capacity; i++) {
48        if (st->array[i] == tp()) {
49          if (IsEmpty(st)) {
50            //cout << "Stack was empty" << endl;
51            st->col_elem += 1;
52            st->end = i;
53            st->array[i] = elem;
54            return 1;
55          }
56          //cout << "Stack was NOT empty" << endl;
57          st->col_elem += 1;
58          st->end = i;
59          st->array[i] = elem;
60          return 1;
61        }
62      }
63      return 0;
64    }
65
66    float Peek(Stack<float>* st) {
```

```
    return st->array[st->end];
  }

  int Peek(Stack<int>* st) {
    return st->array[st->end];
  }

  string Peek(Stack<string>* st) {
    return st->array[st->end];
  }

  char Peek(Stack<char>* st) {
    return st->array[st->end];
  }

  template<typename tp>
  void print(Stack<tp>* st) {
    tp el;
    for (int i = 0; i < (st->end + 1); i++) {
      el = *(st->array[i]);
      cout << "[" << i + 1 << "]" << " --- " << el << endl;
    }
    cout << st->col_elem << " --- elemnts" << endl;
  }

  template<typename tp>
  bool Del(Stack<tp>* stack) {
    if (IsEmpty(stack)) return false;
    stack->array[stack->end] = tp();
    stack->end -= 1;
    stack->col_elem -= 1;
    return true;
  }
```

Листинг 3.2 — Программная реализация функции валидации строки

```
bool check_valid(string* str) {

  // lower
  int len = (*str).size();
  for (int i = 0; i < len; i++) {
    (*str)[i] = tolower((*str)[i]);
  }
```

```
8
9   // checking the brackets
10  int count = 0;
11  for (int i = 0; i < len; i++) {
12    if ((*str)[i] == '(') count += 1;
13    if ((*str)[i] == ')') count -= 1;
14    if (count < 0) return false;
15  }
16
17
18  string tg = "tg";
19  string _3[4] = { "sin", "cos", "ctg", "atg" };
20  string _4[3] = { "asin", "acos", "actg" };
21
22
23  for (int i = 0; i < len; i++) {
24    char elem = (*str)[i];
25    if (is_num(elem)) {
26      if (i == 0) {
27        if (!((is_num((*str)[1]) or is_oper((*str)[1])))) {
28          if (((*str)[1] == '(') or ((*str)[1] == '.')) continue;
29          cout << "Unable symbol next to the first number\nTry again\n
                ";
30          return false;
31        }
32      }
33      else if (i == (len - 1)) {
34        if (!((is_num((*str)[len - 2]) or is_oper((*str)[len - 2])))) {
35          if (((*str)[len - 2] == '(') or ((*str)[len - 2] == '.'))
                continue;//////////
36          cout << "Unable symbol next to the last number\nTry again\n";
37          return false;
38        }
39      }
40      else {
41        if (!((is_num((*str)[i - 1]) or is_oper((*str)[i - 1])) or ((*
              str)[i - 1] == '(')) or (!((is_num((*str)[i + 1]) or is_oper
              ((*str)[i + 1]) or ((*str)[i + 1] == ')'))))) {
42          if ((*str)[i - 1] == '.' or (*str)[i + 1] == '.') continue;
43          cout << "Unable symbol next to the number\nTry again\n";
44          return false;
```

```
45          }
46          continue;
47        }
48      }
49      else if (is_oper(elem)) {
50
51        if ((i + 1) >= len) {
52          cout << "!!!'" << elem << "' can't be the last symbol!!!\nTry
                again\n";
53          return false;
54        }
55
56        if ((i + 1) < len) {
57          if (is_oper((*str)[i + 1])) {
58            cout << "!!!check the the binary operator repetition!!!\n End
                  try again\n";
59            return false;
60          }
61        }
62
63        if (elem == '-') {
64          if ((i + 1) >= len) {
65            cout << "!!!'-' can't be the last symbol!!!\nTry again\n";
66            return false;
67          }
68          continue;
69        }
70
71        if ((i == 0)) {
72          cout << "Binary operator can't be the first\nEnd try again\n";
73          return false;
74        }
75        else {
76          if (((*str)[i - 1] == '(') or ((*str)[i + 1] == ')')) {
77            cout << "Check the correction of arguments for binary
                    operator\nEnd try again\n";
78            return false;
79          }
80          else if ((elem == '/') and ((*str)[i + 1] == '0')) {
81            cout << "division by zero\nTry again\n";
82            return false;
```

```
83          }
84        }
85
86
87      }
88      else if (is_brack(elem)) continue;
89      else {
90        bool indic = false;
91        for (const auto [oper, od] : operators)
92        if (oper[0] == elem) {
93          if ((i + oper.size()) > len) {
94            continue;
95          }
96          string cur_func = "";
97          for (int j = 0; j < oper.size(); j++) {
98            cur_func += (*str)[i + j];
99          }
100         if (!(cur_func == oper)) {
101           continue;
102         }
103         cur_func += (*str)[i + oper.size()];
104         if (cur_func != (oper + '(')) {
105           cout << "!!!You must write the '(' after functions!!!\nTry
                  again\n";
106           return false;
107         }
108         else {
109           i += (oper.size() - 1);
110           indic = true;
111           break;
112         }
113       }
114       if (indic) continue;
115       cout << "!!!Such a function does not exist!!!\nTry again\n";
116       return false;
117     }
118   }
119   return true;
120 }
```

Листинг 3.3 — Программная реализация функции токенезации

```
1 string* create_arr(string* str, int* ln) {
```

```
  int len = (*str).size();

  int ind_arr = 0;
  string* arr = new string[len];

  string cur = "";
  for (int i = 0; i < len; i++) {
    cur += (*str)[i];
    if (is_oper((*str)[i])) {
      if (i == 0) arr[ind_arr] = "-u";
      else if ((i > 0) and ((*str)[i - 1] == '(')) arr[ind_arr] = "-u";
      else arr[ind_arr] = cur;
      cur = "";
      ind_arr++;
    }
    else if (is_num((*str)[i])) {
      if (len > (i + 1)) {
        if (!(is_num((*str)[i + 1]))) {
          arr[ind_arr] = cur;
          ind_arr++;
          cur = "";
        }
      }
      else {
        arr[ind_arr] = cur;
        ind_arr++;
        cur = "";
      }
    }
    else if (is_brack((*str)[i])) {
      arr[ind_arr] = cur;
      ind_arr++;
      cur = "";
    }
    else {
      if (len > (i + 1)) {
        if ((*str)[i + 1] == '(') {
          arr[ind_arr] = cur;
          ind_arr++;
          cur = "";
        }
```

```
      }
      else {
        arr[ind_arr] = cur;
        ind_arr++;
        cur = "";
      }
    }
  }
  (*ln) = ind_arr;
  return arr;
}
```

Листинг 3.4 — Программная реализация функции превода в постфиксную запись

```
string* create_postf_arr(string* st_arr, int *len) {
  Stack<string> stack(*len);
  string* postf_arr = new string[(*len)];

  int ind = 0;

  for (int i = 0; i < (*len); i++) {// i - cursor
    //1.
    if (is_num(st_arr[i])) {
      postf_arr[ind] = st_arr[i];
      ind++;
    }
    //2.
    else if ((st_arr[i]) == "(") {
      Push(&stack, st_arr[i]);
    }
    //3.
    else if ((st_arr[i]) == ")") {
      while (!(IsEmpty(&stack)) and (Peek(&stack) != "(")) {
        Pop(&stack, &((postf_arr)[ind]));
        ind++;
      }
      if (Peek(&stack) == "(") {
        Del(&stack);
      }
      //3.2
      else if (IsEmpty(&stack)) {
        break;
      }
```

```
    }
    //4.
    else {
      while (true) {
        //4.1
        if (IsEmpty(&stack)) {
          Push(&stack, st_arr[i]);
          break;
        }
        //4.2
        else if ((Peek(&stack) == "(")) {
          Push(&stack, st_arr[i]);
          break;
        }
        //4.3
        else if (ord(st_arr[i]) > ord(Peek(&stack))) {
          Push(&stack, st_arr[i]);
          break;
        }
        //4.4
        else {
          Pop(&stack, &((postf_arr)[ind]));
          ind++;
        }
      }
    }
  }

  while (!(IsEmpty(&stack))) {
    Pop(&stack, &((postf_arr)[ind]));
    ind++;
  }
  (*len) = ind;
  return postf_arr;
}
```

Листинг 3.5 — Программная реализация функции вычисления выражений в постфиксной записи

```
void calculate(string s, float left, float right, float step) {
  while ((left > right) || (step <= 0)){
    cout << "Enter correct dates" << endl;
    cout << "LEFT: ";
    cin >> left;
```

```
    cout << "RIGHT: ";
    cin >> right;
    cout << "STEP: ";
    cin >> step;
  }
  cin.ignore();
  cout << "Enter the arithmetic expression: ";
  getline(std::cin, s);
  string s_read;
  for (int i = 0; i < s.size(); i++) {
    if (s[i] != ' ') s_read += s[i];
  }
  int ans = check_valid(&s_read);
  while (!(ans)) {
    getline(std::cin, s);
    s_read.clear();
    for (int i = 0; i < s.size(); i++) {
      if (s[i] != ' ') s_read += s[i];
    }
    ans = check_valid(&s_read);
  }

  int len;
  string* arr = create_arr(&s_read, &len);

  string* post_arr = create_postf_arr(arr, &len);
  cout << endl;
  cout << "LEN: " << len << endl;
  cout << "POSTFIX: ";
  for (int i = 0; i < len; i++) {
    cout << post_arr[i] << "_";
  }
  cout << endl;

  Stack<float> nums(len);
  cout << "Value in the interval from " << left << " to " << right << "
      with step " << step << ":";
  for (float i = left; i <= right; i += step) {
    if (i >= right) i = right;
    int ind = 0;
    for (int j = 0; j < len; j++) {
```

```
      if (is_num(post_arr[j])) {
        if (post_arr[j] == "x") {
          Push(&nums, i);
        }
        else {
          Push(&nums, stof(post_arr[j]));
        }
      }
      else {
        if (nums.col_elem < operands[post_arr[j]]) {
          cout << "!!!Insufficient number of arguments!!!";
          exit(1);
        }
        else {
          if (operands[post_arr[j]] == 1) {
            float _1;
            Pop(&nums, &_1);
            Push(&nums, arithm(_1, post_arr[j]));
          }
          else {
            float _1, _2;
            Pop(&nums, &_1);
            Pop(&nums, &_2);
            Push(&nums, arithm(_1, _2, post_arr[j]));
          }
        }
      }
    }
    float ans;
    Pop(&nums, &ans);
    cout << endl;
    cout << "ANSWER(x = " << i << "): " << ans << endl;
  }
}
```

ЗАКЛЮЧЕНИЕ

В ходе лабораторной работы был изучен алгоритм преобразования инфиксных выражений в обратную польскую запись и последующего их вычисления.

В процессе работы удалось реализовать алгоритм перевода из инфиксной записи в постфиксную, реализовать функцию вычисления значения выражения из постфиксной записи.

Для достижения поставленной цели были успешно выполнены основные задачи:

1) разобрать суть основных понятий используемых для преобразовании и вычислении
2) разработать алгоритмы валидации, токенизации, проверки, перевода и вычисления
3) реализовать алгоритмы
4) провести тестирование, проверить работоспособность реализаций алгоритмов

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Дмитриева Марина Валерьевна Организация данных в виде очереди // КИО. 2007. №6. URL: https://cyberleninka.ru/article/n/organizatsiyadannyhvvideocheredi (дата обращения: 25.10.2024).