ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Московский институт электроники и математики им. А. Н. Тихонова

Непомнящий Артем Евгеньевич, группа БИВ 205

Разработка программы «Тренажёр обратной польской записи на двоичных числах с функциями различного числа аргументов»

Курсовая работа по дисциплине «Алгоритмизация и программирование»
по направлению 09.03.01 Информатика и вычислительная техника
студентов образовательной программы бакалавриата
«Информатика и вычислительная техника»

Выполнил:
Студент группы
БИВ205
Непомнящий А.Е. /

Подпись

Руководитель:
Волкова Л.Л. /

Подпись

Москва 2021 г.

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Московский институт электроники и математики им. А.Н. Тихонова

ЗАДАНИЕ
на междисциплинарную курсовую работу

Студенту группы БИВ 205
Непомнящему Артему Евгеньевичу

1. Тема работы
   • Разработка программы «Тренажёр обратной польской записи на двоичных числах с функциями различного числа аргументов»

2. Требования к работе:
   • Разработать приложение, преобразующее выражение, состоящее из двоичных чисел и функций различного числа аргументов, из инфиксной нотации в постфиксную. Вычислить результат выражения.
   • Требования к функциональным характеристикам:
      1) Программа должна выводить на экран выражение, преобразованное в обратную польскую нотацию.
      2) Программа должна выводить на экран результат вычисления выражения.
   • Требования к надёжности системы:
      Программа должна совершать проверку корректности входных данных и допустимости математического выражения.
   • Требования к программной части:
      Программа должна быть написана на языке программирования C.

3. Спецификация входных и выходных данных:
   • Перечень и требования к входным данным:
      1) Исходное выражение в инфиксной форме записано в строке файла.
      2) Двоичные числа представлены последовательностью нулей и единиц.
      3) Знак оператора – один из следующих символов: «&», «^», «|», «~».
      4) Знак функции максимума от произвольного количества переменных – символ «f», за которым следуют аргументы, разделенные запятой, в круглых скобках.
   • Перечень выходных данных:
      1) Обратная польская запись выражения.
      2) Результат вычислений.

4. Пояснительная записка будет включать следующие документы:
   • Описание алгоритма решения задачи
   • Тестирование

Аннотация

Объектом разработки данной курсовой работы является консольное приложение для работы с выражениями, представленными двоичными числами и функциями различного числа переменных.

Целью курсовой работы является преобразование инфиксного выражения в обратную польскую запись и вычисление результата этого выражения.

Курсовая работа выполнена на 16 листах с использованием двух источников.

Annotation

The object of development of this course work is a console application for working with expressions represented by binary numbers and functions of various numbers of variables.

The aim of the course work is to convert an infix expression into a reverse polish notation and calculate the result of this expression.

Course work is done on 16 sheets using two sources.

Оглавление

Введение

Обратная польская запись математического выражения — это такой способ записи выражений, в котором знак операции записывается после операндов. Особенностью такого способа записи выражения является то, что порядок выполнения операций однозначно задаётся порядком следования операторов в выражении, и таким образом пропадает необходимость использования скобок, ассоциативности и приоритетов операций [1]. Преимуществом обратной польской нотации является то, что она короче инфиксной записи за счет отсутствия скобок.

В МИЭМ используются тренажёры как средство подготовки студентов к сдаче полноценных лабораторных работ, в том числе на физических установках. Так, в рамках ВКР и проектных МКР в 2021 году создан плагин для тренажёра студентов по лабораторным работам по электротехнике средствами векторного draw.io и созданных плагинов для автоматической проверки корректности созданных объектов и связей между ними по метаданным [2]. Таким образом, задача разработки тренажёров для самостоятельной подготовки студентов к выполнению лабораторных работ на базе воспроизведения моделируемых процессов является актуальной.

Цель данной курсовой работы – разработка консольного приложения-тренажёра, которое преобразует инфиксное выражение, представленное двоичными числами и функциями различного числа аргументов, в постфиксное, и вычисляет его значение.

Задачами курсовой работы являются изучение алгоритмов получения выражения в обратной польской нотации, изучение алгоритмов вычисления результата постфиксного выражения с функциями различного числа переменных, разработка и тестирование ПО.

1 Описание алгоритма решения задачи

1.1 Разработка структуры программы

Для решения поставленных целей необходимо реализовать следующие функции:

- Разбор исходного выражения на лексемы;
- Перевод выражения из инфиксной нотации в постфиксную;
- Вычисление выражения, представленного в постфиксной форме.

Считывание исходного выражения из файла необходимо производить в строку, которую затем нужно разбить на лексемы (токены).

Результатом работы функции перевода выражения в постфиксную запись так же является строка, выводящаяся в консоль, которую необходимо разбить на токены.

Результатом работы функции вычисления выражения является число, представленное в двоичном формате, которое выводится в консоль.

Для решения задачи понадобятся: стек операций для перевода выражения, стек чисел для вычисления результата, массив токенов для хранения лексем выражения.

1.2 Алгоритм разбора входной строки

Лексема, считанная из входной строки, хранится в виде структуры:

```
struct token {
int type;
char *value;
};
```

int type – тип токена.

char *value – значение токена, если его необходимо хранить.

Поддерживаются следующие типы лексем:

- type=1 – оператор (&, ^, ~, |)
- type=2 – двоичное число (набор нулей и единиц)
- type=3 – левая скобка
- type=4 – правая скобка
- type=5 – функция от n переменных f (реализована как функция максимума)
- type=6 – запятая
- type=0 – неизвестный символ
- type=-1 – конец строки

Пробельные символы в процессе токенизации пропускаются.

Таким образом, строка, разбитая на лексемы, хранится в виде массива структур.

1.3 Алгоритм перевода выражения в обратную польскую запись

При переводе выражения из инфиксной формы в постфиксную используется стек для хранения операций. Обработка массива лексем происходит в цикле, пока не встретится токен с типом -1.

- Если токен – число, то добавить его в результирующую строку.
- Если токен – функция f, то положить его в стек.
- Если токен – запятая:
    - Пока токен на вершине находится стека не левая скобка, перекладывать операторы из стека в результирующую строку. Если в стеке не нашлось левой скобки, то выражение некорректно (пропущена запятая или левая скобка).
- Если токен – оператор 1:
    - Пока на вершине стека находится оператор 2 с приоритетом выше или равному приоритету оператора 1, перекладывать оператор 2 из стека в результирующую строку.
    - Поместить оператор 1 в стек.
- Если токен – левая скобка:
    - Если на вершине стека находится функция f, добавить левую скобку в результирующую строку. Таким образом оставляется пометка, необходимая для вычисления значения функции произвольного количества переменных. Она не является составляющей обратной польской записи и не выводится пользователю в строке результата.
    - Положить левую скобку в стек.
- Если токен – правая скобка:
    - Пока на вершине стека находится не левая скобка, перекладывать операторы из стека в результирующую строку.
    - Если в стеке не нашлось левой скобки, то выражение некорректно.
    - Удалить из стека левую скобку.
    - Если на вершине стека находится функция f, переложить ее из стека в результирующую строку.
- Иначе – выражение некорректно, встречен неизвестный токен.

Далее, если в стеке еще остались операторы, перекладывать их в результирующую строку. Если в процессе встретилась левая или правая скобка, выражение некорректно (присутствует незакрытая скобка).

Если выражение, заданное пользователем, корректно, результатом работы функции будет вывод обратной польской нотации. В противном случае выводится сообщение о советующей ошибке и дальнейшие вычисления в программе не производятся.

1.4 Алгоритм вычисления выражения в обратной польской нотации

При вычислении значения выражения, записанного в постфиксной форме, используется стек для хранения чисел. Обработка массива лексем происходит в цикле, пока не встретится токен с типом -1.

- Если токен – число, то положить его в стек, предварительно переведя в десятичную систему счисления.
- Если токен – левая скобка (являющаяся вспомогательным индикатором для вычисления значений функций от n переменных), то положить в стек число -1, свидетельствующее о позиции индикатора в стеке. Данное действие возможно, так как все двоичные числа и результаты выполнения операций над ними, помещенные в стек, будут иметь положительные значения в десятичной системе счисления (следовательно, индикатор определен в стеке однозначно).
- Если токен – оператор или функция f:
    - Если токен – оператор, то соответствующая ему операция применяется к требуемому количеству чисел, взятых из стека. Результат выполненной операции помещается в стек. Если в стеке недостаточно элементов для выполнения операции, выводится ошибка.
    - Если токен – функция f и в стеке меньше двух значений (одно и значений будет индикатором), выводится ошибка (нет аргументов в функции). Иначе:
        - Пока не встретился индикатор -1, перекладывать числа из стека в дополнительный массив.
        - Применить функцию f (в данной программе это функция максимума) к дополнительному массиву.
        - Удалить из стека индикатор -1.
        - Результат работы функции f помещается в стек.

Если в стеке осталось одно значение, то оно является искомым результатом вычисления выражения. Программа выводит его, предварительно переведя обратно в двоичную систему счисления, и успешно завершает свою работу.

Если же в стеке присутствует больше одного значения, или стек пуст, то выражение некорректно и выводится сообщение о соответствующей ошибке, результат отсутствует.

2 Разработка ПО

В ходе разработки были выделены следующие типы проверки корректности данных и корректности математического выражения:

- Проверка на наличие неизвестных символов в исходном выражении;
- Проверка правильности расстановки скобок в выражении;
- Проверка правильности разработки разделителей функции в выражении;
- Проверка правильности количества операторов в выражении;
- Проверка правильности количества чисел в выражении;
- Проверка наличия аргументов у функции в выражении;

Стек операторов, а также стек чисел реализованы через односвязный список. Для работы со стеком операций реализованы следующие функции, аналогичные функциям для работы со стеком чисел:

1) Добавление оператора в стек:

```
opstck *op_push(opstck *HEAD, char a) {
    opstck *PTR;
    PTR = (opstck*) malloc(sizeof (opstck));
    PTR->c = a;
    PTR->next = HEAD;
    return PTR;
}
```

2) Удаление оператора из стека:

```
char op_del(struct opstck **HEAD) {
    opstck *PTR;
    char a;
    if (*HEAD == NULL) return '\0';
    PTR = *HEAD;
    a = PTR->c;
    *HEAD = PTR->next;
    free(PTR);
    return a;
}
```

3) Очистка стека:

```
void free_opstck(opstck *OPSstck) {
    while (OPSstck != NULL)
        op_del(&OPSstck);
}
```

Все двоичные операции, кроме отрицания, реализованы через соответствующие им битовые операции языка C. Операция отрицания реализована отдельной функцией. Функция f от произвольного количества эквивалента функции максимума max, реализована отдельной функцией.

3 Тестирование

В данном разделе будет представлено описание тестирования разработанного ПО.

1) Тестирование операции «или»:

```
$ ./a.out
ALLOWED OPERANDS: BINARY NUMBERS
ALLOWED OPERATORS: &=AND, |=OR, ^=XOR, ~=NOT, f=MAX
ENTER FILE NAME:
test.txt
EXPRESSION = 10100101|11100000
EXPRESSION IN RPN = 10100101 11100000 |

        CONVERSION SUCCESSFUL!

RESULT = 11100101

        CALCULATION SUCCESSFUL!
```

Рис. 1 – Тестирование операции «или».

```
$ ./a.out
ALLOWED OPERANDS: BINARY NUMBERS
ALLOWED OPERATORS: &=AND, |=OR, ^=XOR, ~=NOT, f=MAX
ENTER FILE NAME:
test.txt
EXPRESSION = 10100101|11100000|00000010|11101000
EXPRESSION IN RPN = 10100101 11100000 | 00000010 | 11101000 |

        CONVERSION SUCCESSFUL!

RESULT = 11101111

        CALCULATION SUCCESSFUL!
```

Рис. 2 – Тестирование операции «или».

2) Тестирование операции «и»:

```
$ ./a.out
ALLOWED OPERANDS: BINARY NUMBERS
ALLOWED OPERATORS: &=AND, |=OR, ^=XOR, ~=NOT, f=MAX
ENTER FILE NAME:
test.txt
EXPRESSION = 10100101&11100000
EXPRESSION IN RPN = 10100101 11100000 &

        CONVERSION SUCCESSFUL!

RESULT = 10100000

        CALCULATION SUCCESSFUL!
```

Рис. 3 – Тестирование операции «и».

```
$ ./a.out
ALLOWED OPERANDS: BINARY NUMBERS
ALLOWED OPERATORS: &=AND, |=OR, ^=XOR, ~=NOT, f=MAX
ENTER FILE NAME:
test.txt
EXPRESSION = 10100101&11100000&10000000
EXPRESSION IN RPN = 10100101 11100000 & 10000000 &

        CONVERSION SUCCESSFUL!

RESULT = 10000000

        CALCULATION SUCCESSFUL!
```

Рис. 4 – Тестирование операции «и».

3) Тестирование операции «исключающее или»:

```
$ ./a.out
ALLOWED OPERANDS: BINARY NUMBERS
ALLOWED OPERATORS: &=AND, |=OR, ^=XOR, ~=NOT, f=MAX
ENTER FILE NAME:
test.txt
EXPRESSION = 10100101^11100000
EXPRESSION IN RPN = 10100101 11100000 ^

        CONVERSION SUCCESSFUL!

RESULT = 1000101

        CALCULATION SUCCESSFUL!
```

Рис.5 – Тестирование операции «исключающее или».

```
$ ./a.out
ALLOWED OPERANDS: BINARY NUMBERS
ALLOWED OPERATORS: &=AND, |=OR, ^=XOR, ~=NOT, f=MAX
ENTER FILE NAME:
test.txt
EXPRESSION = 10100101^11100000^1000100
EXPRESSION IN RPN = 10100101 11100000 ^ 1000100 ^

        CONVERSION SUCCESSFUL!

RESULT = 1

        CALCULATION SUCCESSFUL!
```

Рис.6 – Тестирование операции «исключающее или».

4) Тестирование операции «не»:

```
$ ./a.out
ALLOWED OPERANDS: BINARY NUMBERS
ALLOWED OPERATORS: &=AND, |=OR, ^=XOR, ~=NOT, f=MAX
ENTER FILE NAME:
test.txt
EXPRESSION = ~10010011100
EXPRESSION IN RPN = 10010011100 ~

        CONVERSION SUCCESSFUL!

RESULT = 1101100011

        CALCULATION SUCCESSFUL!
```

Рис.7 – Тестирование операции «не».

5) Тестирование функции нескольких переменных:

```
$ ./a.out
ALLOWED OPERANDS: BINARY NUMBERS
ALLOWED OPERATORS: &=AND, |=OR, ^=XOR, ~=NOT, f=MAX
ENTER FILE NAME:
test.txt
EXPRESSION = f(0,10,101,111,110,11,1111,1001,1000,1000)
EXPRESSION IN RPN =  0 10 101 111 110 11 1111 1001 1000 1000 f

        CONVERSION SUCCESSFUL!

RESULT = 1111

        CALCULATION SUCCESSFUL!
```

Рис.8 – Тестирование функции нескольких переменных.

6) Тестирование сложного выражения 1:

```
$ ./a.out
ALLOWED OPERANDS: BINARY NUMBERS
ALLOWED OPERATORS: &=AND, |=OR, ^=XOR, ~=NOT, f=MAX
ENTER FILE NAME:
test.txt
EXPRESSION = ((1011^1101)|(100000&(0000|100000)))
EXPRESSION IN RPN = 1011 1101 ^ 100000 0000 100000 | & |

        CONVERSION SUCCESSFUL!

RESULT = 100110

        CALCULATION SUCCESSFUL!
```

Рис.9 – Тестирование сложного выражения 1.

7) Тестирование сложного выражения 2:

```
$ ./a.out
ALLOWED OPERANDS: BINARY NUMBERS
ALLOWED OPERATORS: &=AND, |=OR, ^=XOR, ~=NOT, f=MAX
ENTER FILE NAME:
test.txt
EXPRESSION = ((1011^1101)|(100000&(0000|100000)))^~f((1111&1100)|0010,100|11000,11,1,0)|(1100&(0100^(~0101)))|f(11000000000&f(1000000000),1)
EXPRESSION IN RPN = 1011 1101 ^ 100000 0000 100000 | & |  1111 1100 & 0010 | 100 11000 | 11 1 0 f ~ ^ 1100 0100 0101 ~ ^ & |  11000000000  1000000000 f & 1 f |

        CONVERSION SUCCESSFUL!

RESULT = 1000100101

        CALCULATION SUCCESSFUL!
```

Рис.10 – Тестирование сложного выражения 2.

8) Тестирование сложного выражения 3:

```
$ ./a.out
ALLOWED OPERANDS: BINARY NUMBERS
ALLOWED OPERATORS: &=AND, |=OR, ^=XOR, ~=NOT, f=MAX
ENTER FILE NAME:
test.txt
EXPRESSION = ~f(10101^(f(10|1,1,f(~1,~0))|100),101&10001)
EXPRESSION IN RPN =  10101  10 1 | 1  1 ~ 0 ~ f f 100 | ^ 101 10001 & f ~

        CONVERSION SUCCESSFUL!

RESULT = 1101

        CALCULATION SUCCESSFUL!
```

Рис.11 – Тестирование сложного выражения 3

9) Тестирование сложного выражения, содержащего лишнюю левую скобку:

```
$ ./a.out
ALLOWED OPERANDS: BINARY NUMBERS
ALLOWED OPERATORS: &=AND, |=OR, ^=XOR, ~=NOT, f=MAX
ENTER FILE NAME:
test.txt
EXPRESSION = (~f(10101^(f(10|1,1,f(~1,~0))|100),101&10001)
ERROR: PARENTHESIS MISMATCHED!
```

Рис.12 – Тестирование выражения, содержащего лишнюю левую скобку.

10) Тестирование сложного выражения, содержащего лишнюю правую скобку:

```
$ ./a.out
ALLOWED OPERANDS: BINARY NUMBERS
ALLOWED OPERATORS: &=AND, |=OR, ^=XOR, ~=NOT, f=MAX
ENTER FILE NAME:
test.txt
EXPRESSION = ~f(10101^(f(10|1,1,f(~1,~0))|100),101&10001))
ERROR: EMPTY STACK(RIGHT_PARENTHESIS)!
```

Рис.13 – Тестирование выражения, содержащего лишнюю правую скобку.

11) Тестирование сложного выражения, в котором не достает аргумента оператора:

```
$ ./a.out
ALLOWED OPERANDS: BINARY NUMBERS
ALLOWED OPERATORS: &=AND, |=OR, ^=XOR, ~=NOT, f=MAX
ENTER FILE NAME:
test.txt
EXPRESSION = (111^~(10101))^
EXPRESSION IN RPN = 111 10101 ~ ^ ^

        CONVERSION SUCCESSFUL!

ERROR: NOT ENOUGH ARGUMENTS!
```

Рис.14 – Тестирование выражения с пропущенным аргументом.

12) Тестирование выражения с лишним числом:

```
$ ./a.out
ALLOWED OPERANDS: BINARY NUMBERS
ALLOWED OPERATORS: &=AND, |=OR, ^=XOR, ~=NOT, f=MAX
ENTER FILE NAME:
test.txt
EXPRESSION = 10101^11010 101
EXPRESSION IN RPN = 10101 11010 101 ^

        CONVERSION SUCCESSFUL!

ERROR: MORE THAN ONE ELEMENT IN STACK AT THE END OF CALCULATIONS!
```

Рис.15 – Тестирование выражения с лишним числом.

Все тесты успешно пройдены.

Заключение

В процессе работы над данной курсовой работой были реализованы оптимальные алгоритмы, посредством стека, получения выражения в обратной польской нотации и вычисления результата постфиксного выражения, поддерживающие функции произвольного количества переменных.

Результатом курсовой работы является консольное приложение, которое считывает данные из заданного пользователем файла и вычисляет ответ.

Программа была разработана на языке программирования C.

Список использованных источников

1) https://hpmuseum.org/rpn.htm (дата обращения 01.03.2021)
2) Карелина Е.А. Разработка программного обеспечения для автоматизации составления, выполнения и проверки графических заданий в редакторе draw.io. Выпускная квалификационная работа бакалавра. МИЭМ им. А.Н. Тихонова НИУ ВШЭ, 2021 г.