# Advanced Calculator Interpreter in C Programming Language

**CmpE 230 Systems Programming Spring 2023**

Damla Kayıkçı - 2020400228
Çağatay Çolak - 2020400060

**April 1, 2023**

## 1 Introduction

The advanced calculator (AdvCalc) project is an interpreter implementation for an advanced calculator using the C language. This calculator accepts expressions and assignment statements and outputs the results by performing arithmetic and bitwise operations on them. Operations are as follows:

| Operations | Description |
|---|---|
| a + b | Returns summation of a and b. |
| a * b | Returns multiplication of a and b. |
| a - b | Returns the subtraction of b from a. |
| a & b | Returns bitwise a and b. |
| a \| b | Returns bitwise a or b. |
| xor(a, b) | Returns bitwise a xor b. |
| ls(a, i) | Returns the result of a shifted i bits to the left. |
| rs(a, i) | Returns the result of a shifted i bits to the right. |
| lr(a, i) | Returns the result of a rotated i times to the left. |
| rr(a, i) | Returns the result of a rotated i times to the right. |
| not(a) | Returns bitwise complement of a. |

## 2 Program Execution

The user can input expressions and assignment statements in the AdvCalc program. The expressions can be simple or complex mathematical operations involving addition, subtraction, multiplication, and bitwise operations. Assignment statements can be used to assign values to variables that can be used later in expressions. For example, the user can input "x=5+2" to assign the value of 7 to the variable x.

## 3 Input and Output

AdvCalc reads input from the command line and outputs the results to the console. The input and output format is as follows:

## 3.1 Input Format

Each input line should contain a single statement that represents an expression or assignment statement. The statement should adhere to the following syntax:

```
<variable> = <expression>
```

where:

- `<variable>` is the name of a variable (e.g. x, y, z)
- `<expression>` is a mathematical expression that can contain any combination of variables, numbers, and operators (+, -, *, /,

## 3.2 Output Format

After each input statement is executed, AdvCalc outputs the result to the console on a new line. The output format depends on the type of statement:

- For an expression statement, AdvCalc outputs the result of the expression as an integer.
- For an assignment statement, AdvCalc outputs the value assigned to the variable as an integer.

If an error occurs during execution, AdvCalc outputs an error message to the console and terminates execution.

## 3.3 Example Input-Output

```
1 % ./advcalc
2 >x = 1
3 >y = x + 3    %4
4 >z = x * y * y*y   %64
6 64
7 >qqq = xor(131, 198)
8 >qqq
9 69
10 > xor(((x)), x)
11 0
12 > xor(((x)), x) | z + y
13 68
14 > rs(xor(((x)), x) | z + y, 1)
15 34
16 > ls(rs(xor(((x)), x) | z + y, 1), (((1))))
17 68
18 > lr(ls(rs(xor(((x)), x) | z + y, 1), (((1)))), 1)
19 136
20 > rr(lr(ls(rs(xor(((x)), x) | z + y, 1), (((1)))), 1), 1)
21 68
```

```
22 > qqq * not(not(10))
23 690
24 > rr(lr(ls(rs(xor(((x)), x) | z + y, 1), (((1)))), 1), 1) - qqq * not(not(10))
25 -622
26 > 0 & rr(lr(ls(rs(xor(((x)), x) | z + y, 1), (((1)))), 1), 1) - qqq * not(not(10))
27 0
28 > <Ctrl-D>
29 %
```

# 4 Program Structure

The project includes four header files: `token.h`, `tokenizer.h`, `formatController.h`, and `postfixCalculator.h`.

### 4.0.1 `token.h`

The `token.h` header file defines a `token` struct, which represents a token in the expression. The `token` struct includes the token type, the token name and the token value.

### 4.0.2 `tokenizer.h`

The `tokenizer.h` header file defines a `tokenizer` function, which tokenizes the input expression. The tokenizer splits the expression into tokens while extracting undefined characters here and returns them as an array.

### 4.0.3 `formatController.h`

The `formatController.h` header file defines a `formatController` function, which checks the function operations like xor,ls,rs and so on. The formatController function removes commas that are between the left and right expressions of the operations and inserts the operation sign and makes the expression easier to parse.

### 4.0.4 `postfixCalculator.h`

The `postfixCalculator.h` header file defines a `postfixCalculator` function, which evaluates the input expression. The calculator has a method inside called `infixToPostfix` which converts the expression from infix to postfix notation and postfixCalculator uses the output of this method and uses a stack to evaluate the expression.

## 4.1 Program Flow

1. The user enters an expression.
2. The `tokenizer` function is called to tokenize the expression.
3. The `formatController` function is called to make the changes on the tokenized input.
4. The `postfixCalculator` function is called to convert the expression infix to postfix and evaluate the expression.

5. The result is displayed to the user.

## 4.2 Usage

To use the AdvCalc project, follow these steps:

1. Include the required header files in your program.
2. Call the `tokenizer` function to tokenize the input expression.
3. Call the `formatController` function to make the changes on the tokenized input.
4. Call the `postfixCalculator` function to convert the expression infix to postfix and evaluate the expression.
5. Display the result to the user.

## 4.3 Some Of The Functions

### 4.3.1 `shouldntRepeat()`

The shouldntRepeat function checks whether a character should not be repeated in an expression. The function returns 1 if the character is not allowed to be repeated, and 0 otherwise. The function checks for characters that are operators or separators in the expression, such as + - * / | , and =.

```
1 % ./advcalc
2 >1+-1
3 Error!
```

### 4.3.2 `returnIndex()`

The returnIndex function searches for a variable with the given name in the variable array of Token objects representing variables. If the variable is found, the function returns the index of the variable in the array. If the variable is not found, the function returns -1.

# 5 Difficulties Encountered

The use of the C programming language was one of the main difficulties encountered. While C is a widely used language, it may not be familiar to all programmers. The steep learning curve of C also made it difficult to create the program quickly. Longer development periods and a greater chance of code errors were the results of this.

Memory management presented another difficulty while using C. Memory leaks and segmentation errors were frequent problems during the development process since C does not have automatic garbage collection.

# 6 Conclusion

In conclusion, the AdvCalc project successfully implemented an interpreter for an advanced calculator using the C programming language. The program is capable of accepting expressions and assignment statements and performing arithmetic and bitwise operations on them.