جامعة الأخوين
AL AKHAWAYN
UNIVERSITY

CSC 3315 Languages & Compilers

Lab: Designing a Simple Interpreter for Arithmetic Expressions (C)

Pr. Moulay Youssef Hadi

An arithmetic expression is a set of operands connected by operators. The usual notation of arithmetic expressions is called infix notation. Here, we focus on fully parenthesized infix expressions, such as:

`( ( ( 3 + 4 ) * 8 ) - 6 )` OR `( 3 + ( 4 * ( 8 - 6 ) ) )`

Evaluating an infix notation expression can be done in two steps:

1. Conversion to postfix notation.
2. Evaluation of the postfix expression.

Conversion to Postfix Notation

An expression can be represented by a string of characters:

Example: The expression `( 3 + ( 4 * ( 8 - 6 ) ) )` is represented as follows:

(	3	+	(	4	*	(	8	-	6	)	)	)	\0

We consider a string of characters **`expInf`** representing an expression. The conversion of this expression to postfix notation is done using a stack of operators according to the following steps:

1. Create an empty string **`expPost`** to store the postfix expression.
2. Create a stack of character elements.
3. Traverse the expression **`expInf`**:
   - If an opening parenthesis **`'('`** is encountered, do nothing.
   - If it is an operand, it is directly copied into the string **`expPost`**.
   - If it is an operator (**`+`**, **`-`**, **`*`**, **`/`** or **`%`**), it is pushed onto the stack.
   - For each closing parenthesis **`')'`** read, an operator is popped from the stack and copied into **`expPost`**. At the end of the traversal, the string **`expPost`** contains the postfix notation of the expression **`expInf`**.
4. At the end of the traversal, the string **`expPost`** contains the postfix notation of the expression **`expInf`**.

Example: If **`expInf = (3+(4*(8-6)))`**, then **`expPost = 3 4 8 6 - * +`**.

Evaluation of the Postfix Expression

For simplicity, we limit to integer operands between **0** and **9**. To avoid arithmetic problems, we will only use the operators **+**, **-**, *****, and **/**.

Postfix notation is evaluated very simply using a stack of operands:

- Operands are pushed onto the stack as they are read.
- Operators are applied immediately by popping the top two operands from the stack and pushing the result back onto the stack.
- The final result is the only value left in the stack.

Example: Evaluating "**3 4 8 6 - * +**" results in **11**.

The expression	3	4	8	6	-	*	+
Stack state	3	4 3	8 4 3	6 8 4 3	2 4 3	8 3	11

And the evaluation of "**3 4 + 8 * 6 -**" (in infix notation **(((3+4)*8)-6))** results in **50**.

Tasks to Complete:

1. Define the structure of an operand stack **StackOpd**.
2. Write the function **void initStackOpd(StackOpd * adrP)** to initialize an empty operand stack at address **adrP**.
3. Write the function **void empilerOpd(StackOpd *adrP, int x)** to add an element x to the stack at address **adrP**.
4. Write the function **int depilerOpd(StackOpd *adrP)** to remove and return the top of the stack.
5. Redo the previous tasks with an operator stack **StackOpr** (make sure to change the function names).
6. Write the function **char *convertirInf2Post(char *expInf)** that takes an infix notation expression as a parameter and returns its conversion to postfix notation.

7. Write the function **`int calculate(int x, int y, char op)`** that takes two operands `x` and `y` and an operator `op` as parameters to compute the expression `x op y`. Example: the call **`calculate(3, 7, '+')`** should return **`10`**.

8. Write the function **`int evaluer(char *exp)`** that takes a postfix expression as a parameter and returns an integer which is the result of evaluating **`exp`**. To do this:

   - Convert **`exp`** to a postfix notation **`expPost`**.
   - Create an empty integer stack to push/pop operands.
   - Traverse **`expPost`** and evaluate it following the procedure described above.
   - Return the result.

9. Write the main function **`main()`** to:

   - Declare and read a fully parenthesized infix expression **`expInf`**.
   - Evaluate **`expInf`**.

Note:

- **`int isdigit(int c)`**: Returns a non-zero value if **`c`** is a decimal digit (**`ctype.h`**).
- **`int atoi(const char *str)`**: Returns the numeric value represented by **`str`** as an **`int`** (**`stdlib.h`**).