Project: Assembly Language II

COSC 216.001

Due Date: April 9, 2023, 11:59pm

1 Overview

Each student is asked to write an assembly language program that accesses and makes use of C-subroutines to convert an infix expression to a postfix expression and then evaluate the postfix expression.

1.1 Objectives

- Understanding Assembly language code
- Interfacing C and Assembly code
- Designing and planning a larger Assembly program

2 Detailed Description

2.1 Introduction

In this project, you are asked to integrate pre-existing C-functions into your assembly language program. Your code, while written in assembly language, will have to pass parameters, call C functions, and utilize the return values. You will need to understand the C-function interface and call structure. You will implement a program that converts infix expressions to postfix expressions and then evaluates the resulting postfix expressions.

2.2 What to do

You are to implement a program that performs the following tasks:

1. Convert an infix expression to a postfix expression.
2. Evaluate the postfix expression and return the final result.

Your assembly language program should assume a string address is passed in via `R0` as a packed C-style string representing the infix expression. You should then run the infix to postfix algorithm specified below, convert the infix expression to a postfix expression as a string. Then, evaluate the postfix expression and return the final result as an integer via `R0`.

Your code may call the following C-style functions, as necessary (these functions are provided in VisUAL compatible ARM format in util.s):

```
// returns the length of string s
int STRLEN(char *s);

// allocates and returns a string that is the concatenation of strings s1 and s2.
char *CONCAT(char *s1, char *s2);

// If possible, allocates bytes from the heap. This is an all-or-nothing attempt.
void *ALLOCATE(unsigned long bytes);

// multiply 2 integer operands together.
int MULTIPLY(int x, int y);

// return the result of integer division of x by y (x/y)
int DIVIDE(int x, int y);

// return the remainder of integer divison of x by y (x%y)
int MODULO(int x, int y);
```

In many ways this project is much simpler than the similar CMSC204 project. Rather than trying to port that project into Assembly language, re-design and plan around the strengths and weaknesses of the language.

Some simplifying assumptions:

1. The input string will only contain 1-digit, positive numbers
2. The '−' will always be used to represent subtraction (not negative numbers)
3. The only operations supported are: addition, subtraction, multiplication, integer division, and modulus.
4. You can assume the input string is "correct"
5. Your code should provide a means of going from infix-notation and evaluating the expression with a single branch-link instruction.
6. You may modify code from util.s as necessary
7. The GFA for this assignment is to complete at least the infix-to-postfix conversion.

2.2.1 Infix to Postfix Algorithm

Process each character of the input as follows:

1. For each character in the input string:
   (a) Assume your code is accessed as C-functions, and needs to conform to the C-function interface.
   (b) If the character is an digit, append it to the output string.
   (c) If the character is an operator $[*/\% + -]$, pop operators from the stack and append them to the output string until the stack is empty or until the operator at the top of the stack has a lower precedence than the current operator. Then, push the current operator onto the stack.
   (d) If the character is an open parenthesis "(", push it onto the stack.
   (e) If the character is a close parenthesis ")", pop operators from the stack and append them to the output string until an open parenthesis is encountered. Pop the open parenthesis from the stack.
2. Pop any remaining operators from the stack and append them to the output string.

2.2.2 Evaluating the Postfix Expression

1. Process each character of the postfix expression as follows:
   (a) If the character is a digit, push its integer value onto the stack.
   (b) If the character is an operator, pop the top two operands from the stack, perform the operation, and push the result back onto the stack.
2. The final result is the single integer remaining on the stack.

2.3 Running your code

Please use the VisUAL2 ARM emulator to test your code. You can download VisUAL at: `https://github.com/tomcl/V2releases/releases`

3 Submission

Each of your source code files must have a comment near the top that contains your name, StudentID, and M-number.

The project should be submitted by **April 9, 2023, 11:59pm**. Follow these instructions to turn in your project.

You should submit the following files:

- infix_eval.s
- *any other source files your project needs*

The following submission directions use the command-line `submit` program on the class server that we will use for all projects this semester:

- Log into the VDI: `http://desktop.montgomerycollege.edu`
- If necessary, transfer your source code onto the VDI
- Use *Bitvise SSH* client to log into `tpaclinux`
- If necessary, use the *Bitvise SFTP transfer* to upload your source code to the server
- Finally, use the command-line `submit` program to turn in your code

An example command line submission of file: `infix_eval.s`, for project 5, would look like:

```
submit proj5 infix_eval.s
```

Late assignments will not be given credit.

4 Grading

While this rubric is subject to change based on class performance, the current grading rubric for this assignment is as follows:

component	value
Correctness	40
Completeness	40
Code Quality	20

Please note that your code may be subject to plagiarism detection software checks.

5 Getting Help

If you need help with this project, please reach out to the instructor through the class discord server, email, or during office hours.