CS 203 F19 Project 2: Creating A Small Interpreted Programming Language

C W Liew

Version as of: 13:34 Tuesday 29th October, 2019

Due: 11:59pm, December 8th, 2019

Goals

The goals of this project are:

1. build a small programming language
2. gain more experience working with C

General Description

In this project, you will design and implement (in the C programming language) a small programming language to manipulate expressions. The project will modify and extend your earlier project (Project 1).

The programming language will consist of expressions with the following additions and modifications:

- there are built-in functions which are modifications of the commands that you had for subsets. The built-in functions are:
  - f - first
  - r - rest
  - a - append

  These are technically functions with one or two parameters
- new command - 'v' (define): this allows you to define variables
- new command - 'd' (define): functions without parameters
- new command - 'l' : functions with an arbitrary number of parameters
- the 'p' (print) command has been modified to use a variable/function name
- Expressions now include defined functions and variables. With variables, you can now have multiple expressions available.

The expressions you will read in will be of the form:

```
( <operator> <space separated list of operands> ) or
```

or

```
( <function> <space separated list of operands> )
```

or

```
< number $>
```

Examples

Here are some examples of valid expressions (there is a recursive definition)

```
2
(+ 2 4)
(+ 2)
(+ 2 (+ 2 4) 6)
```

New examples:

```
(r b1)
(f (r b2)
(a b1 b2)
```

The valid operations are: ”+”, ”-”, ”*”, **f, r, a**

Data Structure

The basic structure you will use consists of a Expression (Exp) defined as:

```
char* symbol or Exp* first;
Exp* rest;
```

where the symbol can be an operator (e.g., ”+”) or a number (e.g., ”256”).
So the structures corresponding to the examples would be:

- Exp(2)
- ((Exp(+) $->$ Exp(2)) $->$ Exp(4))
- (Exp(+) $->$ Exp(2))
- (Exp(+) $->$ Exp(2) $->$ (Exp(+) $->$ Exp(2) $->$ Exp(4)) $->$ Exp(6))

Commands

The commands that can be used are:

- ’c’: create a new expression. The next line contains the expression. A new data structure is created and the current expression will point to it. The current expression is stored as the variable *foo*.
- ’p’: print the variable named on the next line

- ’e’: evaluate the expression named on the next line (arithmetically)
- ’a’: append a new expression to the current expression. The new expression is given on the next line

  Assuming the current expression is: ”(+ 1 2 3)”.

  Appending ”4” would result in ”(+ 1 2 3 4)”, while appending ”(+ 4)” would result in ”(+ 1 2 3 (+ 4))”
- ’s’: prints a subset of the expression. The subset specification is on the next line and uses ’f’ (first) and ’r’ (rest)

  For example, if the current expression is ”(+ 2 (- 3 4) 5)”

  - ”f” results in ”+”
  - ”r f” results in ”2”
  - ”f f” does not make sense
  - ”r r f” results in ”(- 3 4)”
- **’v’**: define a variable. The next line contains the variable and the line after that contains the expression. A new data structure is created to contain the expression.
- **’d’**: define a function (without parameters). The next line contains the function name and the line after that contains the expression. A new data structure is created to contain the expression.
- **’l’**: define a function with parameters. The next line contains the function name, the line after that the parameters, and then a line that contains the expression. A new data structure is created to contain the expression.

Example

```
c
(+ 2 4 (- 10 7) (* 3 (+ 2 5)))
p
foo
(+ 2 4 (- 10 7) (* 3 (+ 2 5)))
e
30
a
10
e
40
s
r r r f
(-10 7)
```

New Examples

- Defining Variables

```
c
(+ 2 4)
p
foo
(+ 2 4)
d
x
4
d
y
6
d
z
(+ x y)
p
z
(+ x y)
e
z
10
d
b10
(+ 2 4 (- 10 7) (* 3 (+ 2 5)))
p
b10
(+ 2 4 (- 10 7) (* 3 (+ 2 5)))
```

- Defining Functions Without Parameters

```
d
b1
(+ 4 (+ 5 6) 2)
d
b2
(f (r (r b1)))
e
b1
17
```

```
e
b2
11
d
b10
(+ 2 4 (- 10 7) (* 3 (+ 2 5)))
p
b10
(+ 2 4 (- 10 7) (* 3 (+ 2 5)))
d
b20
(f (r (r (r b10))))
e
b20
3
d
b10
(* (+ 3 4) 5 (+ 2 6))
e
b20
8
```

- Defining Functions With Parameters

```
l
f1
x y
(+ (* x y) 3)
c
(f1 2 3)
e
foo
9
c
(f1 (+ 2 3) (+ 4 1))
e
foo
28
```

Figure 1 shows you the structure corresponding to the expression:

```
(+ 2 4 (- 10 7) (* 3 (+ 2 5)) 10)
```

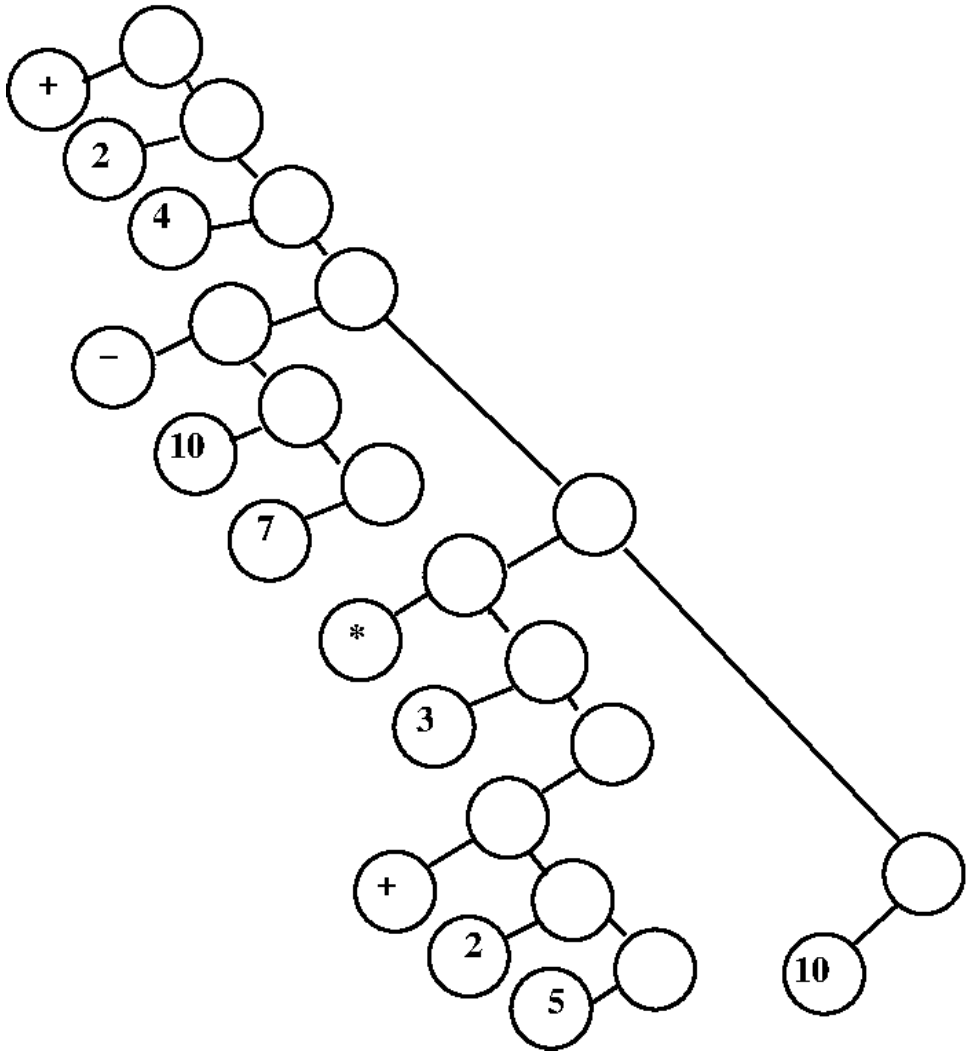

Figure 1: The structure corresponding to the example expression

Grading

Your program will be graded on the following criteria:

- functionality - how much of the specified functionality works?
  - basic - define, print and evaluate variables
  - good functionality: evaluate with built-in functions (first, rest, append)
  - very good functionality: define, print and evaluate functions without parameters
  - max functionality: define, print and evaluate functions with parameters
- design - is your program decomposed in an appropriate manner?
- documentation - how well commented is the program