**3740 Project Summary**

**1.0 Introduction**

For this project, we designed and implemented a programmable calculator called *UofL,* where the implementation done with Scheme Language. The development environment used was Dr. Racket. The programmable calculator is implemented as as interpreter. The calculator will prompt the user to enter in mathematical expresses which is calculated following traditional rules. The calculator also implements non traditional methods and allows users to define functions within the interpreter.

**2.0 Data Structure**

The interpreter only allows static scoping and variables must be defined before its use within a function, i.e. there are no local variables within a function.

Data defined include:

* Variables that can take on values of:
  + integer
  + float
  + boolean (true or false)
* Functions that are user defined
* Variables and Functions are user defined with names using alphanumeric characters

Operations allowed include:

* Arithmetic operators:

The following operators allowed are addition, subtraction, multiplication, division, power, respectively:

* + +, -, \*, /, ^
* Relational operators:

The following operators allowed are equal to, not equal to, greater than or equal to, smaller than or equal to, greater than, smaller than, respectively:

* + ==, <>, >=, <=, >, <

Statements allowed include:

* Assignment:
  + =
    - a = b + 2
* Selection:
  + If then
    - if (a <6) then b = 8 elseif (a > 6) then b = 10 endif
* Iterative:
  + For loops
    - For I = 1 to stepsize 2 do
    - a = a\*2
    - endfor

2.1 BNF Grammar

The UofL interpreter follows the BNF grammar definition method.

|  |  |
| --- | --- |
| <uc\_letter> -> A | B | ... | Z  <lc\_letter> -> a | b | ... | z  <letter> -> <uc\_letter>  | <lc\_letter>  <digit> -> 0 | 1 | ... | 9  <alphanum> -> <letter>  | <digit>  <alphanums> -> <letter>  | <number>  | <letter> <alphanums>  | <number> <alphanums>  <id> -> <letter>  | <letter> <alphanums> | <expr> -> <expr> + <term>  | <expr> - <term>  | <term>  <term> -> <term> \* <factor>  | <term> / <factor>  | <factor>  <factor> -> <exp> ^ <factor>  | <exp>  <exp> -> (<expr>)  | <id>  <assign> -> <id> = <expr> |

*Table 1. This table shows a subset of the BNF used for UofL.*

**3.0 Difficulties**

Initially we followed BNF grammar method that defined the structure of the interpreter. Defining the BNF grammar require the least amount of work but implementing on Dr.Racket was difficult, since the interpreter had to follow scheme constructs. All group members were to new to Scheme language, which led to issues implementing the parser tool. Furthermore, these token and parser were used to implement the interpreter grammer, which resulted in the team having to understand how to handle the tokens. Also, taking the input from the command line and breaking it into an expressions was quite challenging and required a lot of our time.

**4.0 Limitations**

The interpreter can not handle incorrect user input.

For example, when defining a function, the form must follow:

#definefunc *(predefined function name) (predefined variable name)*

*(body)*

*#*definefunc

but, if the form has an incorrect spelling in the #definefunc, such as #definefuna, or if functions or variables have not been predefined, the UofL interpreter will crash and exit.

**5.0 Solutions**

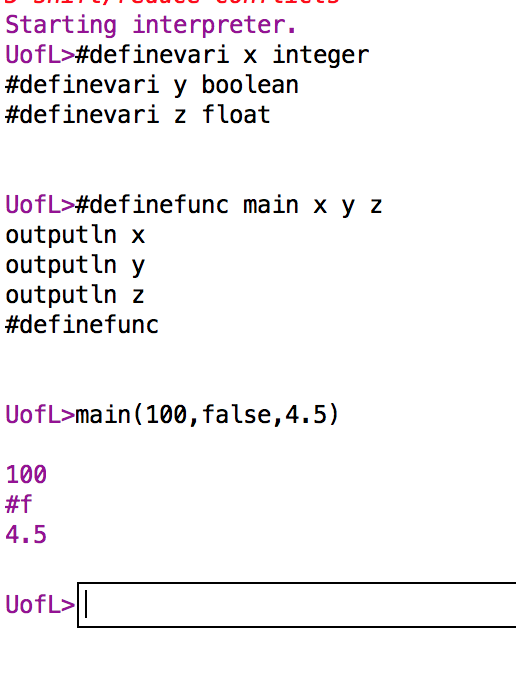


Figure2. Solution for declaring a function and calling the function

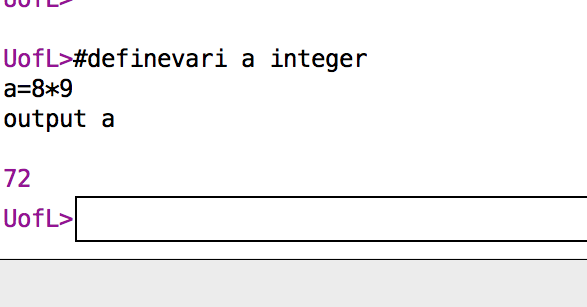


Figure 3. Performing a mathematical expression

|  |
| --- |
|  |
| #definefunc factorial x |
|  |

|  |
| --- |
| #definevari temp integer |
|  |

|  |
| --- |
| temp = x |
|  |

|  |
| --- |
| x = 1 |
|  |

|  |
| --- |
| for I = 2 to temp stepsize 1 do |
|  |

|  |
| --- |
| x = x \* I |
|  |

|  |
| --- |
| endfor |
|  |

|  |
| --- |
| #definefunc |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| #definefunc main |
|  |

|  |
| --- |
| #definevari x integer |
|  |

|  |
| --- |
| input x |
|  |

|  |
| --- |
| factorial(x) |
|  |

|  |
| --- |
| outputln x |
|  |

|  |
| --- |
| #definefunc |
|  |

|  |
| --- |
|  |
|  |

main()