**CPSC 3740 Programming Languages**

# A Programmable Calculator
## Final Project
## Spring 2018
## (50 marks)
## Due: TBA

**Each group can have at most *four* students and each group must finish the project independently. All the members in a group will be awarded the same mark. So it is your responsibility to make sure that each member in your group contributes equally to the project.**

Design and implement a programmable calculator called *UofL*, which is described below. The programming language chosen for your implementation is *Scheme* and the development environment is *Dr. Racket*.

The programmable calculator UofL you are doing in this project mimics a calculator, which has a basic programming capability, in addition to those traditional functionalities, such as arithmetic operations.

## (A) Functionalities in UofL

We describe UofL in the following. It would be more formal to use BNF grammar to describe UofL. But since it is so simple, we would rather just use some examples to show its functionalities.

**The arithmetic operators are**:

| | |
|---|---|
| + | (addition) |
| - | (subtraction) |
| * | (multiplication) |
| / | (division) |
| ^ | (power) |

Since it is a programmable *calculator*, the set of its programming constructs is limited. For this project, we only include the following programming constructs.

**The relational operators are**:

| | |
|---|---|
| == | (equal) |
| <> | (not equal) |
| >= | (bigger than or equal to) |
| <= | (smaller than or equal to) |
| > | (bigger than) |
| < | (smaller than) |

**The data types in uofl are**:

integer
float
boolean (*true* or *false*)

**Arithmetic expression**:

Ex: *a* = (*b* + 2) * 5 + 2^*b*
*b* = *b* + 4 * 6 / *c* + 7 * *c*

We use the ordinary precedence rule and associativity rule to evaluate them

**Relational expression**:

Ex: *a* == *b* to check whether *a* is equal to *b* or not
*a* >= *b* to check whether *a* is bigger than *b* or is equal to *b*

Relational expressions return *true* or *false*.

**Assignment statement**: =

Ex: a = 4
where *a* is a variable you defined before. See below.

**Selection statement**: if then elseif then endif *or* if then endif

```
Ex:  if (a ==4) then
         b = 8
     elseif (a == 5) then
         b = 9
     endif
```

**Iterative statement**: for to stepsize do endfor

```
Ex:  for I = 1 to 10 stepsize 2 do
         other statements
     endfor
```

The iterative statement in uofl is very limited, since this is a calculator. It is a counter-controlled loop, with the counter fixed as I. There also can be at most two embedded loops, with one in the other. The outer loop uses counter I while the inner loop uses counter J, as shown below.

```
Ex:  for I = 1 to 10 stepsize 2 do
         for J = 1 to 20 stepsize 3 do
             other statements
```

```
        endfor
    endfor
```

In the loops, you can use *I* and *J* as integers, if needed.

**Input statement**: input

Ex: input *a*

Get an input from user and put it into variable *a*, where *a* should be declared before.

**Output statement** output

Ex: output *a*

Output the value of *a* to the screen, where *a* should be declared before.

Ex: output "The result is ", *a*

## (B) Creating a program in UofL

The programmable calculator is implemented as an interpreter. Its prompt is UofL>, at which you can type in a command as defined below. The command will be then interpreted by UofL.

(a) To define a variable that can be used in an expression, statement, a function and a program, you need to declare it.

UofL> #definevari *myint* integer

In the above, the command is #*definevari*, which defines myint as an integer

(b) To assign a value to a variable, use the following

UofL> *myint* = 10

The above will assign 10 to integer variable *myint*

(c) To evaluate an arithmetic expression, just type it at the prompt.

UofL> *myint* = *a* * *b* + *b* / 4

where variables *a* and *b* are declared before. The expression is evaluated and the result is assigned to variable *myint*.

UofL> *a* * *b* + *b* / 4

The expression is evaluated and the result is printed out on the screen.

(d) To define a function, you need to use #*definefunc* command

```
UofL> #definefunc myfunc a b
                    a = b ^ 2
                    output a
          #definefunc
UofL>
```

There is no returned result from any function. So a function is a parameterized procedure for a particular computation.

After you define the function, you can call it.

```
UofL> myfunc 1 2
```

There is a *main* function, which is the entrance point of a program (if there is a program).

```
UofL> #definefunc main
```

To execute your program, issue the following command at the prompt.

```
UofL> main
```

(e) UofL is the interpreter adopting static scoping. There are no local variables in any function.

(f) I and J are two special reserved words for controlling loops.

(g) Any variable must be declared and then initialized before its use.

(i) Recursion is not supported in UofL.

(j) #exit will exit UofL and return to Dr.Racket

(k) #clear will clear and remove all the variables, functions, program a user defined before.

## (C) The UofL interpreter

Your interpreter prints to the screen the prompt `UofL>`. A user interacts directly with your interpreter, typing sequences of strings which are then read and executed by the interpreter.

It is assumed that a user program is always grammatically correct, i.e., the same function name is never entered twice, etc. However, if a variable is not declared before its use, it should be reported.

It is assumed that the arithmetic expression, which is for the functionalities of a basic calculator, is always correct.

You can also state any reasonable assumption and introduce some functionalities that you think would help your project. But the above items should be implemented.

## (D) Testing and submitting project:

Before submitting your code,

(1) Create a folder.
(2) Copy all of the source code to this folder
(3) Create a hardcopy README file that:

- gives the names and email addresses of all the team members,
- outlines the organization of your program, and
- specifies what testing process you used to confirm the correctness of your program.

Your interpreter starts with a uofl function call in Dr. Racket, i.e., the marker can execute

> (uofl)

to get the prompt UofL> and start to examine it.

(4) Create a hardcopy document which details what you have done for this project, the key data structures, any difficulties you have encountered, any limitations of your interpreter, etc. It should be no more than 5 pages.

In the document, you also need to show 3-4 sample runs of your interpreter, e.g., the execution of some commands or user-defined functions. In particular, you need to include the solutions to the following two problems.

(a) Use UofL stack language to create a function called *factorial* which takes one integer parameter *n* and calculates *n*!. The function needs to print it out.

(b) Use UofL stack language to create a function called *compare* that takes two integer parameters m and n. The function compares which one is bigger and outputs a message showing this. Then create two loops with one nested in the other to implement the following operations.

```
for I = 1 to 30 stepsize 2 do
        for J = 1 to 80 stepsize 3 do
                compare(i, j)
        endfor
endfor
```

(5) Submit the project: TBA
(6) A demo of the project: TBA

**(E) If necessary, there might be other information that would be announced.**