F29LP: Programming Assignment PART 1 [10 MARKS]

Your overall task is to develop a compiler for the programming language given below, called FUNC. You are expected to use the labs throughout the course to work on it, as this is where you can get help with any difficulties you have. This task is composed of two parts:

- **Part 1** is concerned with the implementation of **the compiler's front end** (this document). This is worth **10 marks**.
- **Part 2** is concerned with the implementation of **the compiler's back end**. This is also worth **10 marks** and released later in due course. The compiler should eventually produce MIPS code that can be emulated using the MARS MIPS emulator.

SUBMISSION DETAILS You have to submit a .zip archive of your source code on Vision before each deadline.

The Source Language

The FUNC language has the following syntax:

```
<program> ::= <methods>
<methods> ::= <method>;[<methods>]
<method> ::= method <id>([<args>]) [vars <args>]
     begin <statements> [return <id>;] endmethod
<args> ::= <id>[,<args>]
<statements> ::= <statement>;[<statements>]
<statement> ::= <assign> | <if> | <while> | <rw>
<rw> ::= read <id> | write <expr>
<assign> ::= <id> := <expr>
<if> ::= if  <cond> then <statements> [else <statements>] endif
<while> ::= while <cond> begin <statements> endwhile
<cond> ::= <bop> ( <exprs> )
<bop> ::= less | lessEq | eq | nEq
<exps> ::= <exp> [,<exps>]
<exp> ::= <id>[( <exps> )] | <int>
```

`<int>` is a natural number

`<id>` is any string starting with character followed by characters or numbers (that is disjoint from the keywords)

- Each program should have a function called **main** with no arguments and no return value.
- All other functions should have one or two return values.
- You should support the following built-in functions, which accept two integers and returns an integer:
  - **plus,** which adds the arguments;
  - **times,**which multiplies the arguments;
  - **minus,** which subtracts the arguments;
  - **divide,** which divides the arguments.
- All the boolean operators (**less, lessEq, eq, nEq)** are binary
- The **read** command assumes that the given variable is an integer.

The following example illustrates a valid FUNC program (more examples later in the document):

```
method pow(x, y) vars i, res
begin
   res := x;
   i := 1;
   while less(i,y)
   begin
      res := times(res,x);
      i := plus(i,1);
   endwhile
   write res;
   return res;
endmethod

method main() vars a, b, x
begin
   a := 5; b := 2;
   x := pow(b,a);
   if  eq(x,32) then write 1; else write 0; endif;
endmethod
```

PART 1: Front End: 10 Marks [Provisional Deadline: Week 10]

Your task for Part 1 is to implement the front end of a compiler for FUNC. To complete this part you need to understand lexical analysis, syntax analysis and abstract syntax trees. Specifically, your task is:

1. **To produce A FLEX file** with a suitable token representation for the FUNC language (3 marks) and to generate a lexical analyser for FUNC.
2. **To implement a recursive descent parser** for the grammar given above (4 marks), using (1)
3. **Produce an AST** with a suitable representation of the nodes generated by the parser (3 marks).

You can do (1), (2) and (3) either in C, in Java, or in Python.

For partial solutions, a suitable number of marks will be deducted.

PART 2 will be released on Vision in due course.