Mini PASCAL Language Specifications

1. **Keywords:** program, integer, real, boolean, char, var, to, downto, if, else, while, for, do, array, and, or, not, begin, end, read, and write. The keywords are not case-sensitive. (Include any keywords used in the below, but missing from the list.)

2. **Variables or Identifiers:** The name of a variable can be composed of letters, digits, and the underscore character. It must begin with a letter. The variable names are not case-sensitive, so uppercase and lowercase letters mean same here. However, the keywords are not allowed to use as a variable names.

   All variables must be declared before we use them in the program. All variable declarations are followed by the *var* keyword. A declaration specifies a list of variables, followed by a colon (:) and the type. Syntax of variable declaration is

```
var
    variable_list : type;
```

   Here, *variable_list* is a list of variables separated by a comma (,) and *type* from the list $\{char, integer, real, boolean\}$.

   Example:

```
var
      age , weekdays : integer;
      taxrate , net_income: real;
      choice , isready: boolean;
      initials , grade: char;
```

   Note that at the variable declaration, assigning a value to one or more variables is not allowed.

3. **Operators:** An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. We allow the following types of operators:

   - Arithmetic operators:
     $+$ (*addition*) ,
     $-$ (*subtraction*),
     $*$ (*multiplication*),
     / (*division*, return real value),
     % (*reminder*, returns integer type)
   - Relational operators:
     $=$ (equals, comparison operator)
     $<>$ (not equal to)
     $<, >, <=, >=$ (these operators have usual meaning)
   - Boolean operators:
     and : boolean AND operator, if both the operands are true, then condition becomes true.
     or : boolean OR Operator. If any of the two operands is true, then condition becomes true.
     not: boolean NOT Operator. Used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.

4. **Statements:** In the program, a statement can be of any of the following:

   (a) **Read and Write statements:**

   - **write**: it prints the text or values of variables on the screen. The syntax is given below:

     ```
     write("text");    // prints the text on the screen.
     write(variable_list);  //prints the values of the variables on the screen.
     ```

     Note that in *write*("*text*"), text is just a sequence of characters, it will not have any meaning. Hence, do not tokenize the text.
     example:
     write("Welcome to CS F363");
     write(day, age, cost); //day, age, cost are variables and the type of these variables need not be the same.

   - **read**: takes the value from the user as an input and the syntax is given below:

     ```
     read(id);  //where id is a varaible in the program
     ```

   (b) **Assignment statement:** To assign a value to a variable, follow this syntax:

   ```
   variable_name := expression;
   ```

   here the *expression* is a single value or a variable, or an arithmetic expression over constants/variables with arithmetic operators mentioned above.

   (c) **Block of statements:** A set of one or more statements is consider as a block and each block starts with **begin** and ends with **end**.

   ```
   begin
   statement_1;
   ....
   ......
   statement_k;

   end
   ```

(d) **Conditional statements:** We allow if-then and if-then-else statements as in PASCAL language.

- **Simple if**: *if condition then S*;
  where *condition* is a Boolean or relational expression and S is a compound statement (block of statements). See an example below:

  ```
  i:= 10;

  if i > 10 then
          begin
            i :=10;
            i := i - 1;
            write(i);
          end;
  ```

- **if-then-else**: *if condition then* S_1 *else* S_2;
  where *condition* is a Boolean or relational expression, and S_1 and S_2 are compound statements (block of statements). Note that there is no ; (semicolon) after S_1.
  example:

  ```
  i:= 10;

  if i > 10 then
          begin
            i :=10;
            i := i - 1;
            write(i);
          end
  else
          begin
            i:=20;
            write(i);
          end;
  ```

(e) **Looping statements:** We allow while-do and for-do loops as in PASCAL. For the sake of simplicity, we do not consider nested loops.

- **while-do**: *while condition do* S;
  where *condition* is a Boolean or relational expression and S is a compound statement (block of statements). See an example below:

```
while (number>0) do
        begin
                sum := sum + number;
                number := number - 1;
        end;
```

- **for-do**: $for\ variable{-}name\ := \ initial\_value\ to\ [downto]\ final\_value\ do\ S$;
  Where, the $variable - name$ specifies a variable of ordinal type, called control variable or index variable; $initial\_value$ and $final\_value$ values are values (or arithmetic expressions) that the control variable can take; and S is the body of the for-do loop that is a group of statements / block statements. See examples below:

```

b:=20;

 for a := 10 to b+10  do

begin
       write("value of a: ");
       write(a);
end;
```

```

b:=20;
a:=0;
//example to illustrate downto
for a := a+5 downto b-15  do

begin
        write("value of a: ");
        write(a);
end;
```

5. **Program structure:** the program starts with keyword "program" followed by the name of the program and terminates with ; (semicolon). Next is the variable declaration section, then followed by main program block. The main program block starts with keyword "begin" and ends with keyword "end" followed by a period (.).

```
program name_of_the_program;

var
        variabl_list:type;    //declaration of varaibles

begin // main program block starts





end. // the end of main program block
```

Here, *name_of_the_program* follows the rules of variables.

Example:

```
program AddTwoNumbers;


var
        num1, num2, sum: Integer;

begin
        Write("Enter the first number: ");
        read(num1);

        Write("Enter the second number: ");
        read(num2);

        // Perform addition
        sum := num1 + num2;

        // Display the result
        Write("The sum is ");
        write(sum)
end.
```

6. **Arrays:** we consider only one-dimensional arrays as in PASCAL and we consider a static declaration of array. The syntax is given below:

```
aray_name: array[c1..c2] of type;
```

where *array_name* is an identifier (variable), *array* and *of* are keywords. Further, $c1$ and $c2$ are integer constants such that $c1 \leq c2$, and $type \in \{integer, char, real, boolean\}$. See an example in the below:

```
program ArraySum;

var
        numbers: array[1..10] of Integer;
        i, sum: Integer;

begin
        // Read 10 values into the array
        writeln("Enter 10 integer values: ");
        for i := 1 to 10 do
          begin
                read(numbers[i]);
          end;

        // Calculate the sum of the values in the array
        sum := 0;
        for i := 1 to 10 do
          begin
                sum := sum + numbers[i];
          end;

        // Display the sum
        write("The sum is : ");
        write(sum);


end.
```