While Interpreter User Manual

Authors:

Behrooz Omidvar Tehrani, Jander Nascimento (MoSIG M1 Students)

We have made a “While Interpreter” in the mode of “Dynamic Bindings” for variables and “Static Bindings” for procedures.

Obviously in each situation, we need a table as “Sigma” carrying variables declared and their value in the state “Sigma”. Also, we need a table for procedures to store their name, their body and the environment that they are being defined in.

To use this implemented language, follow these rules:

Declaring Variables

To declare a variable, use the reserved word **Var** and the identifier for your variable. Start declaring variables by the keyword **declare**.

```
declare
var x;
var y;
```

Assigning Value

To assign value to a variable, just write the variable name, then the assignment symbol, and at last your desired value.

```
X := 4;
```

Also you can assign a variable to another variable.

```
X := Y;
```

While Interpreter User Manual

Authors: *B. Omidvar Tehrani, J. Nascimento (MoSIG M1 Students)*

Creating a Block

To create a block, just use the keyword **Begin**. To end a block, use the keyword **End**.

```
Begin
.
.
.
End
```

Hint: The line before **End**, should not be ended by Semi-colon “;”.

Using Conditions

To have your own condition, use keyword **If**, followed by a Boolean condition in parenthesis, then the keyword **Then**, and the operation(s) to be done if the condition is evaluated as true. Also you can continue the structure by keyword **Else** followed by the operation(s) to be run if the condition is evaluated as false.

```
if (CONDITION) then OP1 else OP2 ;
```

```
if (y > 5) then y := 6 else z := 6 + y ;
```

Junction

You can join two Boolean statements together by junction (AND). You can join two Boolean statements together by junction using the symbol **/\**. The result will be true if and only if two Boolean statements are true. In other conditions, it will go false.

```
if (x = 1 /\ x > z) then x := x+2 else z := 0 ;
```

Performing Operations (Addition, Multiplication)

You can add or multiply integer numbers and integer variables in this program easily by using their original symbol.

As an example, the following command, will add 2 to “u” and place the final value in “t”.

```
t := 2 + u;
```

As an example, the following command, will place the 2nd power of “X” into “z”.

```
z := X * X;
```

Comparison (Equality, Inequality)

You can also check the equality or inequality of integer values and variables that will output true or false as Boolean.

- Hint: Use “:=” for assignment and “=” for equality comparison.

For example, if we consider X has the value “5”, the OP1 will be run in the following code:

```
if (X < 9) then OP1 else OP2
```

Negating (NOT)

You can make the result of Boolean statement reversed by using symbol “**^**”.

For example, if we consider X has the value “5”, the OP1 will be run in the following code:

```
if (^X > 9) then OP1 else OP2
```

Repeatable structure (WHILE)

A repeatable structure is defined in our language called “While” to repeat some commands as far as a special command is evaluated as true.

```
while (CONDITION) do OP od
```

Operations will be limited between two keywords “do” and “od”. For example, if we consider X has the initial value “5”, the OP operations will be run for 4 times in the following code:

```
while (X > 1) do x := x - 1; od
```

Declaring procedure (with or without parameter)

If you want to declare a procedure to have the ability to call it somewhere later, you can sue the keyword **proc**.

```
proc proc_name() is OP
```

Procedures can have zero or more parameters.

```
proc proc_name1(param 1,param 2) is OP
```

For instance, we make a procedure called “Q” that gets a number and adds it to two.

```
proc Q (X) is
begin
X := X +2;
end
```

Calling a procedure

The produced procedure can be called somewhere. To call a procedure, just use the keyword **call**.

```
call proc_name();
call proc_name1(X);
```

Here, we have put some outputs of our implemented language for some examples.

Example 1 (ex1.w)

Source

```
declare
var x
begin
x := 1
end
```

Output

```
<assign variable=x value=1 />
<environment type=variable >
      Name:x Value:1
</environment>
```

Example 2 (ex4.w)

Source

```
declare
var x  ;
var z
begin
x := 200  ;
z := 2 * x
end
```

Output

```
<assign variable=x value=200 />
<assign variable=z value=400 />
<environment type=variable >
      Name:x Value:200
      Name:z Value:400
</environment>
```

Example 3 (ex5.w)

Source

```
declare

var x  ;
var z
begin

x := 200  ;
z := 2 * 4 + 5

end
```

Output

```
<assign variable=x value=200 />
<assign variable=z value=13 />
<environment type=variable >
      Name:x Value:200
      Name:z Value:13
</environment>
```

Example 4 (ex6.w)

Source

```
declare

var x  ;
var z
begin
x := 2;
if ( x = 1 ) then z:=5 else z:=0

end
```

Output

```
<assign variable=x value=2 />
<executing condition='x 1  =  = ' going_to='else'>
      <assign variable=z value=0 />
</executing>
<environment type=variable >
      Name:x Value:2
      Name:z Value:0
</environment>
```

Example 5 (ex10.w)

Source

```
declare
var x ;
proc q() is
declare
var w
begin
x := 50;
w := 72
end
begin
x := 4 ;
call q ()
end
```

Output

```
<assign variable=x value=4 />
<environment type=variable >
      Name:x Value:4
</environment>
<call name=q >
      <environment type=variable >
```

```
            Name:x Value:4
      </environment>
      <assign variable=x value=50 />
      <assign variable=w value=72 />
      <environment type=variable >
            Name:x Value:50
            Name:w Value:72
      </environment>
      <environment type=variable >
            Name:x Value:50
      </environment>
</call>
<environment type=variable >
      Name:x Value:50
</environment>

<environment type=procedure >
      Name:q addr:164966904
</environment>
```

Example 6 (ex13.w)

Source

```
declare
var x ;
proc q() is
declare
var w
begin
x := x + 1 ;
w := 50
end
begin
x := 1 ;

while ^(x > 4) do
call q ()
od

end
```

Output

```
<assign variable=x value=1 />
<environment type=variable >
      Name:x Value:1
</environment>
<call name=q >
      <environment type=variable >
            Name:x Value:1
      </environment>
      <assign variable=x value=2 />
      <assign variable=w value=50 />
```

```
        <environment type=variable >
                Name:x Value:2
                Name:w Value:50
        </environment>
        <environment type=variable >
                Name:x Value:2
        </environment>
</call>
<environment type=variable >
        Name:x Value:2
</environment>
<call name=q >
        <environment type=variable >
                Name:x Value:2
        </environment>
        <assign variable=x value=3 />
        <assign variable=w value=50 />
        <environment type=variable >
                Name:x Value:3
                Name:w Value:50
        </environment>
        <environment type=variable >
                Name:x Value:3
        </environment>
</call>
<environment type=variable >
        Name:x Value:3
</environment>
<call name=q >
        <environment type=variable >
                Name:x Value:3
        </environment>
        <assign variable=x value=4 />
        <assign variable=w value=50 />
        <environment type=variable >
                Name:x Value:4
                Name:w Value:50
        </environment>
        <environment type=variable >
                Name:x Value:4
        </environment>
</call>
<environment type=variable >
        Name:x Value:4
</environment>
<call name=q >
        <environment type=variable >
                Name:x Value:4
        </environment>
        <assign variable=x value=5 />
        <assign variable=w value=50 />
        <environment type=variable >
                Name:x Value:5
                Name:w Value:50
        </environment>
```

```
      <environment type=variable >
            Name:x Value:5
      </environment>
</call>
<environment type=variable >
      Name:x Value:5
</environment>
<environment type=procedure >
      Name:q addr:163951176
</environment>
```