PROJECT REPORT
MINIOO – INTERPRETER

Submitted by: Bharathi Priyaa Thangamani

For the class

Honors Programming Languages(spring 2016)

OBJECTIVE:

To develop a minioo interpreter in OCaml which satisfies the grammar specified in Syntax and Semantics of MiniOO.

SPECIFICATIONS:

- The grammar that I have used in my interpreter is the same as mentioned in the Syntax and Semantics of MiniOO.
- Formal semantics definition are also same as specified
- Data types like CmdType, expr Type , stack, heap, Environment, Closure, Conf etc. exactly match the definition listed in the project specification.
- Additional Arithmetic operators like (*,+,) have also been implemented.

MODULES DEFINED:

The following main modules are defined in the OCAML YACC file.

1) **PRINT ABSTRACT SYNTAX TREE:**

   Print_ast_tree (program, indent) - Takes the (program string, int)as input and prints an abstract syntax tree. An integer value for indent is also passed.

2) **STATIC SEMANTICS CHECKER:**

   static_check(program) - Takes the program string and does a check on the semantics of the program. It throws an error if there the program is not semantically correct.

3) **MINIOO INTERPRETER:**

   Interpret(program) - Interpret is the main function which takes the program string and interprets it. During this process, interpret calls a series of recursive functions, which evaluate expressions and maintain the state of stack and heap.

   Object which live in the scope of the interpreter: A Conf object of program string, stack , heap.

EXAMPLES

EXAMPLE 1:

var x ; {malloc(x) ; x.F=100}

Explanation: This is a simple example which illustrates malloc and the assignment of location expressions.

In the absence of malloc(x), the interpreter indeed throws an error for any location expression.

AST TREE:

```
PROGRAM
Declaration:x
  Command List:
       Sequential control:
       Command list 1:
            Dynamic memory allocation:
                 Malloc:Identifier:x

       Command list 2:
            Field Assignment:
                 Expression:Identifier  x
                   Operator .
                 Expression : Field =F
                   Operator =
                 Expression : Numeric literal 100
```

INTERPRETER RESULTS

```
---------------------- Stack
 [0] Declare (x, 0)
---------------------- Printing Heap contents
Heap[0]
val=Heap [1]
  Heap[1]F=100
```

Example 2:

```
var x; {x=10; while x< 20 x = x +1}
```

AST TREE:

```
    Declaration:x
      Command List:
           Sequential control:
           Command list 1:
                Variable Assignment:
                     Identifier:x
                     Expression:
                          Numeric literal 10
           Command list 2:
                While Block:
                     Boolean Expression
                          Expression:
                               Identifier  x
                          Boolean Operator : <
                          Expression:
                               Numeric literal 20

                     Variable Assignment:
                          Identifier:x
                          Arithmetic Expression:
                               Expression:
                                    Identifier  x
                               Operator: +
                               Expression:
                                    Numeric literal 1
```

INTERPRETER RESULTS

```
-------------------- Stack
 [0] Declare (x, 0)
------------------- Printing Heap contents--------------
Heap[0]
   val=20
```

Example 3:

```
var p;
     {p = proc y: if y < 6 p = 4 else p = 3;
      var x;
          {x= 10;
          {p = 1; var z; z=20}}
}
```

Explanation: The above program can be shortened as below for better understanding :

var p ; Sequential{ PROC CMD1 ; var x ; Sequential {CMD2,CMD3}}

We observe here that there are 3 variables declared p,x,z. Scope of "p" is global whereas scope of variables "x" and "z" is restricted to the scope of their respective sequential control.

Also note that the procedure y is defined within the sequential control C1. So any reference to y outside its scope will result in an error. Also, there is no call to the PROC y being made within the scope and hence it is will not be executed at all.

At the end of the execution, we expect the result of p=1, x=10 and z=20 stored in the heap. Also, we can observe from the full output that there was no Call frame in the stack, which proves that there was no call made to remote procedure call.

The abstract syntax tree for the above example is :

```
Declaration:p
      Command List:
           Sequential control:
           Command list 1:
                Variable Assignment:
                     Identifier:p
                     Recursive procedure :y
                          If Block:
                               Boolean Expression
                                    Expression:
                                         Identifier  y
                                    Boolean Operator : <
                                    Expression:
                                         Numeric literal 6

                               Variable Assignment:
                                    Identifier:p
                                    Expression:
                                         Numeric literal 4
                          Else Block:
                               Variable Assignment:
                                    Identifier:p
                                    Expression:
```

```
                                        Numeric literal 3
          Command list 2:
               Declaration:x
               Command List:
                    Sequential control:
                    Command list 1:
                         Variable Assignment:
                              Identifier:x
                              Expression:
                                   Numeric literal 10
                    Command list 2:
                         Sequential control:
                         Command list 1:
                              Variable Assignment:
                                   Identifier:p
                                   Expression:
                                        Numeric literal 1
                         Command list 2:
                              Declaration:z
                              Command List:
                                   Variable Assignment:
                                        Identifier:z
                                        Expression:
                                             Numeric literal 20
```

THE STATE OF STACK AND HEAP AT THE PENULTIMATE EXECUTION STEP IS

```
-------------------- Stack
 [0] Declare (z , 2)
 [1] Declare (x , 1)
 [2] Declare (p , 0)
-------------------- Heap
Heap[0]
val=1
Heap[1]
val=10
Heap[2]
val=lnull
```

EXAMPLE 4: ATOM AND PARALLEL EXECUTION

```
"var x; {atom({x=10 ; x=x+1}) ||| x = 1}" | ./minioo


SYNTAX TREE:
        PROGRAM
            DECLARATION
            COMMAND LIST
                SEQUENTIAL CONTROL ->
                    Parallelism block
                        C1: ATOM
                        C2: CMD
```

Explanation: Note that we have an Atom within a Sequential control which has concurrent operator.
Parallelism block may select either one of the two commands(C1,C2). In case, Atom is choosing, the execution of Atom Block should completely fully before returning control to the Parallelism block. We observe the same results below:

```
Stack & Heap Contents; (Note that After Atom got executed,
X was set to 1)
-------------------- Stack
 [0] Declare (x, 0)
-------------------- Heap
Heap[0]
val=1
```

EXAMPLE 5: PARALLEL EXECUTION WITHOUT ATOM

```
    "var x; {{x=10 ; x=x+1} ||| x = 1}"
```

Syntax tree:

```
        PROGRAM
            DECLARATION
            COMMAND LIST
                SEQUENTIAL CONTROL ->
                    Parallelism block
                        B1 :    CMD LIST[C1,C2]
                        B2 :    CMD[C3]
```

Note that in this case, we do not have Atom and hence commands in the parallel block can be executed in any order. There are three commands namely (C1,C2,C3). One order of execution is C1,C3,C2 according to parallel execution.
The expected value of Heap in this case would be 2; Please find below the order in which the values in Heap is changed:

```
-------------------- Stack
 [0] Declare (x , 0)
-------------------- Heap
Heap[0]
val=lnull
-------------------- Stack
 [0] Declare (x , 0)
-------------------- Heap
Heap[0]
val=10
-------------------- Stack
 [0] Declare (x , 0)
-------------------- Heap
Heap[0]
val=1
-------------------- Printing Final Stack  --------------------
-------------------- Printing Final Heap contents-------------
Heap[0]
val=2
```