Documentation for 6.S081 Assignment 2 – Ali Zartash

NOTE: This also covers the design of the paser and lexer which was assignment 1 (as it was not graded)

Design

To approach the first part of the project, which was the lexer and parser, I decided to use the Flex and Bison instead of the Antlr4 library. The reason for this is that I wanted to write my own AST using a pure virtual base class (for example Statement and Expression) and then many concrete subclasses such as Return statements or Binary Expressions, or even constants. With Flex and Bison, it was not much difficult to get the AST ready. I also think a major factor in my decision to choose Flex and Bison was that there was a book available online on Flex and Bison and just reading the first couple of pages in the introductory chapter gave me a good tutorial of how to use it. The lexer or scanner was implemented in the given lexer.lex starter file and it was not a lot of work, just writing down the regular expressions and deciding what tokens should be used. I used all nonterminals as tokens in the grammar. Next, the grammar had to be implemented in the parser.yy file which was in the bison format (BNF) and this was also not extremely complicated. I used new constructors at the leaves of the AST and whenever the grammar was at an intermediate node, it would just combine two of the lower nodes into an expression. For example, if I encountered the integer constants, I would create a new Integer object. Then if I see a Binary Expression, I would combine the left and right operands into a binary expression object. This part also involved the implementation of a PrettyPrinter which I tried to make as similar to standard code style. The parser passed all public and private test cases.

The second assignment (Interpreter) was much more difficult. There is one public test (the last one) that my Interpreter is still not passing, but it is an extremely complicated test case and it is difficult for me to see where it is failing to work correctly in my implementation of the interpreter. This assignment involved many more decisions than the previous one.

I decided to use the Visitor Pattern to implement the visitor. This was suggested by the TAs and also I already had practice implementing a visitor in the PrettyPrinter. In particular, the visitor class was already created (abstract class). The files that I ended up making were as follows:

AST.h: This was the header file of the AST. I forward declared the Visitor class in this because the Visitor class is particular to this AST and also the AST had to declare accept(Visitor& v) methods which required forward declaration. I actually implemented most of the methods within the header file (except for the accept methods because of a reason given below).

AST.cpp: This was the implementation file for the AST. I actually implemented most of the AST within the header – this was what I had already done in assignment 1 because the AST itself is not a very complicated class. However, I had to remove the accept methods and put them in the AST.cpp file as they needed the complete type of Visitor to be declared to be able to call the visitor.visit methods. Thus I put these methods in the .cpp file and then imported Visitor.h.

Value.h: This is the header (and implementation) file for the Value data types of MITScript which are String, Boolean, Integer, None, Function, StackFrame, and Record. It also includes a StackFrame data type which is also a Value. I decided to make the StackFrame a Value because it gives a nice abstraction – the stackframes themselves are heap allocated addresses thus it made sense to keep the stackframe

as a value as well. I also used an abstract Value type which I then made all these Value concrete objects as subclasses of.

Visitor.h: Interface for the Visitor type.

PrettyPrinter.h: for assignment 1. Has header file as well as the implementation.

Interpreter.h: This is the header file for the interpreter which is a subclass of Visitor. It also includes declarations of the exceptions required by the specification.

Interpreter.cpp: This is the file containing the implementation of the interpreter, which details given below.

As suggested in the Assignment, I used a local state variable Value* rval to keep the return value of the expressions, functions etc. I used a RetrunVal to differentiate between when a function returns or a high level return statement is called and when a value is simply evaluated. I also defined a local helper method called isInstanceOf to determine the type of Value that I was dealing with to dynamically determine the behavior of the program.

The state variables of the interpreter included a stack to which function calls push on and pop off of, the stack had stackframes which was defined in the Value.h file. The stackframe used internally a map (standard map) from strings to values. Most of the other implementation was the same as the specification and suggestions as the TAs.

Extras

I think one extra design decision that I made was to add “special” nodes to the AST (which are never parsed). These special nodes are used in the predefined functions. For example, print() – I made a new AST node called SPrint and then used the visitor pattern on this AST to not return anything but redirect output to cout. This SPrint node was basically a statement thus easily went inside the Function declaration and worked just as a regular function. I think many people would have implemented it this way – I do not know of any other way to do it.

Difficulties

I think memory leaking is a big issue for me. I am using pointers in a lot of places and I don’t think I free memory for each of the new calls I am making which is particularly tricky in cases where there is recursive structure. I think a good idea would be to use virtual destructors then define custom destructors for all the concrete subclasses and then delete all pointers when the destructors are called. Debugging the interpreter was extremely hard – I also thought that the assignment was too long. But it was extremely interesting.

Contribution

This was an individual assignment so I did all the work myself.