Jackie Ramsey

CPSC 326

4/29/23

Project Report

The extension chosen for this project was to implement switch statements as a new MyPL construct. Similarly to C++, switch statements are used to execute certain code blocks that were identified by the switch expression. The way it works is that the switch expression is evaluated once. This expression's value is then compared to the values of each case. If there is a match, then the code block of that case will be executed. If there is no match, then either no code blocks will be executed, or the default code block will be executed. The default case is optional and is used like an else statement if the switch expression does not match a case. There is also a break keyword that can be optionally used in the case statements. If the program comes across the break keyword, it breaks out of the switch block. If no break keyword is found, it continues on.

Examples:

```
switch(1) {
  case 0:
    stmt
  case 1:
    stmt
  default:
    stmt
}
```

```
switch(1) {
  case 0:
    stmt
    break
  case 1:
    stmt
  default:
    stmt
}
```

```
switch(1) {
  case 0:
    stmt
    break
  case 1:
    stmt
}
```

Since this project was adding a new MyPL construct it involved all aspects of the pipeline. This process started from Token creation all the way to code generation. For lexical analysis, the following tokens were added: SWITCH, CASE, BREAK, DEFAULT, COLON. These tokens were then implanted into lexer.cpp to create the token sequences to be passed for syntax analysis. Within simple_parser.cpp, three functions were created to implement the grammar of switch statements.

Grammar:

Next, the ast_parser.cpp, print_visitor.cpp, and semantic_checker.cpp were implemented so that the abstract syntax tree could be used for code generation. The first thing that had to be done was to create classes so that the visitor could be used, etc. Two classes were created: SwitchStmt, CaseStmt. Lastly, the code_generator.cpp was implemented so that the ASTs could be converted to the VM instructions.

This extension was successfully completed and all components work correctly. This can be seen through the unit tests and running the example files over –lex, --parse, --print, --check, --ir. If I had more time on this project, I would have made it so that the switch statement could take in an expression instead of a constant variable. I went the route of doing a constant variable due to the previous structure of the ast classes.

This extension was tested in two ways. The first was through unit tests. These unit tests were grouped into the different parts of the pipeline. This verified that all parts were working correctly as the pipeline was built. I tested for cases such as empty cases, no matching of cases, default cases, wrong types, and much more. The second way this extension was tested was by creating some example MyPl programs and executing them on the finished MyPL implementation.

Instructions for building and running the extension:

1. *cmake .*
2. *make*
3. testing
    a. *./project_tests*
    b. *mypl.cpp example_file.mypl*

https://youtu.be/NLSOU5DygFM