Evan Shoemaker

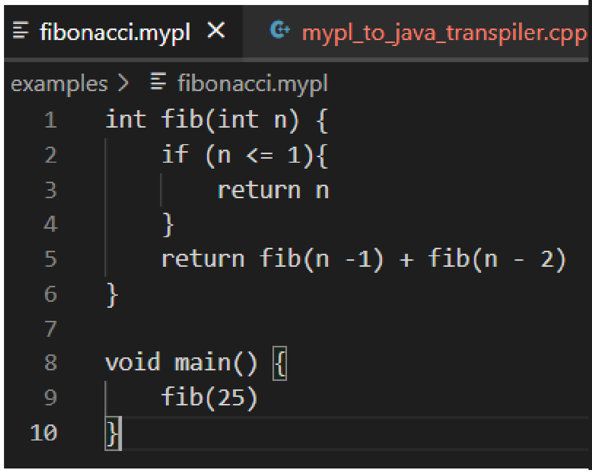
CPSC326

May 12, 2023

MyPL to Java Transpiler

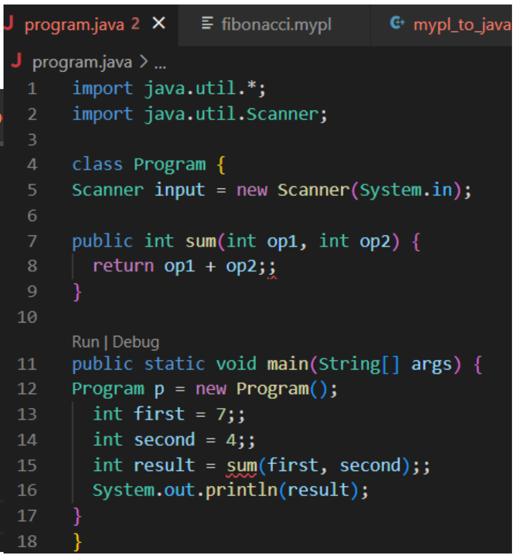
I implemented a transpiler that converts MyPL source code to Java. The goal of this project was to allow any MyPL program to be converted to Java, compile, and run in Java without issue. One example of a program one could convert is:

MyPL source code: **Resulting Java program:**



```
int fib(int n) {
    if (n <= 1){
        return n
    }
    return fib(n -1) + fib(n - 2)
}

void main() {
    fib(25)
}
```



```
import java.util.*;
import java.util.Scanner;

class Program {
Scanner input = new Scanner(System.in);

public int sum(int op1, int op2) {
  return op1 + op2;;
}

public static void main(String[] args) {
Program p = new Program();
  int first = 7;;
  int second = 4;;
  int result = sum(first, second);;
  System.out.println(result);
}
}
```

My project does a variety of different things to make a MyPL program into a (close to) compilable and runnable Java program. It converts incompatible tokens to their applicable Java equivalents. This includes converting "and" to "&&", "or" to "||", "bool" to "boolean", "string" to "String", etc. It also converts (some of) MyPL's built-in functions into the corresponding Java ones. For example, it converts concat()'s syntax to use Java's syntax (param1.concat(param2), not concat(param1, param2). It also converts (partially) addresses the fact that length() in MyPL can be used for both strings and arrays. In Java, the length of an array is a property, not a method. However, to get the length of a string, one uses a method called length(). My implementation adds "()" for string literals and assumes its an array otherwise, just using length(). The case where a string is stored in a variable is something I still need to do. Additionally, my implementation addresses importing a few packages (utils and Scanner) to support the equivalent Java functions analogous to MyPL's built in functions. Since there are no structs in Java, my implementation converts them to classes. Because Java arrays require a pair of brackets in the datatype before the variable name (in**[]** xs = new int[10]), my implementation adds those. My implementation also adds a driver class (Program) as required in Java, and defines all other functions as public member functions of this class. I also added semicolons where applicable. On a high level, these changes required modifying the lexer a little for token conversions like “and” to “&&”, and print_visitor.cpp extensively for almost everything else.

I successfully completed struct conversion, function conversion, if/for/while statements, adding semicolons, the concat() built-in function, arrays (asides from a few edge cases), and basic token conversion (as previously mentioned). What remains left to be done or is mostly complete but not quite fully functional includes the remaining built-in functions, removing duplicate semicolons, addressing function calls within main (putting “p.” before every function call in main after instantiating Program p so it doesn’t throw a “non-static method referenced in a static context” error) without doing so within the bodies of other functions, and being able to create an array of structs. If I had more time, I would fix these issues as well as make some of the solutions to converting built-in MyPL functions more flexible. For example, instead of instantiating a global scanner object regardless of whether the MyPL source code has a call to input() or not, I would seek to only create the Scanner object if it’s needed. Additionally, I would write more unit tests and figure out how to test the performance of the original MyPL program against the Java transpilation.

I developed unit tests testing the aspects I changed above. My unit tests tested array conversion, struct conversion, and empty main function, a simple sum function, simple token conversions, and some of the helper functions. I also tested my implementation on the example pretty print files from homework 4, along with some of my own (simple-sum.mypl and fibonacci.mypl).

Running and Building:

To run and build my project, first clone the repo from github.

Navigate into the repo’s root directory, then type

```
cmake .
```

install cmake if you need to. If it builds with no issue, type

```
make
```

once the build is complete, you are ready to write a MyPL program to transpile.

Once you have written your desired program, use the –java option and do the following as seen in the line below:

```
./mypl --java yourProgram.mypl >> program.java
```

Note that that Java program MUST be called “program.java”.

Finally, compile and run the Java program:

```
javac program.java
```

```
java program.java
```

Note that my implementation is incomplete is incomplete and will result in some compile errors that need to be fixed. Some of the most common ones will be deleting extra semicolons and adding “p.” to function calls in main.

Demo Video Link:

https://youtu.be/wI15UtY4QBw