IN2009 Coursework 2022-23 Part 2

Submission

Submit your solution as ***a single*** *Java source file* **`LP23xCompiler.java`**

You are required to write a compiler which generates SSM assembly code for the language LP23x. You are provided with a formal grammar for LP23x and a prototype compiler, as well as grammars for two simpler variants of the language (LP23xs and LP23xef: these correspond to the language subsets assessed in tasks 1 and 2, respectively). *Note that these three grammars are essentially solutions to the tasks set for Part 1 of the coursework, except that variables are declared explicitly (this makes life slightly easier for a compiler writer).*

You are provided with some test inputs (the expected output is given as a header comment in each file). This is *not* an exhaustive test suite and you are free to write your own tests. Assessment will be carried out by automated testing using these and other tests.

The semantics of the language is essentially standard and most language constructs have an obvious counterpart in languages like Java and C. Key differences and similarities:

- LP23x is untyped. All expressions evaluate to a 32-bit signed integer.
- LP23x has three types of variables: global variables, formal parameters, and local variables. Global variables are statically allocated. Formal parameters and local variables are stack-allocated. The scope of a formal parameter or local variable is the body of the function in which it is declared. Formal parameter and local variable names are allowed to clash with global variable names (creating a hole in the scope of the global variable) but all local variable and formal parameter names within a given function must be distinct.
- In Boolean constructs, 0 is treated as False and all other values are treated as True. Comparison and logical operators all evaluate to either 0 or 1.
- If a function call terminates without executing a return statement, the function returns 0 by default. Return statements are also allowed in the main program block, where the effect is to halt execution (the return expression is evaluated but its value is ignored in this case).
- As in C and Java, the `&&` and `||` operators have "short circuit" semantics. When the first argument of `&&` evaluates to False (0), the second argument is not evaluated and the operator returns 0 immediately. Dually, when the first argument of `||` evaluates to True (any non-zero value), the second argument is not evaluated and the operator returns 1.
- All other operator and function arguments are evaluated left to right and passed by value (fully evaluated before the call).
- New arrays are created using expressions of the form `malloc[`*e*`]` where the value of *e* determines the size of the new array. The behaviour of `malloc` is to allocate a block of heap memory of the specified size, initialise its contents to all zeroes, and return its start address. Arrays in LP23x are like arrays in C (*not* Java): arrays do not have a length field and there is no run-time bounds checking. If a program attempts to use a non-array value as an array, or to access an array out of bounds, the expected behaviour is undefined (none of the provided test inputs do these things).

Start by downloading and unzipping `IN2009CourseworkPart2.zip`. This is a raw project folder. Open it in IntelliJ and configure it in essentially the same way as for the GettingStarted project (your compiler will rely on both SBNF.jar and SSM.jar and you will need to set your CLASSPATH so that you can run your compiler in a terminal window).

You are required to implement your compiler by completing the provided prototype LP23xCompiler class. ***Do not*** *modify the name of this class or add a package declaration (adding a package declaration changes the fully-qualified class name).* ***Do not*** *modify the signatures of either the constructor or the compile method.* ***Failure to comply with these instructions will cause all assessment tests to fail, resulting in a mark of zero.***

The prototype contains a number of convenience methods for emitting assembly code and for generating safe and fresh names. You should not need to modify these methods (but you are free to do so). Use fresh names for labels in your generated code (remember that all label names must be unique). The purpose of

To test your compiler, compile a source program, execute the generated code and check the output. For example (assuming current directory is the data folder):

```
java LP23xCompiler lp23x.sbnf tests/lp23xef/fac.lp23x fac.ssma
java Run fac.ssma
```

The expected output for this test is `362880`.

An LP23x program is said to be *well-formed* if no scoping rules are violated, no undeclared variables are used, no undefined functions are called, and all function calls have the correct number of arguments. When writing your compiler you can assume that all programs are well-formed (no error checks are required). All the provided test inputs are well-formed.

1. **[50 marks]** Correct compilation of well-formed programs which use only the basic language features, excluding any use of functions, arrays or the increment operator.
2. **[30 marks]** Correct compilation of well-formed programs which use the basic language features plus functions.
3. **[20 marks]** Correct compilation of well-formed programs which use all language features. Note that the SSM provides no support for heap management, so you will need to implement your own scheme for dynamically allocating memory. You are not expected to implement any form of garbage collection. To test that your generated code really is initialising new arrays correctly, pass the `–ranmem` flag when executing your SSM code (this puts random initial values in the machine's memory on start up). For example:

   ```
   java Run -ranmem emptyArray.ssma
   ```

   Check that you have at least version 1.5 of SSM.jar (the `-ranmem` option is not supported by version 1.4).