Started on	Thursday, 19 May 2022, 7:08 PM
State	Finished
Completed on	Friday, 27 May 2022, 1:41 AM
Time taken	7 days 6 hours
Marks	18.24/34.00
Grade	**53.64** out of 100.00

Information

In the compiler super-quiz you will implement parts of a compiler for a small programming language described below and discussed in the lectures. It comprises three quizzes about

1. the scanner,
2. the parser and
3. the code generator.

You will get most of the marks for implementing the compiler and a small part for additional questions about compilers.

The implementation parts of the quizzes are incremental: the parser requires a working scanner and the code generator requires a working parser. You will receive Python 3 templates for each of the three parts. You have to study these programs to understand how they work and you will be asked to extend them.

Feedback about the errors is limited. It is therefore essential that you thoroughly test your programs before you submit them (see below).

Templates and Precheck

Code that you submit to a question must be based on the template given in that question. The precheck goes some way to ensure this, but this remains your own responsibility. Please use the Precheck button before submitting your answer using the Check button. A precheck attracts no marks and no penalty; you can use it any number of times and also after incorrect submissions. If your code fails the precheck, it will fail the full check as well. If it passes the precheck, it may or may not pass the full check. Note that a precheck will not do any testing of your code; even syntactically incorrect submissions based on the provided template may pass it.

Your code should be derived from the given template by modifying only the parts mentioned in the requirements; other changes are at your own risk. Feel free, however, to experiment during the development.

Syntax

The compiler will accept programs with the following syntax:

Program = *Statements*
Statements = *Statement* (`;` *Statement*)*
Statement = *If* | *While* | *Assignment*

If = `if` *Comparison* `then` *Statements* `end`
While = `while` *Comparison* `do` *Statements* `end`
Assignment = `identifier` `:=` *Expression*

Comparison = *Expression Relation Expression*
Relation = `=` | `!=` | `<` | `<=` | `>` | `>=`

Expression = *Term* ((`+` | `-`) *Term*)*
Term = *Factor* ((`*` | `/`) *Factor*)*
Factor = (*Expression*) | `number` | `identifier`

Identifiers contain only lower-case letters. Numbers are represented by non-negative integers in base-10 notation. In the following quizzes you will need to modify this BNF for if-then-else-end statements, write statements, read statements and Boolean expressions. An example of a program using the above BNF extended by read and write statements is:

```
read n;
sum := 0;
while n > 0 do
  sum := sum + n;
  n := n - 1
end;
write sum
```

Testing

You will need to create your own programs for thoroughly testing your compiler.

This means you will need to create a comprehensive collection of test inputs with the aim of covering all parts of the specification and especially any parts of the code you have changed.

If your submission fails for any input, the code certainly contains at least one error. You will need to identify the error and correct it before resubmitting, not just trying some changes to the code. You should not stop looking for errors when you find the first one; there may be others.

Note that testing provides no guarantees. Your program might work for all your tests and still fail some or even most tests on the quiz server. You will need to create tests with a good coverage.

Some methods for finding errors are:

- read and understand all parts of the specification (including this information box);

- understand what each part of the program is supposed to do;
- create test cases covering all parts of the code that have been changed;
- systematically create sequences of characters (possible inputs);
- randomly create possible inputs;
- carefully review parts of the code that have been changed;
- check if all variables are assigned to before they are used;
- check if every expression is defined in all cases: if it is not, try to come up with an input that causes it to be undefined;
- check if all loops and recursions terminate;
- check for every piece of the code: why is it there? can it fail? if so, under which circumstances? is there an input that causes these circumstances?

Testing works best as a combination of being systematic and creative; try to come up with unusual inputs.

Information

By submitting your solution below you confirm that it is entirely your own work.

Your submissions are logged and originality detection software will be used to compare your solution with other solutions. Dishonest practice, which includes

- letting someone else create all or part of an item of work,
- copying all or part of an item of work from another person with or without modification, and
- allowing someone else to copy all or part of an item of work,

may lead to partial or total loss of marks, no grade being awarded and other serious consequences including notification of the University Proctor.

You are encouraged to discuss the general aspects of a problem with others. However, anything you submit for credit must be entirely your own work and not copied, with or without modification, from any other person. If you need help with specific details relating to your work, or are not sure what you are allowed to do, contact your tutors or lecturer for advice. If you copy someone else's work or share details of your work with anybody else, you are likely to be in breach of university regulations and the Computer Science and Software Engineering department's policy. For further information please see:

- Academic Integrity Guidance for Staff and Students
- Academic Misconduct Regulations

Question **1**

Partially correct

Mark 14.54 out of 30.00

Please download the compiler template or the PLY compiler template. You can use either to complete this question; the choice is yours.

1. Copy the changes to the parser template you made in the previous quiz into the compiler template.
2. Read and understand the compiler template. In particular, you need to understand how the symbol table manages identifiers, how labels are used, how the `code` methods of the AST classes work and how the `true_code` and `false_code` methods are used to translate control flow.
3. Read some AST classes to understand the purpose of the method `code`. Implement the method `If_Else_AST.code`.
4. The final requirement considerably extends the compiler. You should only attempt it if you have a good understanding, in particular of the methods `false_code` and `true_code`. So far the compiler only supports comparisons of expressions as conditions. The goal is to extend these by the logical connectives `and`, `or` and `not`. Use the following syntax:

   BooleanExpression = *BooleanTerm* (`or` *BooleanTerm*)*
   BooleanTerm = *BooleanFactor* (`and` *BooleanFactor*)*
   BooleanFactor = `not` *BooleanFactor* | *Comparison*

   BooleanExpression replaces *Comparison* in the conditions of if-then-end, if-then-else-end and while-do-end-statements. Introduce the necessary tokens, abstract syntax tree classes and parsing functions. You will need to adapt the existing parser functions that use conditions. The new `false_code` and `true_code` methods should implement short-cut evaluation. There is no need to store Boolean values in variables or on the stack.

The compiler is called to translate the input program to a Java assembly, which can be translated to a Java class file.

On submission your code is tested with several programs and, for each program, several runs with different inputs. Tests are carried out using the following procedure, which you should reproduce using your own programs and inputs.

1. Your compiler is run using Python 3 with the program passed via standard input. The Java assembly is expected via standard output. Assuming your compiler is in the file `compiler.py`, the program is in the file `program`, and the Java assembly goes to the file `Program.j`, this amounts to calling
   `python3 compiler.py < program > Program.j`
   An error at this stage is indicated by
   `Error in compiling program X (HINT)`
   `X` is an identifier of the program and `HINT` is a brief description of the constructs it uses.
2. If stage 1 was successful, the Java assembly is translated to a Java class file using the Jasmin assembler, see http://jasmin.sourceforge.net/. Jasmin is in the file `jasmin.jar`. The call is
   `java -Xmx100m -jar jasmin.jar Program.j`
   This generates the class file `Program.class`. An error at this stage is indicated by
   `Error in assembling Java byte code for program X (HINT)`
3. If stage 2 was successful, the Java class file is run using Java with the test passed via standard input. The result is expected via standard output. Assuming the test is in the file `test_input`, and the result goes to the file `test_output`, this amounts to calling
   `java -Xmx100m Program < test_input > test_output`
   An error at this stage is indicated by
   `Error in running Java byte code for test Y of program X (HINT)`
   `Y` is the sequential number of the run for program `X`.
4. If stage 3 was successful, the result is compared with the expected output. A difference is indicated by
   `Wrong output for test Y of program X (HINT)`
5. If there is no difference, you pass this test.

The final line of feedback shows the total number of tests that your compiler has passed. Each test counts equally towards the marks for this question. By implementing `If_Else_AST.code` you can get up to 7 of 13 tests correct. The remaining 6 tests use Boolean expressions.

Your first submission to this question will attract no penalty. Thereafter each wrong submission attracts 10% penalty.

Answer: (penalty regime: 0, 10, ... %)

```
1  import re
2  import sys
3
4  class Scanner:
5      '''The interface comprises the methods lookahead and consume.
6         Other methods should not be called from outside of this class.'''
7
8      def __init__(self, input_file):
```

```
9           '''Reads the whole input_file to input_string, which remains constant.
10             current_char_index counts how many characters of input_string have
11             been consumed.
12             current_token holds the most recently found token and the
13             corresponding part of input_string.'''
14          # source code of the program to be compiled
15          self.input_string = input_file.read()
16          # index where the unprocessed part of input_string starts
17          self.current_char_index = 0
18          # a pair (most recently read token, matched substring of input_string)
19          self.current_token = self.get_token()
20
21      def skip_white_space(self):
22
```

	Got
☑	Correct output for test 1 of program 10 (if-then-else-end, division, assignments) Correct output for test 1 of program 11 (read, write, nested while-do-end, nested if-then-else-end, inequality, subtraction, assignments) Correct output for test 2 of program 11 (read, write, nested while-do-end, nested if-then-else-end, inequality, subtraction, assignments) Correct output for test 3 of program 11 (read, write, nested while-do-end, nested if-then-else-end, inequality, subtraction, assignments) Error in assembling Java byte code for program 12 (Boolean expressions, read, write, while-do-end, if-then-end, subtraction, assignment) Correct output for test 1 of program 14 (write, while-do-end, if-then-end, nested if-then-else-end, arithmetic, assignments) Correct output for test 1 of program 15 (read, write, nested while-do-end, if-then-end, if-then-else-end, arithmetic, assignments) Correct output for test 2 of program 15 (read, write, nested while-do-end, if-then-end, if-then-else-end, arithmetic, assignments) Error in assembling Java byte code for program 16 (Boolean expressions, read, write, if-then-end) 7 of 13 tests correct

Partially correct

Marks for this submission: 16.15/30.00. Accounting for previous tries, this gives **14.54/30.00**.

Question **2**

Correct

Mark 0.90 out of 1.00

Give the contents of the stack after the following instructions are run on the Java virtual machine. Assume the stack is initially empty.

Order the values from the bottom to the top of the stack and separate them by commas, e.g. `7,3,7,2,4,7`

```
sipush 9
sipush 8
sipush 3
sipush 6
sipush 3
imul
sipush 1
sipush 2
sipush 6
sipush 2
idiv
iadd
isub
iadd
imul
sipush 0
```

Answer: (penalty regime: 10, 20, ... %)

stack contents: 9,8,42,0

	Got
✔	Correct

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00. Accounting for previous tries, this gives **0.90/1.00**.

Question **3**

Correct

Mark 1.00 out of 1.00

Assume that variables x, y, and z are stored at relative locations 0, 1, and 2 respectively, and that each has the initial value 0.

Give their values after the following code is run on the Java virtual machine.

```
sipush 5
sipush 7
istore 2
istore 1
iload 1
iload 2
iload 1
istore 0
sipush 3
iload 2
iload 0
iload 0
iadd
istore 0
imul
istore 2
iadd
istore 1
```

Answer: (penalty regime: 10, 20, ... %)

x = 10
y = 12
z = 21

	Got
✔	Correct

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Question **4**

Correct

Mark 0.80 out of 1.00

Translate the assignment d := (b * b) - ((4 * a) * c) to Java virtual machine instructions as shown in the lectures. Assume variables a, b, c and d are stored at relative locations 0, 1, 2 and 3, respectively.

Give the instructions one per line with a space between each instruction and its operand, e.g.

```
sipush 7
iadd
sipush 2
sipush 7
isub
```

Answer: (penalty regime: 10, 20, ... %)

```
1  iload 1
2  iload 1
3  imul
4  sipush 4
5  iload 0
6  imul
7  iload 2
8  imul
9  isub
10 istore 3
11
```

	Got
✔	Correct

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00. Accounting for previous tries, this gives **0.80/1.00**.

Question **5**

Correct

Mark 1.00 out of 1.00

How many times is the `goto` instruction executed when the following instructions are run on the Java virtual machine?

```
sipush 63
istore 1
l1:
iload 1
sipush 1
if_icmple l2
iload 1
sipush 2
idiv
istore 1
goto l1
l2:
```

Answer: 5

Correct

Marks for this submission: 1.00/1.00.

◄ Quiz 8 = Assignment Part 2: Syntax Analysis

Jump to...

Quiz 10: Computability ►