

[Dashboard](#) / [My courses](#) / [COSC261-2022](#) / [Quizzes](#) / [Quiz 7 = Assignment Part 1: Lexical Analysis](#)

Started on Saturday, 9 April 2022, 2:36 AM

State Finished

Completed on Friday, 13 May 2022, 5:05 PM

Time taken 34 days 14 hours

Marks 31.18/33.00

Grade **94.47** 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

Correct

Mark 20.00 out of 20.00

Please download the [scanner template](#).

1. Read and understand the scanner template. In particular, you need to understand how it represents the consumed part of the input and how it finds the next token. Look up Python's `re` library for regular expression matching. In particular, check how special characters are used and escaped.
2. Implement the methods `Scanner.skip_white_space` and `Scanner.consume` according to their specifications. Working implementations of these methods are necessary for the remaining parts of the compiler. The only attribute you need to access in `Scanner.consume` is `current_token`; use `get_token` for the character-level work. Note that white-space includes spaces, tabs and newline characters.
3. Extend the method `Scanner.get_token` to check if the input contains extra non-white-space characters that do not match any token (see the docstring of this method for more detail). In case of a lexical error, the program will output all recognised tokens before the error except the last one (which is the current token when the error occurs); do *not* compensate for this (for example, by adding additional print-statements).
4. Extend the class `Token` by the reserved words `read` and `write`, by the symbols `!=`, `*` and `/` for inequality, multiplication and division, and by number constants. You will have to add a regular expression for each of these tokens and new token constants `READ` and `WRITE`. Use the existing token constants for inequality, multiplication, division and numbers.

The scanner methods are repeatedly called to show all tokens of the input, which is a program. For example, assume the scanner is in the file `scanner.py` and the file `program` contains

```
z := 2;
if z < 3 then
  z := 1
end
```

Running the scanner by `python3 scanner.py < program > tokens` will produce the file `tokens` which contains

```
ID z
BEC
NUM 2
SEM
IF
ID z
LESS
NUM 3
THEN
ID z
BEC
NUM 1
END
```

The values of identifiers and numbers are displayed after the corresponding tokens.

On submission your code is tested with several programs. Tests are carried out using the following procedure, which you should reproduce using your own programs.

1. Your scanner is run using Python 3 with the program passed via standard input. The token sequence is expected via standard output. Assuming your scanner is in the file `scanner.py`, the program is in the file `program`, and the token sequence goes to the file `tokens`, this amounts to calling `python3 scanner.py < program > tokens`. An error at this stage is indicated by `Error in compiling 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 token sequence is compared with the expected token sequence. A difference is indicated by `Wrong output for X (HINT)`.
3. If there is no difference, you pass this test.

The final line of feedback shows the total number of tests that your scanner has passed. Each test counts equally towards the marks for this question.


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         '''Consumes all characters in input_string up to the next

```

	Got
	<p>Correct output for program 01 (assignment, statements)</p> <p>Correct output for program 02 (if-then-end, assignments)</p> <p>Correct output for program 03 (while-do-end, addition, subtraction, assignments)</p> <p>Correct output for program 04 (while-do-end, addition, subtraction, multiplication, division, inequality, assignments)</p> <p>Correct output for program 05 (write, addition, subtraction, assignments)</p> <p>Correct output for program 06 (write, while-do-end, if-then-end, addition, assignments)</p> <p>Correct output for program 07 (read, while-do-end, if-then-end, addition, division, assignments)</p> <p>Correct output for program 08 (read, write, while-do-end, addition, subtraction, assignments)</p> <p>Correct output for program 09 (read, write, nested while-do-end, multiplication, subtraction, assignments)</p> <p>Correct output for program 10 (if-then-else-end, division, assignments)</p> <p>Correct output for program 11 (read, write, nested while-do-end, nested if-then-else-end, inequality, subtraction, assignments)</p> <p>Correct output for program 13 (if-then-end, assignments)</p> <p>Correct output for program 14 (write, while-do-end, if-then-end, nested if-then-else-end, arithmetic, assignments)</p> <p>Correct output for program 15 (read, write, nested while-do-end, if-then-end, if-then-else-end, arithmetic, assignments)</p> <p>Correct output for program 17 (while-do-end, subtraction, assignments, lexical error)</p> <p>15 of 15 tests correct</p>

Passed all tests! 

Correct

Marks for this submission: 20.00/20.00.

Question **2**

Correct

Mark 10.00 out of 10.00

In this question you will create a scanner for the same language using the [PLY](#) scanner generator. The PLY package is installed on the lab machines. See the instructions in the following template if you want to install PLY on your own machine.

Please download the [PLY scanner template](#).

1. Read and understand the PLY scanner template. Using a generator, there is no need for character-level access to create tokens. It is only necessary to specify tokens and to provide corresponding regular expressions.
2. Extend the scanner by the reserved words `read` and `write`, by the symbols `!=`, `*` and `/` for inequality, multiplication and division, and by number constants.
3. The regular expression for identifiers also captures reserved words. Modify the corresponding action so that the returned token type is the reserved word, not the general identifier, in these cases.

The PLY scanner can be run in the same way as the previous scanner and should generate the same output (except in the case of a lexical error, where the error message is slightly different; do not worry about this). On submission, it will be tested in the same way. You should again test with your own programs.

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

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

```

1  """
2  This program uses PLY (Python Lex-Yacc). Documentation for PLY is
3  available at
4      https://www.dabeaz.com/ply/ply.html
5
6  PLY can be installed on your own system using pip, which comes
7  preinstalled on recent versions of Python (>= 3.4). Using pip the PLY
8  package can be installed with the following command:
9      pip3 install ply
10 This requires Internet access to download the package.
11 """
12
13 import ply.lex as lex
14 import sys
15
16 """
17 PLY's scanner works by matching regular expressions to the tokens.
18 If you need a reminder of the syntax for regular expressions, check
19 the following link:
20     https://docs.python.org/3/library/re.html
21
22 All tokens that the lexer can find must be declared in a list of

```

	Got
<div>✓</div>	<p>Correct output for program 01 (assignment, statements)</p> <p>Correct output for program 02 (if-then-end, assignments)</p> <p>Correct output for program 03 (while-do-end, addition, subtraction, assignments)</p> <p>Correct output for program 04 (while-do-end, addition, subtraction, multiplication, division, inequality, assignments)</p> <p>Correct output for program 05 (write, addition, subtraction, assignments)</p> <p>Correct output for program 06 (write, while-do-end, if-then-end, addition, assignments)</p> <p>Correct output for program 07 (read, while-do-end, if-then-end, addition, division, assignments)</p> <p>Correct output for program 08 (read, write, while-do-end, addition, subtraction, assignments)</p> <p>Correct output for program 09 (read, write, nested while-do-end, multiplication, subtraction, assignments)</p> <p>Correct output for program 10 (if-then-else-end, division, assignments)</p> <p>Correct output for program 11 (read, write, nested while-do-end, nested if-then-else-end, inequality, subtraction, assignments)</p> <p>Correct output for program 13 (if-then-end, assignments)</p> <p>Correct output for program 14 (write, while-do-end, if-then-end, nested if-then-else-end, arithmetic, assignments)</p> <p>Correct output for program 15 (read, write, nested while-do-end, if-then-end, if-then-else-end, arithmetic, assignments)</p> <p>Correct output for program 17 (while-do-end, subtraction, assignments, lexical error)</p> <p>15 of 15 tests correct</p>

Passed all tests! ✓

Correct

Marks for this submission: 10.00/10.00.

Question **3**

Correct

Mark 0.80 out of 1.00

Order the following phases of a compiler.

1. scanning
2. parsing
3. semantic analysis
4. machine-independent optimisation
5. code generation
6. machine-dependent optimisation

Your answer is correct.

CorrectMarks for this submission: 1.00/1.00. Accounting for previous tries, this gives **0.80/1.00**.Question **4**

Partially correct

Mark 0.38 out of 1.00

Identify the kind of each of the following Python 3 tokens.

yield	identifier
*	operator symbol
throw	reserved word
//	operator symbol
elif	identifier
0o0	none of the others
""elif""	string constant
if_then_else	none of the others

Your answer is partially correct.

You have correctly selected 3.

Partially correct

Marks for this submission: 0.38/1.00.

Question **5**

Not answered

Mark 0.00 out of 1.00

Consider the following sequence of definitions of regular expressions:

$\text{digit} = 0-9$

$\text{hexdigit} = \text{digit}[a-f][A-F]$

$\text{hexinteger} = 0(x|x)\text{hexdigit}^+$

Give the shortest string that matches hexinteger. If there are several matching strings with the same minimal length, give the lexicographically smallest according to the ASCII order. (Here a string is a sequence of symbols over the ASCII alphabet, not a Python string - that is, do not put your answer in quotes.)

Answer:

[◀ Assignment setup](#)

Jump to...

[Quiz 8 = Assignment Part 2: Syntax Analysis ▶](#)