



A10.8 Stages of compilation, linkers and loaders

Operating systems, software and translators · A level · OCR H446 1.2.2,
AQA 7517 4.6.1.3, Eduqas A500QS 1.8 · about 45 min

Name _____

Class _____

Date _____

What this lesson is about

Lexical analysis, syntax analysis, code generation and optimisation, then static and dynamic linking and the loader.

- 1. Lexical analysis
- 2. Syntax analysis
- 3. Code generation
- 4. Optimisation
- Libraries, linkers and loaders

Questions

6 marks in all

1. Put the stages of compilation in order.

[1 mark]

Number the lines 1 to 4 to put them in the right order.

- ___ Optimisation
- ___ Lexical analysis
- ___ Syntax analysis
- ___ Code generation

2. Which of these happen during lexical analysis?

[1 mark]

Tick every answer that is true.

- ☐ A Comments and whitespace are removed
- ☐ B The source is split into tokens
- ☐ C Identifiers are added to the symbol table
- ☐ D The abstract syntax tree is built

3. At which stage is a missing closing bracket detected?

[1 mark]

- ☐ A Syntax analysis
- ☐ B Lexical analysis
- ☐ C Code generation
- ☐ D Linking

4. What is an advantage of dynamic linking over static linking?

[1 mark]

- ☐ A Executables are smaller, and many programs can share one copy of a library in memory
- ☐ B The program will run even if the library is missing
- ☐ C The program always uses the exact library version it was built with
- ☐ D No loader is needed

5. Which part of the operating system copies an executable from secondary storage into memory and adjusts its addresses so it can run? [1 mark]

6. What does this program print? [1 mark]

```
source = "total = total + 5 # add five"
code = source.split("#")[0]
tokens = code.split()
print(len(tokens), tokens)
```

The task: a lexer

Write `tokenise(text)`, the lexical analysis stage for a tiny language. `text` is a string that may contain several lines. It returns a list of `(type, text)` tuples in order, where `type` is one of: - "KEYWORD": a word in `KEYWORDS`; - "IDENTIFIER": any other word, made of letters, digits and underscores and not starting with a digit; - "NUMBER": one or more digits; - "OPERATOR": one character from `OPERATORS`; - "PUNCTUATION": one character from `PUNCTUATION`. Whitespace is skipped, and a `#` starts a comment that runs to the end of its line and produces no tokens. Do not use Python's own `tokenize` module or regular expressions: read the characters yourself. Then print each token of `source` on its own line as `<type> <text>`, for example `KEYWORD while`. Finally print the symbol table: `symbols:` followed by each different identifier, in the order it first appears, separated by `,`. The robot stays still.

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

KEYWORDS = ["while", "if", "else", "def", "return"]
OPERATORS = "+-*/<>"
PUNCTUATION = "():,;"

source = "while gap > 20: # keep going\n    gap = gap - step"

def tokenise(text):
    tokens = []
    return tokens
```

Plan your program here, then type it in and press Run.



Do it on the robot

www.bugbotlab.com/learn/a10-8-stages-of-compilation/

The simulator checks it and tells you when it passes. Nothing to install, no account.

Challenges

1. Make the lexer report a lexical error, with the position, for a character it does not recognise, such as `$`.
2. Add two-character operators `==`, `<=` and `>=`. How does the lexer know whether `=` is finished?
3. Add division to `generate` and `fold`, as a `DIV` instruction, and check that in `speed + 60 / 4` the division is folded to `PUSH 15.0`.