1.4

4. What arguments can you make against the idea of a single language for all programming domains?

不同的領域有其獨特需求，沒有任何一種語言能同時有效滿足。例如，系統程式設計需要低階控制與高效能，這是 C 或 Rust 的強項；而資料科學則依賴快速原型與豐富的函式庫，這正是 Python 擅長的領域。

通用語言還會面臨效能與生產力之間的取捨問題。若同時要兼顧專業開發者與初學者，語言勢必變得複雜又低效。此外，不同領域使用不同的程式設計範式——物件導向、函數式、宣告式——它們各自解決不同型態的問題，若強行合併只會導致語法混亂、結構臃腫。

再者，各語言都有專屬的生態系與工具鏈，如編譯器、函式庫與框架，要完全取代幾乎不可能。

1.18

18. Many contemporary languages allow two kinds of comments: one in which delimiters are used on both ends (multiple-line comments), and one in which a delimiter marks only the beginning of the comment (one-line comments). Discuss the advantages and disadvantages of each of these with respect to our criteria.

Block Comments（多行註解，雙邊符號如 /* ... */）
適合長篇說明或文件註解，易於分隔文字與程式碼。若註解過長，結尾符號易被忽略，造成程式碼被隱藏。
一次即可註解整段程式，快速方便。 若區塊內已有 /* ... */，無法再巢狀使用，易出錯。
不易出現「半行註解」。若結尾遺失，整個檔案可能被註解掉。不支援巢狀註解（如 C/C++），註解程式碼時易誤傷。
編譯器需維護「跨行狀態」，處理較複雜。適合文件生成工具（如 Javadoc）。

Line Comments（單行註解，前置符號如 // 或 #）
適合短註解或行內說明，直觀明確。多行註解時每行都需前綴符號，降低可讀性。
寫入快速，不需結尾符號。 若要註解多行，需逐行加符號，較麻煩。
每行獨立，無巢狀問題。錯誤範圍小，不會誤註太多程式。行尾註解過多可能造成閱讀困難。
編譯器只需忽略當行之後的內容，實作簡單。與版本控制系統整合良好（逐行比對）。需要依靠編輯器才能批量加註解。

2.14

14. What are the arguments both for and against the idea of a typeless language?

支持無型語言（Typeless Language）的理由

1. 語法簡單、靈活性高：不用宣告型別，撰寫快速且自由。
2. 開發速度快：適合原型設計、快速測試與互動式開發（如 Python、JavaScript）。
3. 可重用性高：同一函式可處理多種資料型態，程式更通用。
4. 適合特定應用領域：AI、資料分析、腳本開發中，速度與實驗性比型別安全更重要。

反對無型語言的理由

1. 錯誤偵測困難：型別錯誤要等到執行時才發現，可能導致程式崩潰。
2. 維護困難：大型專案中無法保證資料一致性，修改程式風險大。
3. 效能較低：執行時必須不斷檢查型別，速度比靜態語言慢。
4. 工具支援不足：IDE 難以提供自動補全與靜態分析，除錯較不方便。

3.4

EXAMPLE 3.4 An Unambiguous Grammar for Expressions

```
<assign> → <id> = <expr>
<id> → A | B | C
<expr> → <expr> + <term>
       | <term>
<term> → <term> * <factor>
       | <factor>
<factor> → ( <expr> )
         | <id>
```

Rewrite the BNF of Example 3.4 to add the ++ and -- unary operators of Java

<assign> → <id> = <expr>

<id> → A | B | C

<expr> → <expr> + <term> | <term>

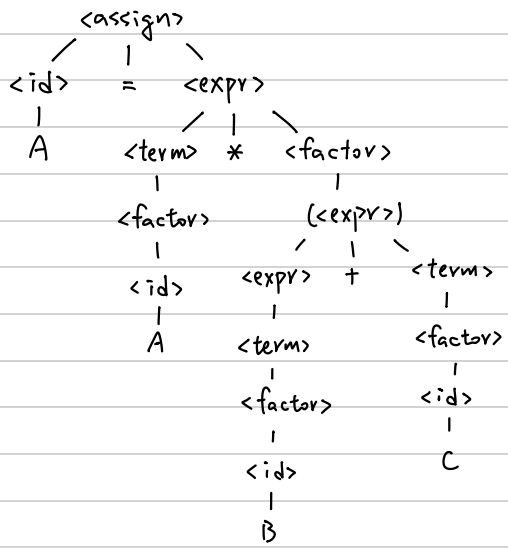
<term> → <term> * <factor> | <factor>

<factor> → (<expr>) | <id> | <id> ++ | <id> - -

3.7

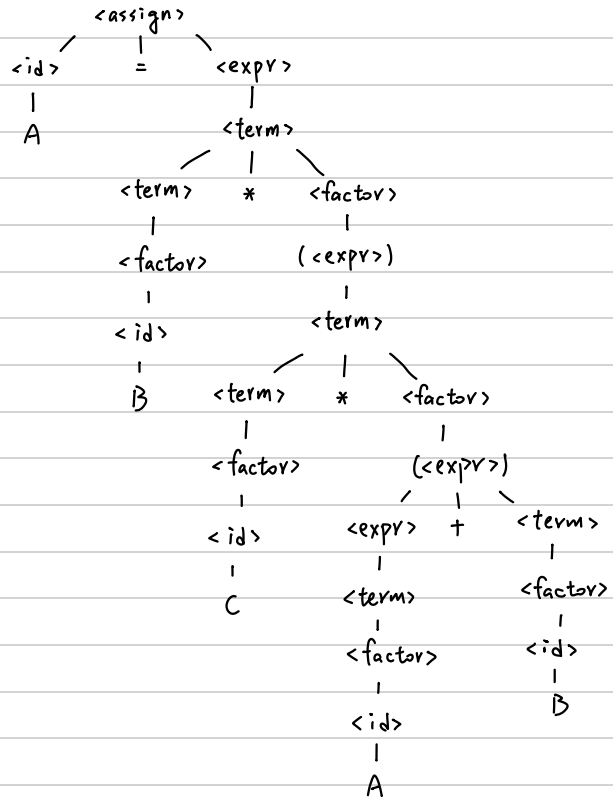
(c) A = A * (B+C)

<assign> → <id> = <expr>
→ A = <expr>
→ A = <term>
→ A = <term> * <factor>
→ A = <factor> * <factor>
→ A = <id> * <factor>
→ A = A * <factor>
→ A = A * (<expr>)
→ A = A * (<expr> + <term>)
→ A = A * (<term> + <term>)
→ A = A * (<factor> + <term>)
→ A = A * (<id> + <term>)
→ A = A * (B + <term>)
→ A = A * (B + <factor>)
→ A = A * (B + <id>)
→ A = A * (B + C) *



(d) A = B * (C * (A+B))

<assign> → <id> = <expr>
→ A = <expr>
→ A = <term>
→ A = <term> * <factor>
→ A = <factor> * <factor>
→ A = <id> * <factor>
→ A = B * <factor>
→ A = B * (<expr>)
→ A = B * (<term>)
→ A = B * (<term> * <factor>)
→ A = B * (<factor> * <factor>)
→ A = B * (<id> * <factor>)
→ A = B * (C * <factor>)
→ A = B * (C * (<expr>))
→ A = B * (C * (<expr> + <term>))
→ A = B * (C * (<term> + <term>))
→ A = B * (C * (<factor> + <term>))
→ A = B * (C * (<id> + <term>))
→ A = B * (C * (A + <term>))
→ A = B * (C * (A + <factor>))
→ A = B * (C * (A + <id>))
→ A = B * (C * (A + B)) *



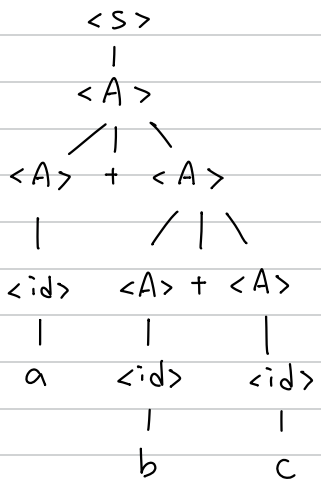
3.8

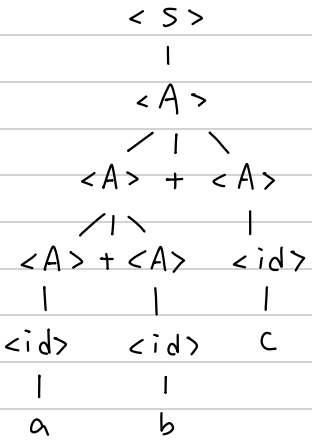
8. Prove that the following grammar is ambiguous:

<S> → <A>

<A> → <A> + <A> | <id>

<id> → a | b | c





∵ 有 2 種 parse tree

∴ ambiguous

3.11

11. Consider the following grammar:

<S> → <A> a <B> b

<A> → <A> b | b

<B> → b

Which of the following sentences are in the language generated by this grammar?

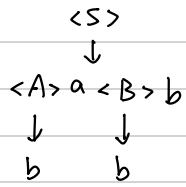
a. babb

b. bbbabb

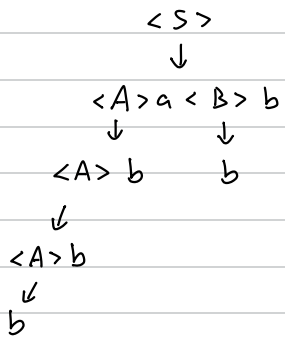
c. bbaaaaabc

d. aaaaaa

a. babb



b. bbbabb



c. bbaaaaabc

有 c，不可能

d. aaaaaa

沒有 b，不可能

3.13

13. Write a grammar for the language consisting of strings that have *n* copies of the letter a followed by one more number of copies of the letter b, where $n > 0$. For example, the strings abb, aaaabbbbb, and aaaaaaaabbbbbbbbb are in the language but a, ab, ba, and aaabb are not.

<S> → a <A> b

<A> → b | <S>

3.23

(b) $b = (c+10)/3$ $\{b > 6\}$

$\Rightarrow (c+10)/3 > 6$

$\Rightarrow c+10 > 18$

$\Rightarrow c > 8$ ✗

(c) $a = a + 2*b - 1$ $\{a > 1\}$

$\Rightarrow a + 2*b - 1 > 1$

$\Rightarrow a + 2*b > 2$

$\Rightarrow 2*b > 2 - a$

$\Rightarrow b > \frac{2-a}{2}$ ✗

5.6

(a)

(i) sub1

(ii) sub1

(iii) main

(b)

(i) sub1

(ii) sub1

(iii) sub1

5.8

sub1 : a(sub1), y(sub1), z(sub1), x(main)

sub2 : a(sub2), b(sub2), z(sub2), y(sub1), x(main)

sub3 : a(sub3), x(sub3), w(sub3), y(main), z(main)