113-2 國立成功大學編譯系統課程

NCKU Compiler Construction - 2025 Spring

Homework 2. Syntactic Analysis (Parser)

Deadline: 2025/06/22 23:59:59

Late submissions are not accepted

本學期的作業配分

- 這學期總共有 3 份作業，作業成績佔學期成績 40%
  - Homework 1: 10%
  - Homework 2: 15%
  - Homework 3: 15%

編譯器的步驟

- 詞法分析 Lexical analysis (Scanning)
- 語法分析 Syntax analysis (Parsing)
- 語意分析 Semantic analyzer
- 中間碼生成
- 程式碼優化

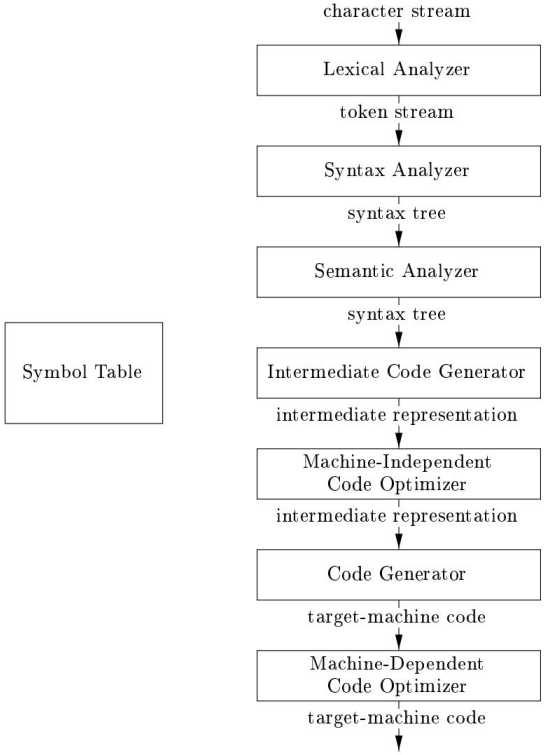
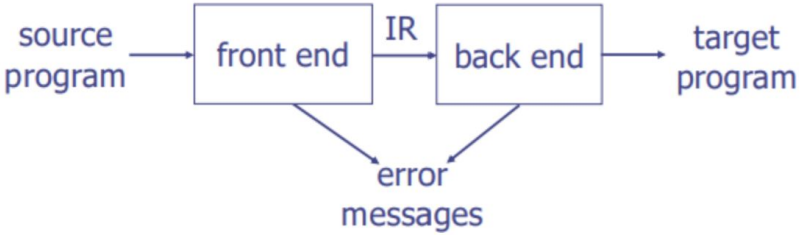


Figure 1.6: Phases of a compiler

Analysis-Synthesis Model



- 綜合來說，編譯的流程可以分成：
  - 分析 Analysis (front end)
    - 將程式拆成很多零件
    - 得到中間碼 intermediate representation (IR)
  - 合成 Synthesis (back end)
    - 利用中間碼組合出目標程式
    - 最佳化 (可做可不做)

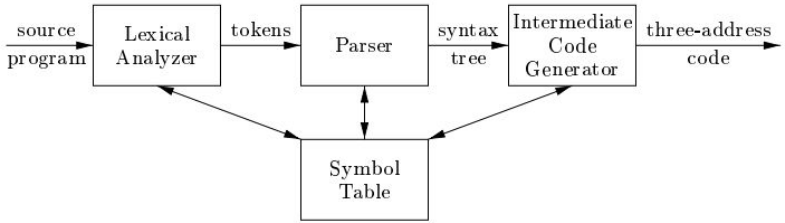
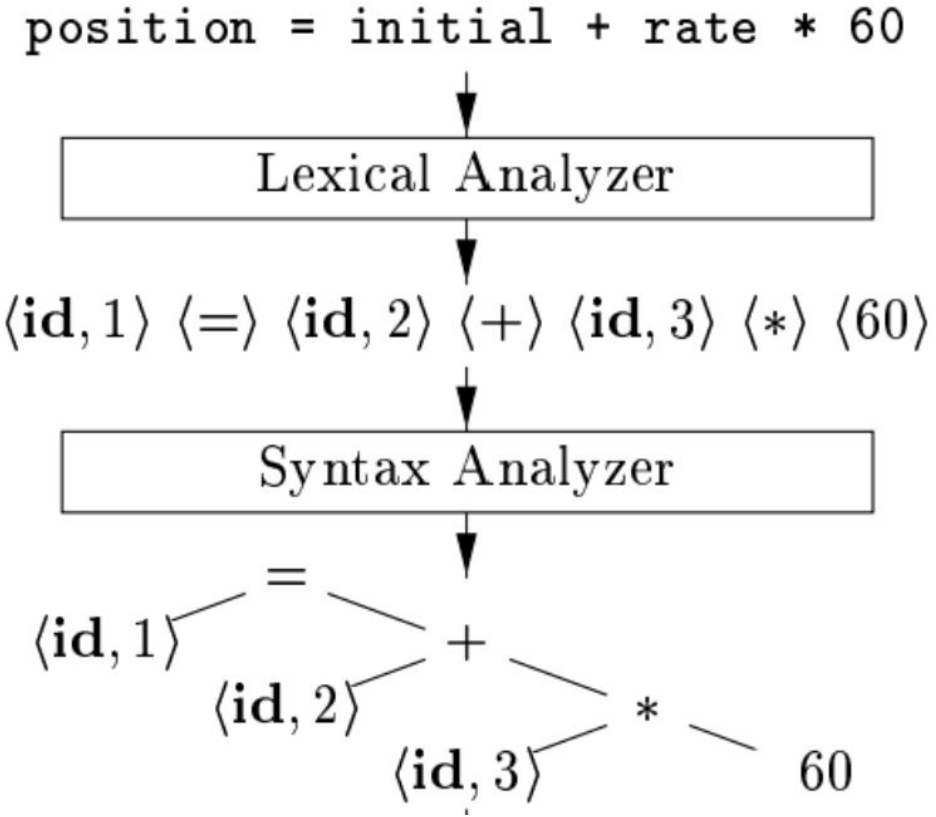


Figure 2.3: A model of a compiler front end

Analysis

- Linear Analysis (Lexical Analysis)
  - 掃描程式碼，將文字拆解成許多片段 (Token)
- Hierarchical Analysis (Syntax Analysis) (Homework 2)
  - 將這些 Token 組成文法
- Semantic Analysis
  - 辨識語法錯誤，與型別問題

Analysis


position = initial + rate * 60
Lexical Analyzer
⟨id, 1⟩ ⟨=⟩ ⟨id, 2⟩ ⟨+⟩ ⟨id, 3⟩ ⟨∗⟩ ⟨60⟩
Syntax Analyzer
=
⟨id, 1⟩
+
⟨id, 2⟩
*
⟨id, 3⟩
60

Syntactic Analysis (Parser)

- 這是實作 Compiler 的第二個步驟!
- 語法分析，就是 check 語法對不對的一個步驟
- 就像是英文有自己的文法，而 Parser 要做的事情就是 check 程式碼的語法是否符合規定

Syntactic Analysis (Parser)

- 作業一已經將程式碼切成很多 Token
- 接下來我們要把 Token 照順序放入 Parser 解析語法
    - 你可以想像成我們要把這些 Token

      組合成一個可以被 "理解" 的句子
    - 而要創造出一個句子，我們必須要先有語法
    - 所以現在我們要制訂出一套文法標準

Syntax Definition 語法定義

- 每一個語言都有一個既定的 "規則"
- 這一個規則描述了程式語言在編寫時必須遵守的一些規範
- 我們通常把一個 "規則" 稱之為 文法 (Grammar)

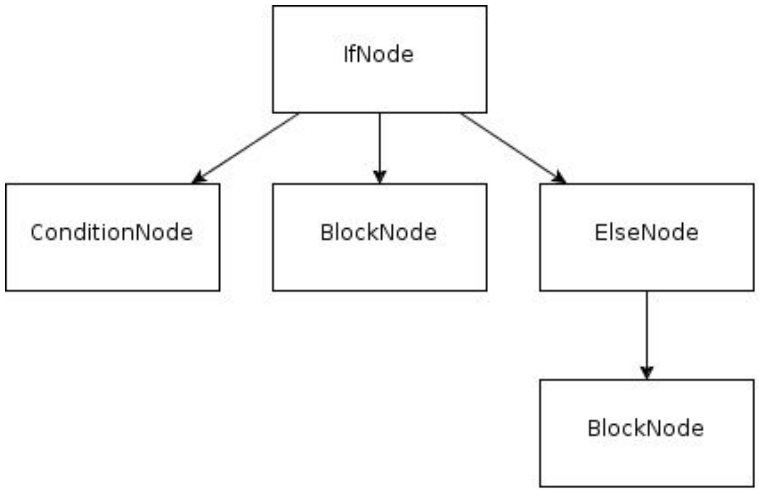
if (expression) statement **else** statement

Syntactic Analysis (Parser)

- 我們的目標是：
  - 將 Token 重新以正確的順序 (優先級) 輸出
  - 簡易的判斷所有變數的可視範圍 (Scope)
  - 針對所有的 Scope Level 輸出一個表格，表格內有該等級範圍內的所有變數資訊

Grammar Design 語法設計

- 這個步驟我們最重要的是要建立出一個 Parser Tree
- 這一個樹要能表示解析所有的語法規則



Grammar Design

- 建立 Parser Tree 的種類又分成兩種：
  - Top-Down Parser (從根開始往下)
  - Bottom-Up Parser (從葉子開始往上長)
  - 但在這一個作業我們會直接使用別人寫好的工具，這一個工具會幫助我們更快的建立出 Parser Tree

Grammar Design (Yacc)

- Yet Another Compiler Compiler
- 使用 LALR(1) 的方式做語法分析
- 我們可以用 BNF (巴科斯範式) 的表示法來直接表明我們想要設計的文法
    - 這邊特別科普一下：其實 BNF 就只是 CFG 所衍生出來的一種特殊寫法而已，所以本質上還是以 CFG 為基礎

Grammar Design (Yacc)

- 設計文法的一開始我們要跟 Lex 做搭配，先定義好所有 token 的屬性
- 這樣你設計文法的時候 Yacc 才會認得這個 token 是誰

```
8   /* Token without return */
9   %token LET MUT NEWLINE
10  %token INT FLOAT BOOL STR
11  %token TRUE FALSE
12  %token GEQ LEQ EQL NEQ LOR LAND
13  %token ADD_ASSIGN SUB_ASSIGN MUL_ASSIGN DIV_ASSIGN REM_ASSIGN
14  %token IF ELSE FOR WHILE LOOP
```

Grammar Design (Yacc)

- 針對有 value 的 token 我們要背著他的屬性
- 所有 token 的型態在一開始 %union 要先定義好

```
%union {
    int i_val;
    float f_val;
    char *s_val;
}
```

```
/* Token with return, which need to sepcify type */
%token <i_val> INT_LIT
%token <f_val> FLOAT_LIT
%token <s_val> STRING_LIT
```

Grammar Design (Yacc)

- 如何設計文法？(BNF 格式)
- 下列文法可解析：
    - int apple = 10;
    - int Sumikko_Gurashi;

```
Sample // 宣告變數
    : INT ID '=' INT_LIT ';'
    | INT ID ';'
```

Grammar Design (Yacc)

- 我們解析文法的時候要搭配語意動作
- 輸出我們當前解析到的資訊，確保我們解析過程正確
- 只要在符號的右方使用大括號即可
- 當解析到 int 時就會執行語意動作

```
1  Sample // 宣告變數
2      : INT { printf("%d\n","INT") } ID '=' INT_LIT ';'
3      | INT ID ';'
```

語法制導翻譯方案 Syntax-Directed Translation Scheme

- 語意動作可以讓我們決定該動作執行的時間 (順序)
- 遍歷語法樹時，到達語意動作產生的節點才會執行該動作

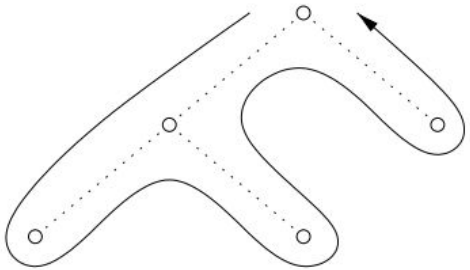


Figure 2.12: Example of a depth-first traversal of a tree

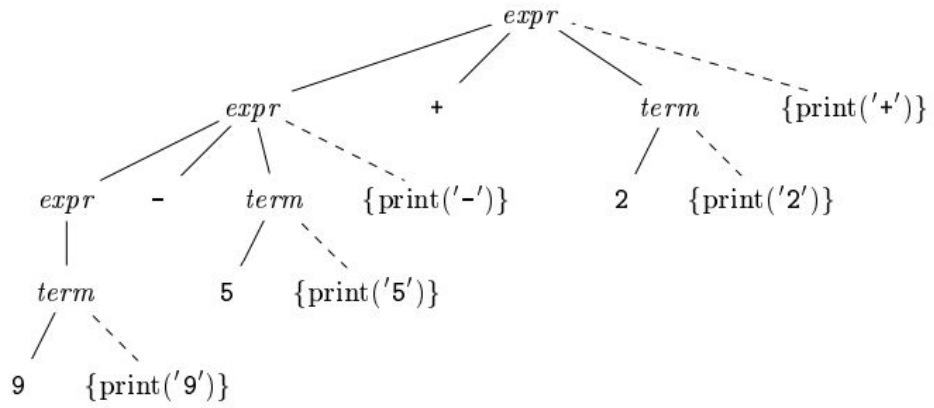


Figure 2.14: Actions translating 9-5+2 into 95-2+

Grammar Design (Yacc)

- 其餘更詳細的語法請參閱課程簡報：D
- 接下來我們講解我們希望你的 Yacc 可以輸出什麼

Grammar Design (Yacc) - Output

- 將 Token 分析意思後，以正確的順序輸出

```
fn main() {
    let mut x: i32 = 0;
    println( x);
    x = 10;
    println( x);
    x += 2;
    println( x);
    x -= 3;
    println( x);
    x *= 4;
    println( x);
    x /= 5;
    println( x);
    x %= 6;
    println( x);

    let mut yy: f32 = 3.14;
    println( yy);
    yy = 10.4;
    println( yy);
    yy += 2.0;
    println( yy);
    yy -= 3.0;
    println( yy);
    yy *= 4.0;
    println( yy);
```

```
> Create symbol table (scope level 0)
func: main
> Insert `main` (addr: -1) to scope level 0
> Create symbol table (scope level 1)
INT_LIT 0
> Insert `x` (addr: 0) to scope level 1
IDENT (name=x, address=0)
PRINTLN i32
INT_LIT 10
ASSIGN
IDENT (name=x, address=0)
PRINTLN i32
INT_LIT 2
ADD_ASSIGN
IDENT (name=x, address=0)
PRINTLN i32
INT_LIT 3
SUB_ASSIGN
IDENT (name=x, address=0)
PRINTLN i32
INT_LIT 4
MUL_ASSIGN
IDENT (name=x, address=0)
PRINTLN i32
INT_LIT 5
```

Grammar Design (Yacc) - Output

- 各個符號的優先級可以參考測資答案

Grammar Design (Yacc) - Output

- Insert:
- 遇到變數時，輸出他被 Insert 到的 scope 與 address

```
1  fn main() { // Your first rust program
2      println("Hello World!");
3      /* Hello
4      World */ /*
5      */
6  }
```

```
1  > Create symbol table (scope level 0)
2  func: main
3  > Insert `main` (addr: -1) to scope level 0
4  > Create symbol table (scope level 1)
5  STRING_LIT "Hello World!"
6  PRINTLN str
7
8  > Dump symbol table (scope level: 1)
9  Index     Name      Mut       Type      Addr      Lineno    Func_sig
10
11 > Dump symbol table (scope level: 0)
12 Index     Name      Mut       Type      Addr      Lineno    Func_sig
13 0         main      -1        func      -1        1         (V)V
14 Total lines: 6
15
```

Grammar Design (Yacc) - Output

- 建議寫的時候參考解答輸出，更加了解每一個測試資料要求輸出的東西！
- 每一個測試資料確認完輸出順序後再動手會比較好 ><

Homework 2 作業上傳

- 請將 compiler.y 與 makefile 壓縮並上傳 moodle
- zip file name: your_student_id.zip

Homework 2 作業上傳

- 想先在自己系統寫的人請使用以下指令安裝環境
- sudo apt install flex bison

Homework 2: Grading

- 本次作業滿分為 120 分
- 評分標準與上一個作業相同

作業提示: 注意運算符號的結合性

- 有些符號屬於左結合，有些屬於右結合

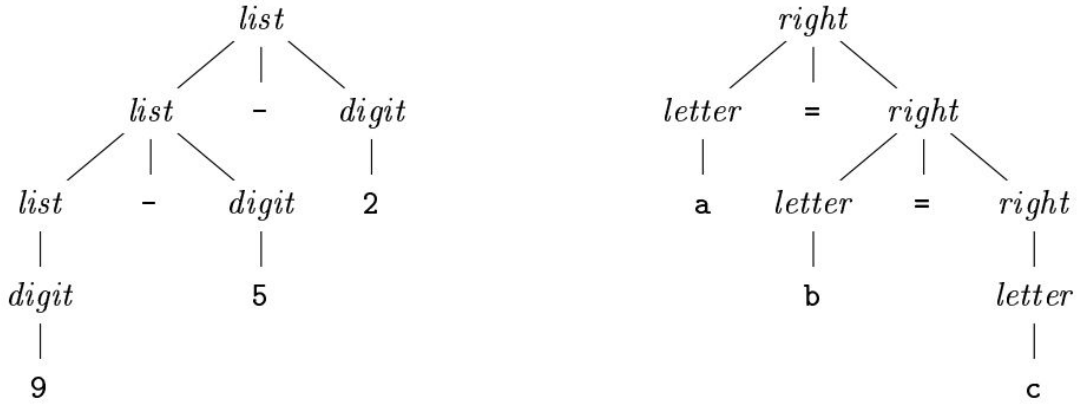


Figure 2.7: Parse trees for left- and right-associative grammars

作業提示：有時候光依靠結合性無法解決問題

- 當一個文法具有許多運算符號時依舊無法解決模稜兩可的問題
  - 結合性只能解決同一個運算符號重複出現造成的模稜兩可問題
- 因此我們需要再處理不同運算符號之間的優先順序來解決問題
  - 我們可以透過改寫文法來解決這樣的問題
  - 這是最直接也最方便的方法
- 總結：設計一個好的文法是很重要的！

Homework 2: Challenge Subtask

- 讓你的 Parser 不會發生 Shift-Reduce Conflict
  - 設計文法必須小心，避免發生模稜兩可