Book
Author → Aho, Ulman

* Just In Time Compiler
(JIT compiler)

Date Jan 12th, 2023

Compiler Design

Introduction

* Compiler is a program that translate a source program (a program written in a source language) & translate it to the equivalent program (target program, program written in a target language)

Source Program → [Compiler] → Target Program

Source Program:
- .c/.cpp
- .java

Target Program:
- .obj
- .asm

---

(2) obj1, obj2, ..., objn + DLL → [Linker] → .exe

Source Program + Input → [Interpreter] → Output

(3) .exe → [Loader] → [Memory]

*) The machine language program

(1) .asm → [Assembler] → .o

Date .....

Assembler → The compiler may produce an assembly language program as its output. Because it is easier to produced as output & its easier to debug. The assembly lang. program is then processed by the program called assembler that produces the relocatable object code as its output.

Linker - Large Programs are often compiled in pieces. So the relocatable code may have to be linked together with other relocatable object files, library files into the code that actually runs on the machine. The linker is also responsible for this task. It gives the exe.

Loader → The loader finally puts the executable file into memory for execution.

Structure of the Compiler

It is divided into two parts

i) Analysis (front-end)

ii) Synthesis (Back-end)

The Analysis breaks up the source program into constituent pieces & imposes the grammatical structure on them

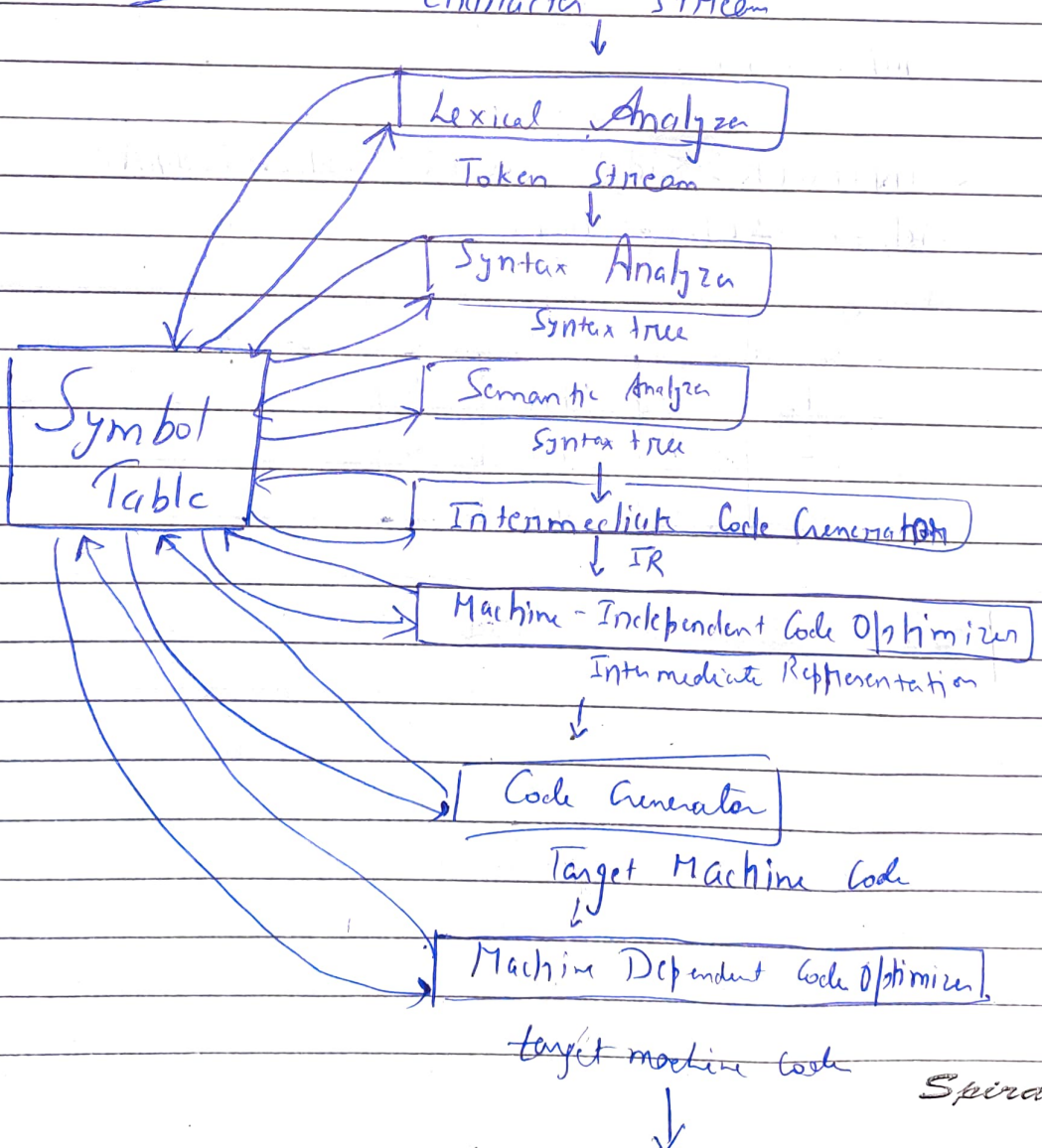
Spiral

Date .....

It then uses the structure to create an intermediate representation (IR) of source program. The Analysis also collects the information about the source program & stores it in a data structure called a symbol table, which is passed along with IR to the Synthesis part.

Synthesis part constructs the desired target program from the IR & information in the symbol table.

Phases of a Compiler (fig 1.6)



Spiral

Date .....

Lexical Analysis

* It is done by the program called lexical analyser. (In process)
Lexical Analysis or scanning.

The first phase of a compiler is called lexical analyser. It reads the stream of characters and groups the characters into meaningful sequences called lexemes.

For each lexeme, the lexical analyser produces as output a token of the form

<token_name, attribute value>

Ex → int a,b,c;

<INTEGER><id,1><COMMA><id,2><COMMA>
<id,3><SEMI-COLON>

Symbol Table →

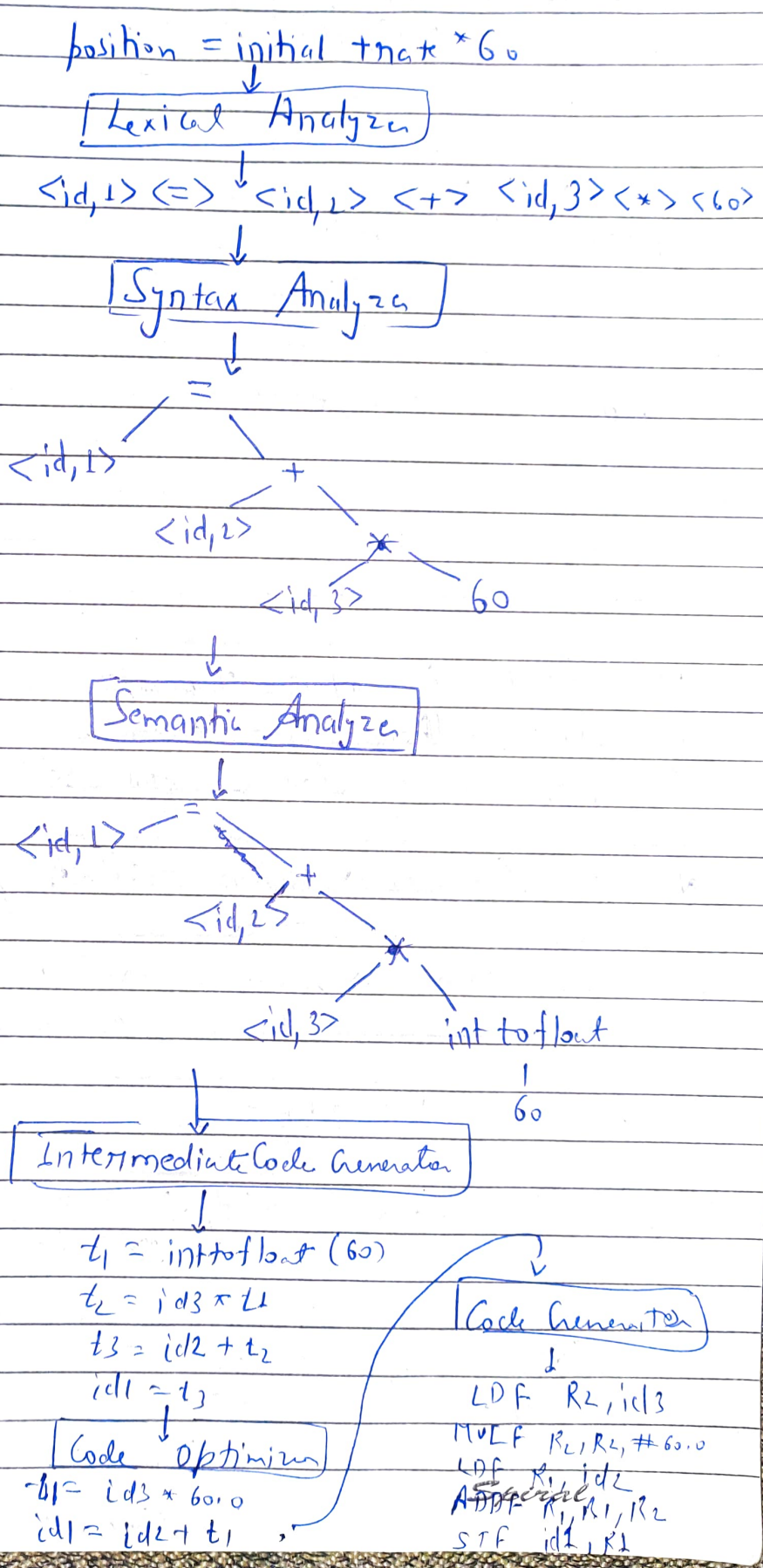
1	a	int	4
2	b	int	4
3	c	int	4

position = initial + rate * 60

Translation of an Assignment Statement

Date .....

fig 1.7



(ch-2 is skipped)

Date Jan. 16th, 2023

Chapter-3 Lexical Analysis

Syntax

whitespaces
pattern {action}
pattern {action}
⋮
pattern {action}

Rule Set Syntax

%%
definition
%%
Rules
%%
auxiliary func°

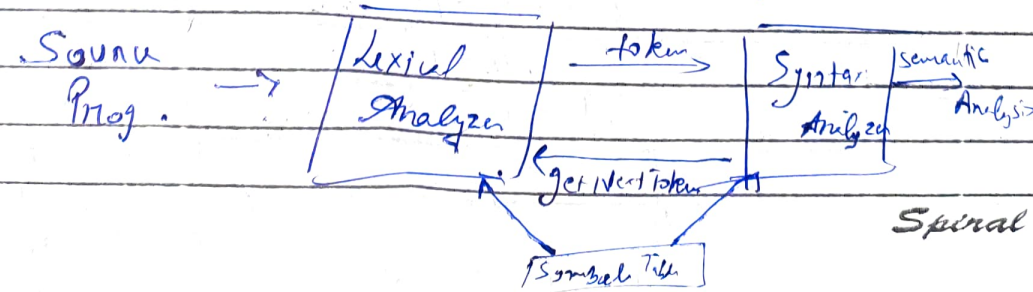
1) Read the input characters
2) Group them into lexemes.
3) Sequence of tokens for each lexeme.

Note → Lexemes → words derived from input character stream

Token → Lexemes mapped into token name & an attribute value

$X = a + b * 2$

Tokens = { <id, 1>, <=>, <id, 2>, <+>, <id, 3>, <*>, <num> }



Spiral

Date .....

Lexeme → sequence of character recognised by a pattern.

pattern → structure that is matched by multiple strings → [_a-z A-Z][a-z A-Z 0-9 _]*

Ex → printf("Total = %d \n", score);

Lexeme is →

<id, 1> <left parenthesis> <string literal, "Total = %d \n"> <comma> <id, 2> <parenthesis> <semicolon>

* We want to define 1 token for each keyword
* Pattern for the keyword will be same as the keyword itself.
* One token identifies all the identifiers.
* Token for each punctuation symbol. Such (,), <, >, <=, etc.
* The token name influences the parsing actions.

Handling Lexical Errors

There are 4 recovery actions that lexical analyzer takes.

1) Delete 1 character from remaining input.
2) Insert a missing character in the remaining input.
3) Replace a character by another character.
4) Transpose two adjacent characters.

Special

Date .....

Alphabet :- finite set of symbols

String :- word made up of finite sequence of symbol, drawn from the alphabet.

* $|s|$ → length of string
* ϵ → string of length 0.

Language :- Countable set of strings over some fixed alphabet (maybe uncountable in some cases).

prefix → by removing 0 or more symbols from the end of string.

Total (n+1) prefixes for string of length n+1.

Suffix → by removing 0 or more symbols from the starting of string.

substring → by removing any prefix or any suffix from the string

Total $\frac{n(n+1)}{2} + 1$ substrings for string of length n.

$\left(\sum_{i=1}^{n} i + 1\right)$

Proper prefix → by removing atleast one character but not all from the string.

Subsequence → deleting 0 or more but not necessarily consecutive symbol string of s.

Total → 2^n

Spiral

Date .....

Union of L_1 & L_2 →

$$L_1 \cup L_2 = \{ s \mid s \in L_1 \text{ or } s \in L_2 \}$$

Concatenation of L_1, L_2 →

$$L_1 L_2 = \{ st \mid s \in L_1 \text{ and } t \in L_2 \}$$

Kleene's Star →

$$L^* = \bigcup_{i=0}^{\infty} L^i$$

Kleene's Closure →

$$L^+ = \bigcup_{i=1}^{\infty} L^i$$

Note →

$|L \cup D| = 62$ $\quad \{ L = \{a, b, \ldots z, A, B, \ldots Z\}$

$|L D| = 520$ $\quad \{ D = \{0, 1, 2 \ldots 9\}$

$L(L \cup D)^*$ → Set of all strings starting with a letter.

$L(a) = \{a\}$ → Regular set

(a) → reg exp

The Unary Order of Precedence → ① The unary operator * has the highest precedence & left associative.

② Concatenation has second highest precedence & left associative

③ Union has the lowest precedence & left associative

Spiral

yytext → Accepted strings by the pattern
& pointer to char.

Date .....

Ex - Parenthesizing the reg exp. $a\,b^*\,c$

$$(a\,|\,((b)^*\,c))$$

Regular Definition

letter_ → A|B| ..... |Z|a|b| ..... z|_
digit → 0|1|2 ..... |9
id → letter_ (letter_ |digit)*
digits → digit +
optional fraction → .digits | ε
optional Exponent → (E(+|-|ε) digits) | ε
number → digits optional Fraction optional Exponent

In other way,

letter_ → [A-Z a-z _]
digit → [0-9]
id → letter_
digits → digit +
number → digits (.digits)? (E[+-]? digits)?

Indian Mobile Number
→ [6-9][0-9]{9}

^ → Beginning of a line ; $ → end of the line
^[a-z] → Beginning of a line
[^a-z] → Negation of a-z
r_1 / r_2 → r_1 when followed by r_2

^[^aeiou]*$ → Any line that does not contain any lowercase vowel.

Spiral

Date .....

Q Write regular definition for the following languages:-

i) All strings of lowercase letter that contain vowels in order

ii) All strings of lowercase letters in which the letters are lexicographically connect. (in ascending order)

iii) String literal

Ans → stn_vowels_in_order = a* [b-d]* e* [f-h]* i* [j-n]* o* [p-t]* u* [v-z]*

[a e i o u]*

stn → a* alpha e* x i* x o* x u*

alpha → [^a e i o u]

stn → alpha a* alpha e* alpha i* alpha o* alpha u* alpha

alpha → [^ a e i o u]

Recommended Answer →

(a) stn → con* a (con|a)* e (con|e)* i (con|i)* o (con|o)* u (con|u)*

con → [^ a e i o u] or [b-d f-h j-n p-t v-z]

(b) stn → a* b* c* ... z*

(c) stn → [a-z A-Z 0-9 _]*

[^ "]

Spiral

"hi\hello"

Date .....

Ans → ~~• "[^"]"~~

Ans →

\"((\\\")|[^"])*\"

* → Single line comment →

(\/\/)[^\n]

or

"//"[^\n]

feb 2nd, 2023

In the process of lexical analysis, if it encounters the whitespace, it does not return to the parser, rather it restarts the process.

Q → Lex program that copies a file, replacing each non-empty sequence of white space by a single blank.

%{

#include<stdio.h>

%}

Date .....

```
%%
[" "\t]+    { fprintf (yyout," "); }
.|\n        { fprintf (yyout, "%s", yytext); }
%%
int main() {
  yyin = fopen ("input.txt", "r");
  yyout = fopen ("output.txt", "w");
  yylex();
  fclose(yyin);
  fclose(yyout);
} return 0;
```