Tutorial 01: CS31003: Compilers:

[M-01]: Phases of a Compiler

[M-02]: Lexical Analysis & Flex Specs.

Indranil Sengupta
Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

isg@iitkgp.ac.in
ppd@cse.iitkgp.ac.in

September 12, 2020

Doubts from the Week

Phased Output

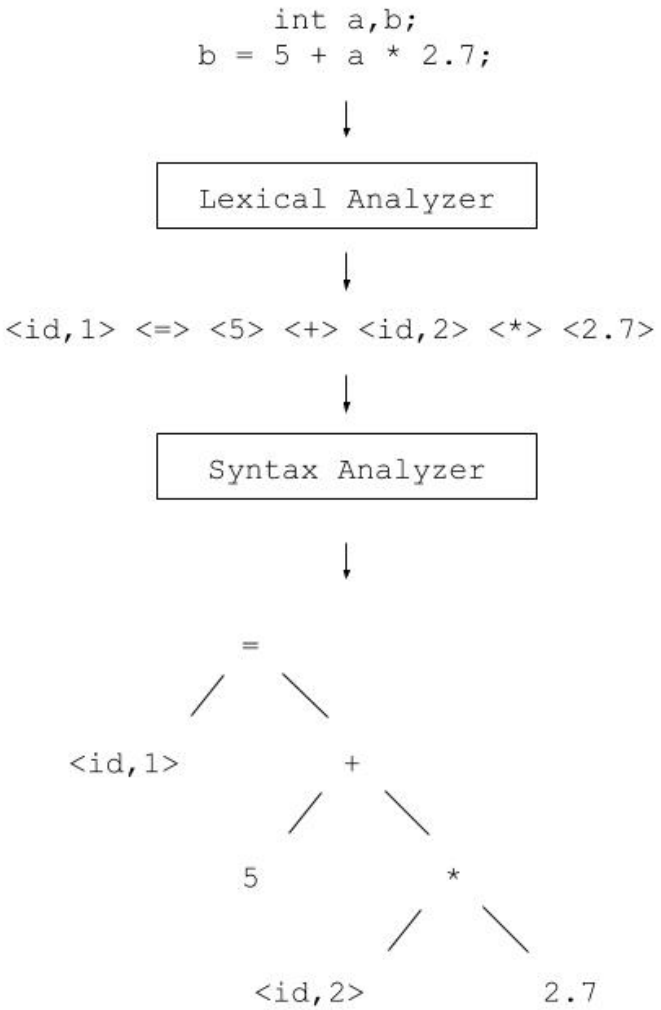
Consider the following assignment in C along with the types of variables:

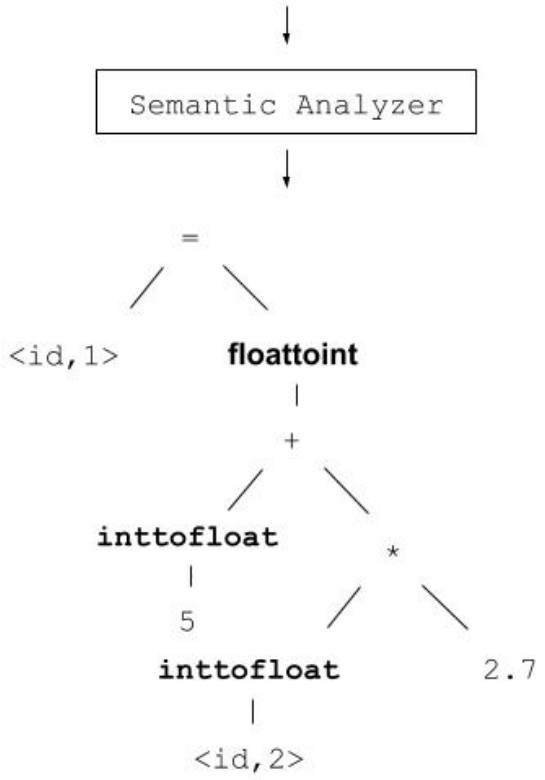
```
int a, b;
b = 5 + a * 2.7;
```

Show the output after every phase of the compiler:

1. Front-End
   a. Lexical Analysis
   b. Syntax Analysis
   c. Semantic Analysis
   d. Intermediate Code Generator
   e. Code Optimization
2. Back-End
   a. Code Optimization
   b. Target Code Generation

Phased Output: Solution

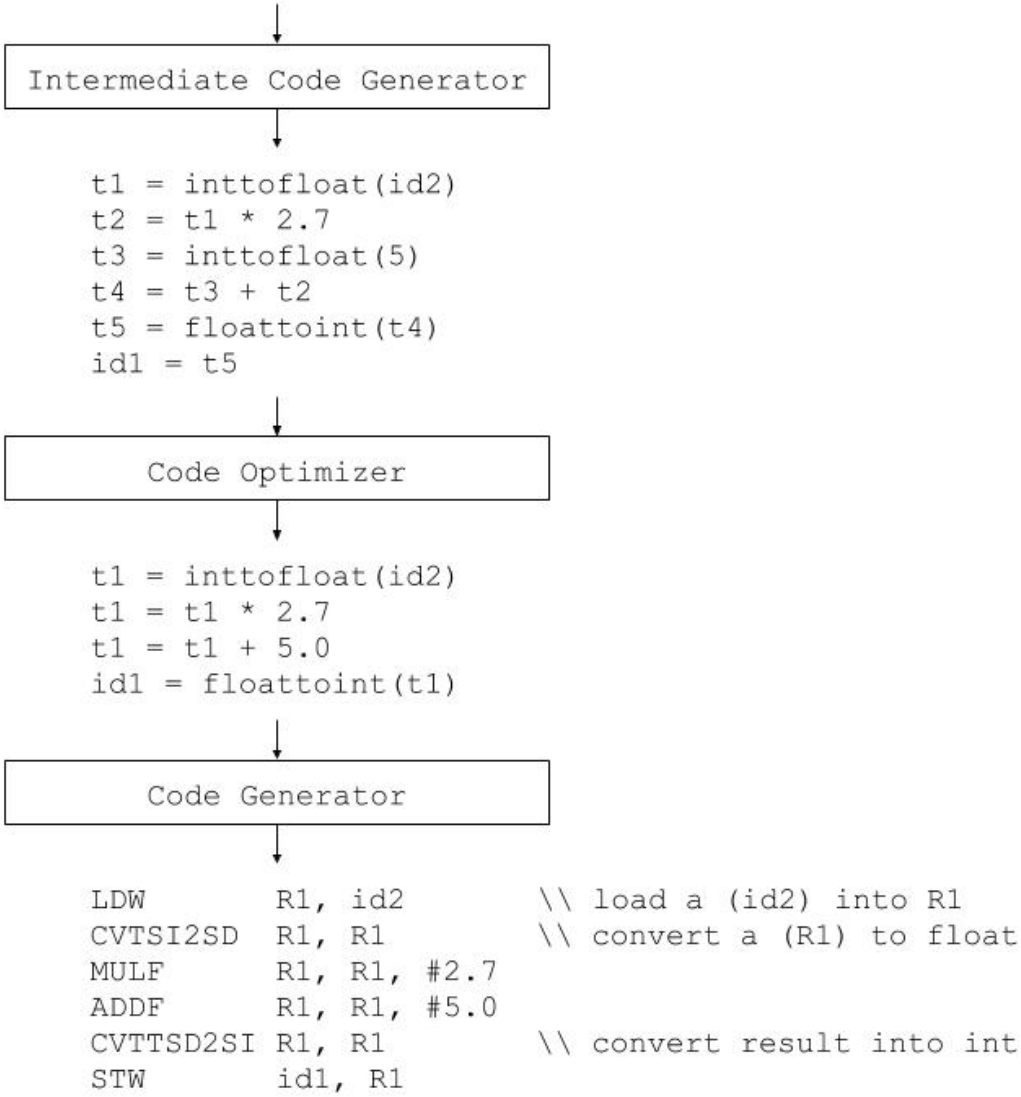




Symbol Table

1	b	...
2	a	...

Phased Output: Solution



C Comments

Consider the comments in C:

> *A sequence of characters preceded by /* and followed by */, and not containing any occurrence of */*

To lexically suppress comments, we need to recognize them:

1. Write a regular expression for comments in C
2. Draw an NFA for comments in C
3. Convert the NFA to DFA
4. Mark the lexical action in every final state of DFA

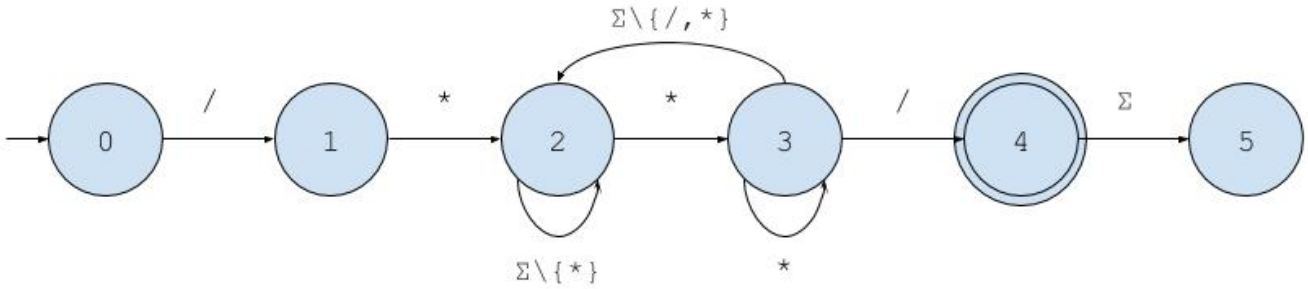
C Comments: Solution

Regular Expression: /*([^*]|*+[^*/])**/

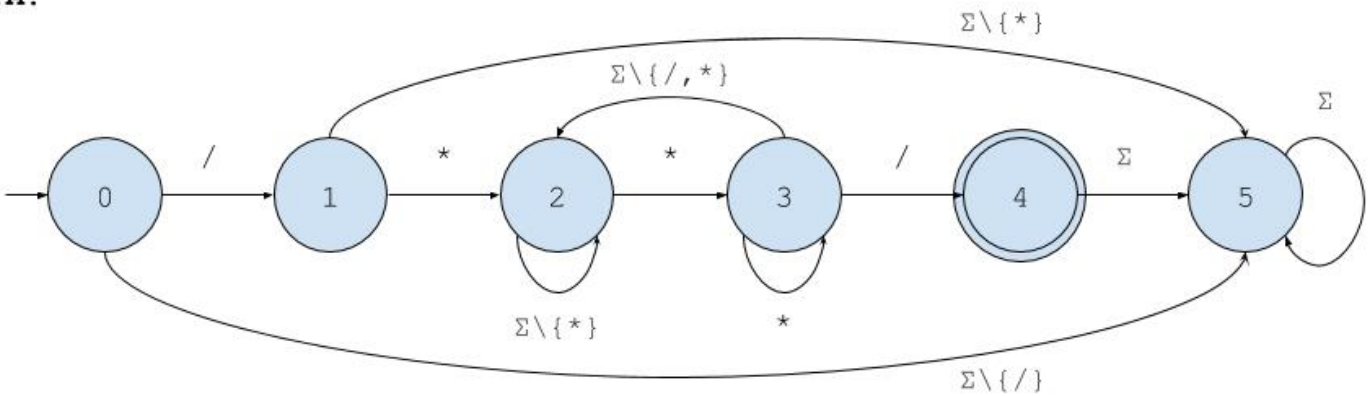
- /* : Comment starts with /*
- [^*] : accept any character except "*"
- *+[^*/] : accept one or more "*" not followed by "*/"
- ([^*]|*+[^*/])* : accept characters not "*" or "*" not followed by "*/"
- */ : Comment ends with */

* This does not include escape sequences like \r, \t, \n, etc

NFA:



DFA:



C Comments: Solution using Start Condition


```
%x comment
%%
int line_num = 1;

"/*"                    BEGIN(comment);

<comment>[^*\n]*        /* eat anything that's not a '*' */
<comment>"*"+[^*/\n]*   /* eat up '*'s not followed by '/'s */
<comment>\n             ++line_num;
<comment>"*"+"/"        BEGIN(INITIAL);
```

Source: https://ftp.gnu.org/old-gnu/Manuals/flex-2.5.4/html_node/flex_11.html

Flex Specs: Lexical Grammar

Consider the following lexical grammar

token:
    keyword | *identifier* | *constant* | *punctuator*
keyword:
    int
identifier:
    identifier-nondigit | *identifier identifier-nondigit* | *identifier digit*
identifier-nondigit: any one of
    lower case letters | upper case letters
digit: one of
    0 1 2 3 4 5 6 7 8 9
constant:
    non-zero-constant | **0**
non-zero-constant:
    nonzero-digit | *non-zero-constant digit*
nonzero-digit: one of
    1 2 3 4 5 6 7 8 9
punctuator: one of
    = ;

Flex Specs: Flex Specification

and the corresponding Flex specification as coded below:

```
%{ /* C Declarations and Definitions */
%}
INT          "int"
ID           [a-z][a-z0-9]*
PUNC         [;]
CONST        [1-9][0-9]*
WS           [ \t\n]
%%
{ID}         { printf("<ID, %s>\n", yytext); /* Identifier Rule */}
{INT}        { printf("<KEYWORD, int>\n"); /* Keyword Rule */ }
"="          { printf("<ASSIGNMENT>\n"); /* Punctuator Rule */ }

{PUNC}       { printf("<SEMICOLON>\n"); /* Punctuator Rule */ }
{CONST}      { printf("<CONSTANT, %s>\n",yytext); /* Constant Rule */ }
{WS}         /* White-space Rule */ ;
\n|.         /* Ignore Rule */ ;
%%
```

Flex Specs: Problems

1. Write the output of the lexical analyser for the following input:

```
int x;
int y;
x = 2;
y = 0;
```

2. Is the output correct according to the lexical grammar? If not, explain the problem(s) and accordingly correct the Flex specification.

Flex Specs: Solution to Problem 1

Input

```
int x;
int y;
x = 2;
y = 0;
```

Output

```
<ID, int>
<ID, x>
<SEMICOLON>
<ID, int>
<ID, y>
<SEMICOLON>
<ID, x>
<ASSIGNMENT>
<CONSTANT, 2>
<SEMICOLON>
<ID, y>
<ASSIGNMENT>
<SEMICOLON>
```

Flex Specs: Solution to Problem 2

There are two problems:

1. Keyword **int** is returned as `ID` because `ID` and `INT` rules are in wrong order.
2. Single constant **0** has not been coded.

Corrected specification would be:

```
%{ /* C Declarations and Definitions */
%}
INT         "int"
ID          [a-zA-Z][a-zA-Z0-9]*
PUNC        [;]
CONST       0|[1-9][0-9]*
WS          [ \t\n]
%%
{INT}       { printf("<KEYWORD, int>\n"); /* Keyword Rule */ }
{ID}        { printf("<ID, %s>\n", yytext); /* Identifier Rule */}
"="         { printf("<ASSIGNMENT>\n"); /* Punctuator Rule */ }

{PUNC}      { printf("<SEMICOLON>\n"); /* Punctuator Rule */ }
{CONST}     { printf("<CONSTANT, %s>\n",yytext); /* Constant Rule */ }
{WS}        /* White-space Rule */ ;
\n|.        /* Ignore Rule */ ;
%%
```

Practice Problems

1. Write Flex specifications (only the Regular Definitions) for the following numeric literals of C language:
   - Decimal Integer Constant
   - Octal Constant
   - Hexadecimal Constant
   - Floating Point Constant
2. Write the Flex specification to match C-style quoted strings using exclusive start conditions, including expanded escape sequences
   - Improve the specification above to include checking for a string that is too long
3. Write Flex specifications for the C Pre-Processor considering the directives: #define, #include, #ifdef, #endif, #if, #else, #ifndef, #undef, and #pragma