CS31003: Compilers

Partha Pratim Das

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

ppd@cse.iitkgp.ernet.in

July 28, 2014

Our sample

- We used this example to understand phases of a compiler
- This is a simple block with declaration and expression statements

```
{
    int x;
    int y;
    x = 2;
    y = 3;
    x = 5 + y * 4;
}
```

Flex Specs for our sample

- C Declarations and definitions
- Definitions of Regular Expressions
- Definitions of Rules & Actions
- C functions

```
%{
/* C Declarations and Definitions */
%}
 /* Regular Expression Definitions */
INT         "int"
ID          [a-z][a-z0-9]*
PUNC        [;]
CONST       [0-9]+
WS          [ \t\n]

%%
{INT}       { printf("<KEYWORD, int>\n); /* Keyword Rule */ }
{ID}        { printf("<ID, %s>\n", yytext); /* Identifier Rule */}
"+"         { printf("<OPERATOR, +>\n"); /* Operator Rule */ }
"*"         { printf("<OPERATOR, *>\n"); /* Operator Rule */ }
"="         { printf("<OPERATOR, =>\n"); /* Operator Rule */ }
"{"         { printf("<SPECIAL SYMBOL, {>\n"); /* Scope Rule */ }
"}"         { printf("<SPECIAL SYMBOL, }>\n"); /* Scope Rule */ }
{PUNC}      { printf("<PUNCTUATION, ;>\n"); /* Statement Rule */ }
{CONST}     { printf("<INTEGER CONSTANT, %s>\n",yytext); /* Literal Rule */ }
{WS}        /* White-space Rule */ ;
%%

main() {
    yylex(); /* Flex Engine */
}
```

Flex I/O for our sample

I/P Character Stream

```
{
    int x;
    int y;
    x = 2;
    y = 3;
    x = 5 + y * 4;
}
```

O/P Token Stream

```
<SPECIAL SYMBOL, {>
<KEYWORD, int> <ID, x> <PUNCTUATION, ;>
<KEYWORD, int> <ID, y> <PUNCTUATION, ;>
<ID, x> <OPERATOR, => <INTEGER CONSTANT, 2> <PUNCTUATION, ;>
<ID, y> <OPERATOR, => <INTEGER CONSTANT, 3> <PUNCTUATION, ;>
<ID, x> <OPERATOR, => <INTEGER CONSTANT, 5> <OPERATOR, +>
<ID, y> <OPERATOR, *> <INTEGER CONSTANT, 4> <PUNCTUATION, ;>
<SPECIAL SYMBOL, }>
```

- The output is generated as one token per line. It has been rearranged here for better readability.
- Every token is a doublet showing the token class and the specific token information.

Wrong Flex Specs for our sample

- Rules for ID and INT have been swapped.
- No keyword can be tokenized as keyword now.

```
%{
/* C Declarations and Definitions */
%}
 /* Regular Expression Definitions */
INT         "int"
ID          [a-z][a-z0-9]*
PUNC        [;]
CONST       [0-9]+
WS          [ \t\n]

%%
{ID}        { printf("<ID, %s>\n", yytext); /* Identifier Rule */}
{INT}       { printf("<KEYWORD, "int">\n"); /* Keyword Rule */ }
"+"         { printf("<OPERATOR, +>\n"); /* Operator Rule */ }
"*"         { printf("<OPERATOR, *>\n"); /* Operator Rule */ }
"="         { printf("<OPERATOR, =>\n"); /* Operator Rule */ }
"{"         { printf("<SPECIAL SYMBOL, {>\n"); /* Scope Rule */ }
"}"         { printf("<SPECIAL SYMBOL, }>\n"); /* Scope Rule */ }
{PUNC}      { printf("<PUNCTUATION, ;>\n"); /* Statement Rule */ }
{CONST}     { printf("<INTEGER CONSTANT, %s>\n",yytext); /* Literal Rule */ }
{WS}        /* White-space Rule */ ;
%%

main() {
    yylex(); /* Flex Engine */
}
```

Wrong Flex I/O for our sample

I/P Character Stream	O/P Token Stream
`{`	`<SPECIAL SYMBOL, {>`
`  int x;`	`<ID, int> <ID, x> <PUNCTUATION, ;>`
`  int y;`	`<ID, int> <ID, y> <PUNCTUATION, ;>`
`  x = 2;`	`<ID, x> <OPERATOR, => <INTEGER CONSTANT, 2> <PUNCTUATION, ;>`
`  y = 3;`	`<ID, y> <OPERATOR, => <INTEGER CONSTANT, 3> <PUNCTUATION, ;>`
`  x = 5 + y * 4;`	`<ID, x> <OPERATOR, => <INTEGER CONSTANT, 5> <OPERATOR, +>`
`}`	`<ID, y> <OPERATOR, *> <INTEGER CONSTANT, 4> <PUNCTUATION, ;>`
	`<SPECIAL SYMBOL, }>`

- Both `int`'s have been taken as `ID`!

Flex Specs (interactive) for our sample


```
%{
#define INT          10
#define ID           11
#define PLUS         12
#define MULT         13
#define ASSIGN       14
#define LBRACE       15
#define RBRACE       16
#define CONST        17
#define SEMICOLON    18
%}

INT         "int"
ID          [a-z][a-z0-9]*
PUNC        [;]
CONST       [0-9]+
WS          [ \t\n]

%%
{INT}    { return INT; }
{ID}     { return ID; }
"+"      { return PLUS; }
"*"      { return MULT; }
"="      { return ASSIGN; }
"{"      { return LBRACE; }
"}"      { return RBRACE; }
{PUNC}   { return SEMICOLON; }
{CONST}  { return CONST; }
{WS}     {/* Ignore
             whitespace */}
%%
```

```
main() { int token;
    while (token = yylex()) {
        switch (token) {
            case INT: printf("<KEYWORD, %d, %s>\n",
                token, yytext); break;
            case ID: printf("<IDENTIFIER, %d, %s>\n",
                token, yytext); break;
            case PLUS: printf("<OPERATOR, %d, %s>\n",
                token, yytext); break;
            case MULT: printf("<OPERATOR, %d, %s>\n",
                token, yytext); break;
            case ASSIGN: printf("<OPERATOR, %d, %s>\n",
                token, yytext); break;
            case LBRACE: printf("<SPECIAL SYMBOL, %d, %s>\n",
                token, yytext); break;
            case RBRACE: printf("<SPECIAL SYMBOL, %d, %s>\n",
                token, yytext); break;
            case SEMICOLON: printf("<PUNCTUATION, %d, %s>\n",
                token, yytext); break;
            case CONST: printf("<INTEGER CONSTANT, %d, %s>\n",
                token, yytext); break;
        }
    }
}
```

– Input is taken from `stdin`. It can be changed by opening the file in `main()` and setting the file pointer to `yyin`.
– When the lexer will be integrated with the YACC generated parser, the `yyparse()` therein will call `yylex()` and the `main()` will call `yyparse()`.

Flex I/O (interactive) for our sample

I/P Character Stream

```
{
    int x;
    int y;
    x = 2;
    y = 3;
    x = 5 + y * 4;
}
```

```
#define INT          10
#define ID           11
#define PLUS         12
#define MULT         13
#define ASSIGN       14
#define LBRACE       15
#define RBRACE       16
#define CONST        17
#define SEMICOLON    18
```

O/P Token Stream

```
<SPECIAL SYMBOL, 15, {>
<KEYWORD, 10, int>
<IDENTIFIER, 11, x>
<PUNCTUATION, 18, ;>
<KEYWORD, 10, int>
<IDENTIFIER, 11, y>
<PUNCTUATION, 18, ;>
<IDENTIFIER, 11, x>
<OPERATOR, 14, =>
<INTEGER CONSTANT, 17, 2>
<PUNCTUATION, 18, ;>
<IDENTIFIER, 11, y>
<OPERATOR, 14, =>
<INTEGER CONSTANT, 17, 3>
<PUNCTUATION, 18, ;>
<IDENTIFIER, 11, x>
<OPERATOR, 14, =>
<INTEGER CONSTANT, 17, 5>
<OPERATOR, 12, +>
<IDENTIFIER, 11, y>
<OPERATOR, 13, *>
<INTEGER CONSTANT, 17, 4>
<PUNCTUATION, 18, ;>
<SPECIAL SYMBOL, 16, }>
```

- Every token is a triplet showing the token class, token manifest constant and the specific token information.

Flex-Bison Flow