

# CS5103: Computing Lab-1

## Lex & Yacc



**Arijit Mondal**

**Dept of CSE**

`arijit@iitp.ac.in`

`https://www.iitp.ac.in/~arijit/`

# Introduction

- Dividing the input into meaningful units
- Control flow is directed by instance of regular expressions in the input stream
- Matches input stream against the table of regular expressions supplied
- Carries out the associated action when a match is found



# Structure of Lex file

%{

definitions

%}

%%

rules

%%

user subroutines

- Rules: `reg_exp {action}`
- `reg_exp` – Starts at the beginning of line and continue till first unescaped white space
- `action` – C statement

# Example

```
1. %{
2. /* simple example */
3. %}

4. %%
5. [\t ]+      {}
6. is | are    {printf("%s: verb\n",yytext);}
7. simple | easy {printf("%s: adjective\n",yytext);}
8. to | from   {printf("%s: preposition\n",yytext);}
9. [a-zA-Z]+   {printf("%s: may be noun\n",yytext);}
10. [0-9]+     {printf("%s: number\n",yytext); return atoi(yytext);}
11. .|\n      {}

12. %%
13. main(){
14.     yylex();
15. }
```

# Compilation

```
$> lex file_name.l
```

```
$> gcc lex.yy.c -o a.out -ll
```

# Rules for matching

- The **longest** match is preferred.
- Among rules that match the same number of characters, the rule that occurs **earliest** in the list is preferred.

# Exclusive state

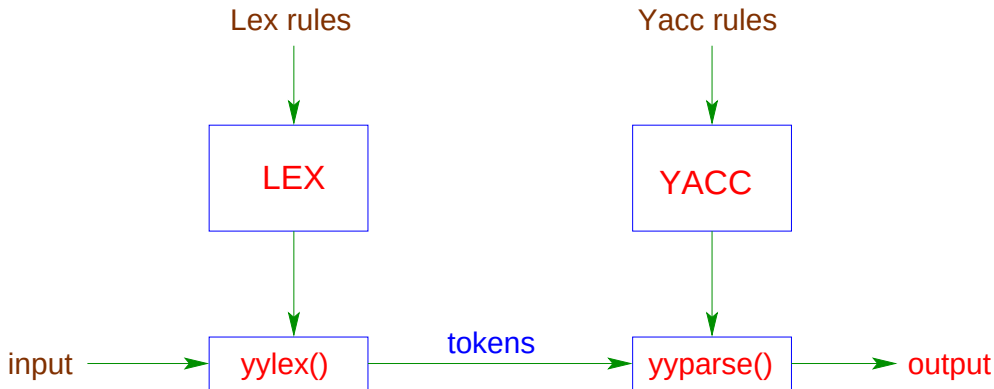
- Sometimes it is desirable to have several sets of lexical rules to be applied at different times in the input.
- Declare a set of **start condition**, s - inclusive, x - exclusive
- Example:

```
%{  
%}  
%x COMMENT  
%%  
"/*"           {BEGIN COMMENT;}  
"<COMMENT>[^*/]"  {}  
"<COMMENT>\"*/\"    {BEGIN 0;}
```

# Communicating with user program

- `yytext` – A character array that contains the string that matched a pattern
- `yylen` – The number of characters matched

# Lex & Yacc: Combined flow



# Structure of YACC file

declarations

%%

grammar rules

%%

user subroutines

- Grammar rules: Production rules eg.  $A \rightarrow B_1 B_2$ ,  $B_1 \rightarrow C_1 C_2$
- Consists of terminal and non-terminal symbols
- Terminal symbols are to be defined as token in the definition section

# Example

Input:

```
a = 5 + 6 ;
```

```
b = 5 + 6 + 9 - 6;
```

Output:

```
a = 11
```

```
b = 14
```

# Example

```
%{ ... %}  
%union { int ival; char *sval; }  
%token <ival> INT  
%token <sval> STRING  
%start statements  
%type <ival> mathexp  
%left '+' '-'  
%%  
statements:  
    | statements endexpression {} ;  
endexpression: STRING '=' mathexp ';' {printf("%s = %d\n",$1,$3);} ;  
mathexp: mathexp '+' mathexp          {$$=$1+$3;}  
        | mathexp '-' mathexp          {$$=$1-$3;}  
        | INT                          {$$=$1;} ;  
%%  
main(){ yyparse(); }  
yyerror(char *s){ printf("Error: %s\n",s); }
```

# Example

```
%{  
#include "y.tab.h"  
%}  
%%  
[\\t ]+      {}  
[0-9]+       {yyval.ival=atoi(yytext); return INT;}  
[a-zA-Z]+    {yyval.sval=(char *)malloc(yyleng+1);  
              strcpy(yyval.sval,yytext); return STRING;}  
"="          {return yytext[0];}  
"+"          {return yytext[0];}  
"-"          {return yytext[0];}  
";"          {return yytext[0];}  
"\\n"        {}  
%%
```

# Compilation

```
$> yacc -d -v file_name.y
```

```
$> lex file_name.l
```

```
$> gcc lex.yy.c y.tab.c -o a.out -ll
```

# Conflicts

- Reduce/Reduce conflict

```
start: a Y  
      | b Z ;  
a : X  
b : X
```

- Shift/Reduce conflict

```
Statement : if ( Expr ) Statement  
          | if ( Expr ) Statement else Statement
```

# Handling shift/reduce conflict

```
%nonassoc LOWER_THAN_ELSE
```

```
%nonassoc ELSE
```

```
%%
```

```
if_statement
```

```
    : IF '(' expression ')' statement %prec LOWER_THAN_ELSE
```

```
    | IF '(' expression ')' statement ELSE statement
```

```
    ;
```

# Makefile-1

- We need to specify dependencies and action / recipe to be taken

```
target: prerequisites  
    <tab>recipe
```

- Example: Let us say we have the following files
  - One library for complex number operations `complex.h` and `complex.c`
  - One library for queries related to geometry `geometry.h` and `geometry.c`
  - One `main.c` file that includes both `complex` and `geometry` header files and calls some of the routines.

# Makefile-2

```
CC=gcc
```

```
DEBUG=-g
```

```
FLAGS=-c
```

```
a.out: main.c complex.o geometry.o
```

```
<tab>${CC} ${DEBUG} main.c complex.o geometry.o
```

```
complex.o: complex.h complex.c
```

```
<tab>${CC} ${FLAGS} ${DEBUG} complex.c
```

```
geometry.o: geometry.h geometry.c
```

```
<tab>${CC} ${FLAGS} ${DEBUG} geometry.c
```

```
clean:
```

```
<tab>rm -rf a.out complex.o geometry.o
```