

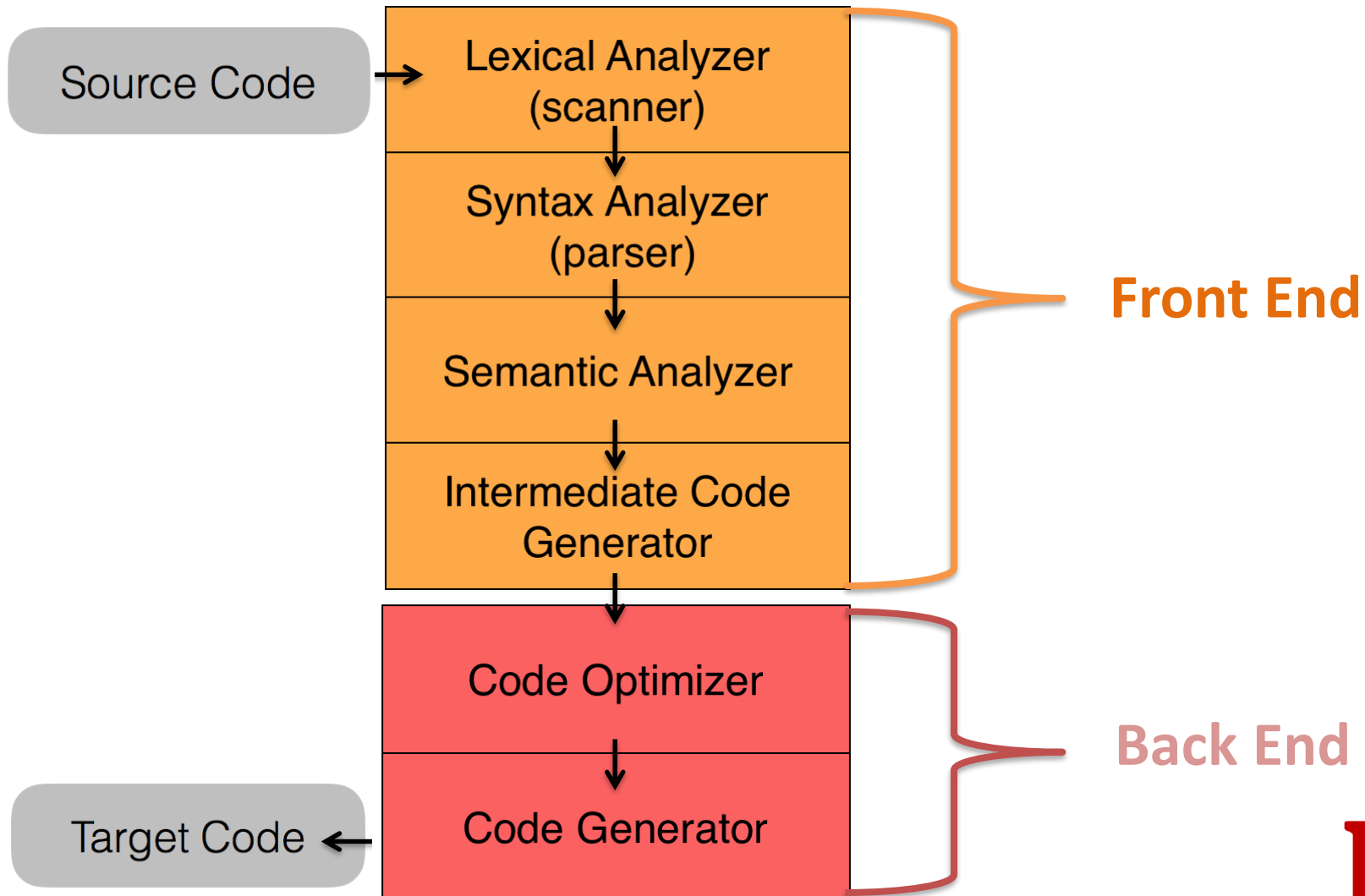
CS340400 Compiler Design Homework 1

Deadline

2019/04/10(Wed.) PM12:00

Overview

The Structure of a Modern Compiler



What are we Going to do?

HW1

source code

`a = b + c * d`

Lexical Analyzer

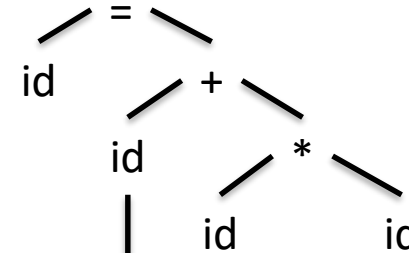
tokens

`id = id + id * id`

HW2

Syntax Analyzer

syntax tree



HW3

Code Generator

generated code

`mul $r1, $r2, $r3`
`add $r0, $r1, $r4 ...`

The Structure of Compiler

HW1

source code

`a = b + c * d`

Lexical Analyzer

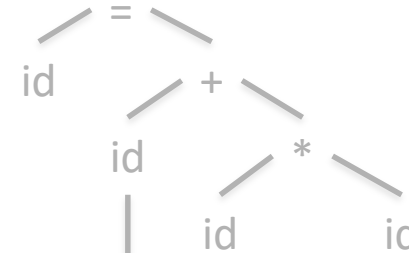
tokens

`id = id + id * id`

HW2

Syntax Analyzer

syntax tree



HW3

Code Generator

generated code

`mul $r1, $r2, $r3`
`add $r0, $r1, $r4 ...`

Lex

- lexical analyser generator

What is Lex?

- Lex is a **program generator** designed for lexical processing of character input streams. It accepts a high-level, problem oriented specification for character string matching, and produces a program in a general purpose language which recognizes **regular expressions**.
- The regular expressions are specified by the user in the source specifications given to Lex.
- Lex generates a **deterministic finite automaton (DFA)** from the regular expressions in the source.
- The Lex written code recognizes these expressions in an input stream and partitions the input stream into strings matching the expressions.

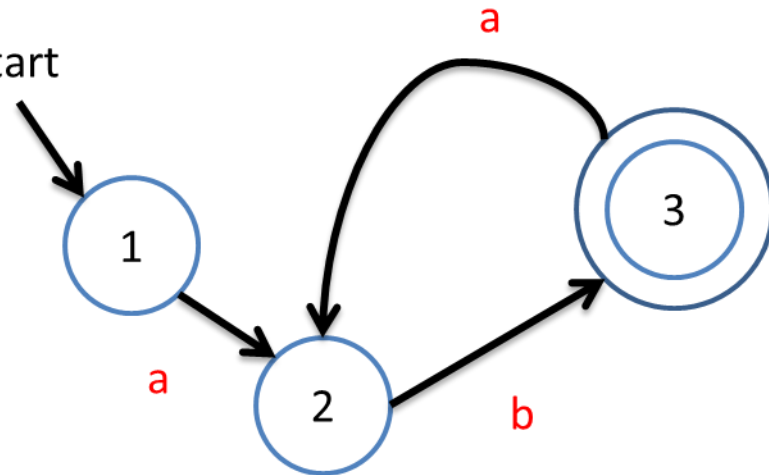
From <http://dinosaur.compilertools.net/lex/>

$(ab)^+$

LEX

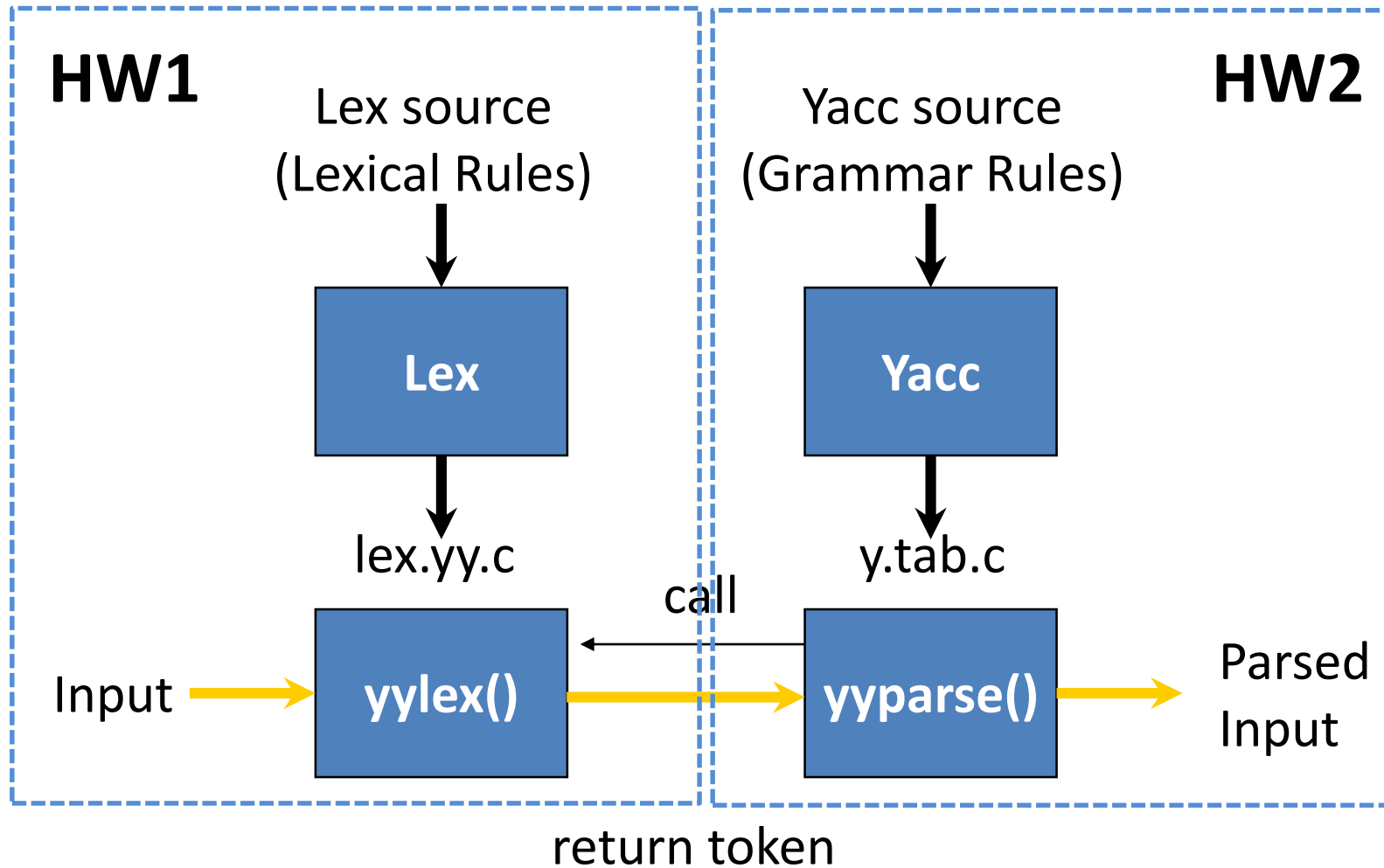
Input Stream

start

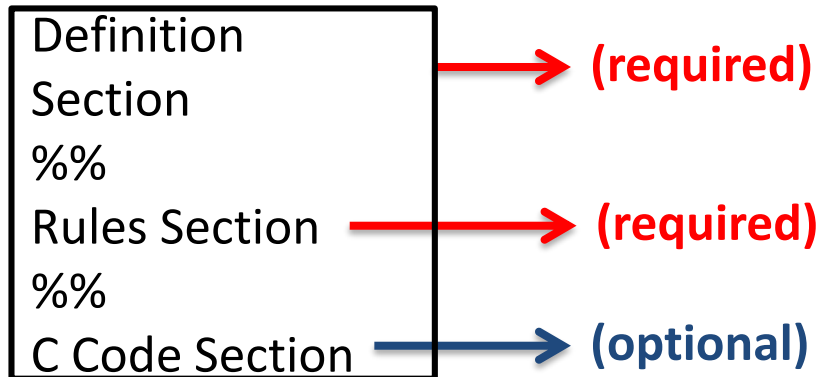


If (matches the DFA expression)
Do Something.....

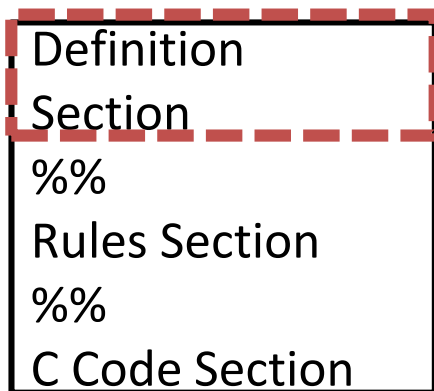
Lex with Yacc



How to Write Lex?



How to Write Lex?

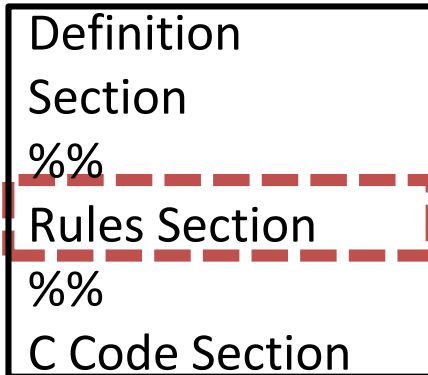


```
%{
#include <stdio.h>

int      lineCount=0;
%}
```

The Definition Section will be copied to the top of generated C program. Include header files, declare variables.

How to Write Lex?



```
\n    { lineCount++;  
      printf("line:%d\n", lineCount); }
```

The Rules Section is for writing regular expression to recognize tokens. When **pattern** is matched, then execute **action**

[Regular expression rule] { The things you want to do; }

How to Write Lex?

Definition
Section
%%
Rules Section
%%
C Code Section

```
int main(void) {  
    yylex();  
    return 0;  
}  
  
int yywrap() {  
    return 1;  
}  
  
// Other function you defined.
```

The C Code Section will be copied to the bottom of generated C program.

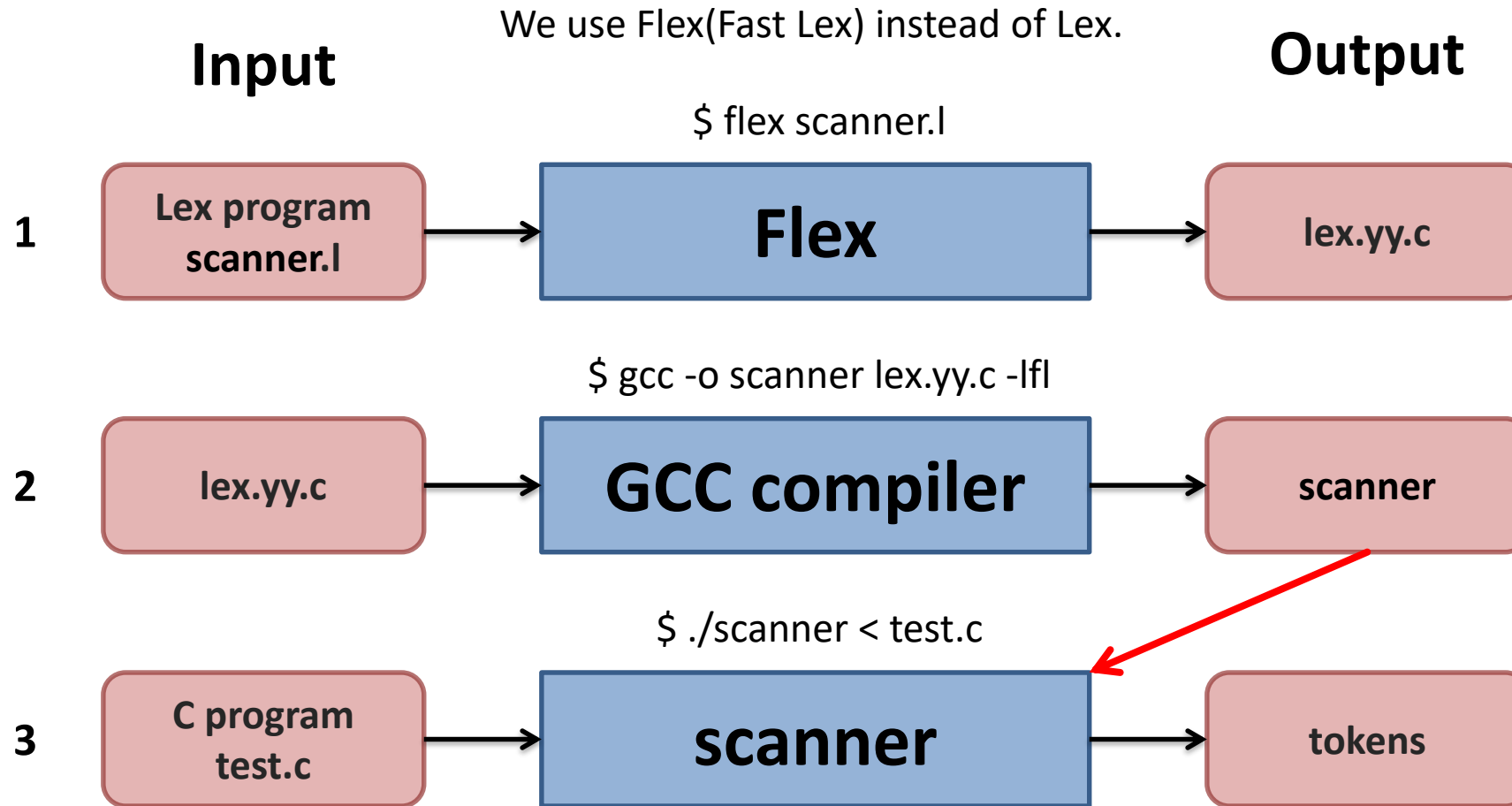
How to Write Lex?

A completed Lex program

Definition
Section
%%
Rules Section
%%
C Code Section

```
%{  
#include <stdio.h>  
  
int      lineCount=0;  
%}  
  
%%  
\n      { lineCount++;  
        printf("line:%d\n", lineCount); }  
  
%%  
int main(void){  
    yylex();  
    return 0;  
} .....
```

Compilation Flow



Flex: the Fast Lexical Analyser Generator

3 Introduction

'flex' is a tool for generating "scanners". A scanner is a program which recognizes lexical patterns in text. The 'flex' program reads the given input files, or its standard input if no file names are given, for a description of a scanner to generate. The description is in the form of pairs of regular expressions and C code, called "rules". 'flex' generates as output a C source file, 'lex.yy.c' by default, which defines a routine 'yylex()'. This file can be compiled and linked with the flex runtime library to produce an executable. When the executable is run, it analyzes its input for occurrences of the regular expressions. Whenever it finds one, it executes the corresponding C code.

The 'flex' input file consists of three sections, separated by a line containing only '%%'.

definitions

%%

rules

%%

user code

Link with
library **libfl.a**

Flex Example:

Count Number of Lines and Number of Characters

count_line.l

```
1 %{
2
3 #include <stdio.h>
4 int num_lines = 0, num_chars = 0;
5
6 %}
7
8 %%
9
10 \n { ++num_lines ; ++ num_chars ; }
11 . { ++num_chars ; }
12
13 %%
14
15 int main(int argc, char* argv[])
16 {
17     yylex() ;
18     printf("# of lines = %d, # of chars = %d\n",
19         num_lines, num_chars );
20     return 0 ;
21 }
```

```
[ims1@linux count_line]$ ./a.out
This is a book
byebye ← Press Enter
# of lines = 2, # of chars = 22
[ims1@linux count_line]$
```

Press Ctrl+D

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| T | h | i | s | | i | s | | a | | b | o | o | k | \n |

| | | | | | | |
|---|---|---|---|---|---|----|
| b | y | e | b | y | e | \n |
|---|---|---|---|---|---|----|

```
[ims1@linux count_line]$
[ims1@linux count_line]$ flex count_line.l
[ims1@linux count_line]$ ls
count_line.l  lex.yy.c
[ims1@linux count_line]$ gcc lex.yy.c -lfl
[ims1@linux count_line]$ ls
a.out count_line.l lex.yy.c
[ims1@linux count_line]$
```

Generate source C-code **lex.yy.c**

Library **libfl.a** 17

Grammar of Input file of Flex

Lex copy data enclosed by **%{** and **%}** into C source file

pattern

action

\n { ++num_lines ; ++ num_chars ; }

• { ++ num_chars ; }

↑
wild card character, represent any
character expect line feed **\n**

User
code

```
1 %{  
2  
3 #include <stdio.h>  
4 int num_lines = 0, num_chars = 0;  
5  
6 %}  
7  
8 %%  
9  
10 \n { ++num_lines ; ++ num_chars ; }  
11 . { ++num_chars ; }  
12  
13 %%  
14  
15 int main(int argc, char* argv[])  
16 {  
17     yylex() ;  
18     printf("# of lines = %d, # of chars = %d\n",  
19         num_lines, num_chars );  
20     return 0 ;  
21 }
```

grammar of input file

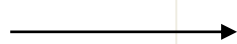
definition section

%%

rule section

%%

user code



pattern action

When **pattern** is matched,
then execute **action**

Can we Compile lex.yy.c without -lfl ?

We want to use **lex.yy.c** on different platforms (Linux and windows), to avoid specific library is lesson one.

```
[ims1@linux count_line]$ gcc lex.yy.c
/tmp/ccgm0gZ8.o(.text+0x30d): In function `yylex':
: undefined reference to `yywrap'
/tmp/ccgm0gZ8.o(.text+0xa4f): In function `input':
: undefined reference to `yywrap'
collect2: ld returned 1 exit status
[ims1@linux count_line]$
```

Library **libfl.a** contains function *yywrap()*

-lfl means “include library **libfl.a**” , this library locates in **/usr/lib**

```
[ims1@linux lib]$ pwd
/usr/lib
[ims1@linux lib]$ ls libf*
libfam.a      libfam.so.0.0.0    libfontconfig.so.1.0  libform.so.5.3    libfreetype.so.6
libfam.la     libfl.a            libform.a             libfreetype.a     libfreetype.so.6.3.2
libfam.so     libfontconfig.so   libform.so            libfreetype.la
libfam.so.0   libfontconfig.so.1 libform.so.5          libfreetype.so
[ims1@linux lib]$ ar -t libfl.a
libmain.o
libyywrap.o
[ims1@linux lib]$
```

→ contains function *main()* and *yywrap()*

Can we Compile lex.yy.c without -lfl ?

count_line.l

```
%{
#include <stdio.h>
int num_lines = 0, num_chars = 0;

%}

%%
\n      { ++num_lines ; ++ num_chars ; }
.       { ++num_chars ; }

%%

int main(int argc, char* argv[])
{
    yylex() ;
    printf("# of lines = %d, # of chars = %d\n",
        num_lines, num_chars );
    return 0 ;
}

/* when yylex() read a EOF, then it call yywrap().
 * Return value of yywrap() is either 0 or 1.
 * if return value is 1, then it means NO any input,
 *   program is end ( yylex() return 0 )
 * if return value is 0, then tells yylex() that
 *   new file is ready, it can go on to process new token.
 *
 * Hence if we have multiple files to be parsed, then we can use yywrap() to
 *   open file one by one
 */

int yywrap()
{
    return 1 ; /* eof */
}
```

Implement function **yywrap**
explicitly

How to Process a file?

count_line.l

```
%%
\n      {
        ++num_lines ;
        ++ num_chars ;
      }
.       {
        ++num_chars ;
      }
%%

int main(int argc, char* argv[])
{
  ++argv ;
  --argc ; /* skip over program name*/

  if ( 0 < argc ){
    yyin = fopen( argv[0], "r" ) ;
  }else{
    yyin = stdin ;
  }
  yylex() ;
  printf("# of lines = %d, # of chars = %d\n",
        num_lines, num_chars );
  return 0 ;
}

/* when yylex() read a EOF, then it call yywrap().
 * Return value of yywrap() is either 0 or 1.
 * if return value is 1, then it means NO any input,
```

lex.yy.c

```
/* Translate the current start state into a value that can b
 * to BEGIN to return to the state.  The YYSTATE alias is fo
 * compatibility.
 */
#define YY_START ((yy_start - 1) / 2)
#define YYSTATE YY_START

/* Action number for EOF rule of a given start state. */
#define YY_STATE_EOF(state) (YY_END_OF_BUFFER + state + 1)

/* Special action meaning "start processing a new file". */
#define YY_NEW_FILE yyrestart( yyin )

#define YY_END_OF_BUFFER_CHAR 0

/* Size of default input buffer. */
#define YY_BUF_SIZE 16384

typedef struct yy_buffer_state *YY_BUFFER_STATE;

extern int yyleng;
extern FILE *yyin, *yyout;

#define EOB_ACT_CONTINUE_SCAN 0
#define EOB_ACT_END_OF_FILE 1
#define EOB_ACT_LAST_MATCH 2
```

yyin is a file pointer in **lex**, function **yylex()** read characters from **yyin**

Lex Predefined Variables

| Name | Function |
|--------------------|---|
| char *yytext | Pointer to matched string. |
| int yyleng | Length of matched string. |
| int yylex(void) | Function call to invoke lexer and return token. |
| int yywrap(void) | Return 1 if no more files to be read. |
| char *yymore(void) | Return the next token. |
| int yyless(int n) | Retain token except the first n characters in yytext. |
| FILE *yyin | Input stream pointer. |
| FILE *yyout | Output stream pointer. |
| ECHO | Print out the yytext. |
| BEGIN | Condition switch. |
| REJECT | Go to the next alternative rule. |

man flex or **info flex** to get more info

Regular Expressions

| . | Any character excepts '\n'. | $. = \{a, b, c, d, \dots\}$ |
|-------|-------------------------------------|---|
| * | Zero or more. | $ab^* = \{a, ab, abb, abbb, \dots\}$ |
| + | One or more. | $ab^+ = \{ab, abb, abbb, \dots\}$ |
| ? | Zero or one. | $a? = \{\epsilon, a\}$ |
| | Or. | $a b = \{a, b\}$ |
| [] | Any character of the character set. | $[abc] = \{a, b, c\}$ |
| () | To group characters. | $(ab)^* = \{\epsilon, ab, abab, \dots\}$ |
| \ | For escape character. | $\backslash^* = \{*\}, \backslash\backslash = \{\backslash\}$ |
| "..." | Literally. | $"a^*" = \{a^*\}$ |
| {} | Repeat. Substitution. | $a\{1,3\} = \{a, aa, aaa\}$ |
| ^ | Head of line. Exclude. | $[^a] = \{b, c, d, e, \dots\}$ |
| \$ | End of line. | |
| / | Followed with specific character | $a/b = \{a^nb\}$ |

Regular Expressions

Input
String

she

Rule
Section

%%

she { printf("she\t"); }

[sS]he { printf("another she\t"); }

he { printf("he\t"); }

s { printf("s\t"); }

%%

- The output result is "she" .
- Always choose the longest matching pattern.
- If the length are the same, choose the first met rule.

More Elegant Way to Write Regular Expressions

```
%{  
#include <stdio.h>  
  
int      lineCount=0;  
%}  
  
ch      [a-z]  
  
%%  
\n      { lineCount++;  
      printf("line:%d\n", lineCount) ;}  
  
{ch}+  { ECHO; }
```

More Elegant Way to Write Regular Expressions

```
%{  
#include <stdio.h>  
  
int      lineCount=0;  
%}  
  
%%  
\n      { lineCount++;  
      printf("line:%d\n", lineCount); }  
  
[:alpha:]+      { ECHO; }
```

Regular Expressions

| Regular Expression | Meaning |
|--------------------|-----------------------------------|
| [a-zA-Z] | Any character of a ~ z and A ~ Z. |
| [0-9] | Any character of 0 ~ 9. |
| [:lower:] | [[:lower:]] = [a-z] |
| [:upper:] | [[:upper:]] = [A-Z] |
| [:alpha:] | [[:alpha:]] = [a-zA-Z] |
| [:digit:] | [[:digit:]] = [0-9] |
| [:alnum:] | [[:alnum:]] = [a-zA-Z0-9] |

Start Condition

- What if you encounter the string like this?

Input
String

```
/* int count  
   is for counting line number */
```

Input
String

```
printf( "int is 32-bit" );
```

Start Condition

```
%{  
...  
%}  
  
%x  COMMENT  
  
%%
```

- Declare at Definition Section
- %s STATE_NAME – inclusive
 - If the start condition is *inclusive*, then rules with no start conditions at all will also be active.
- %x STATE_NAME – exclusive
 - If it is *exclusive*, then only rules qualified with the start condition will be active.

Start Condition

Input
String

/*int

%{

...

%}

%x COMMENT

/* Exclusive */

%%

“/*”

int

<COMMENT>int

%%

{ BEGIN COMMENT; }

{ printf(“normal\n”); }

{ printf(“special\n”);

BEGIN 0; }

Start Condition

Input
String

/*int

%{

...

%}

%s COMMENT

/* Inclusive */

%%

“/*”

int

<COMMENT>int

%%

{ **BEGIN COMMENT;** }

{ printf(“normal\n”); }

{ printf(“special\n”); }

BEGIN 0; }

Versions of Lex

- AT&T: lex
http://www.combo.org/lex_yacc_page/lex.html
- GNU: flex
<http://www.gnu.org/manual/flex-2.5.4/flex.html>
- a Win32 version of flex
<http://www.monmouth.com/~wstreett/lex-yacc/lex-yacc.html>
or Cygwin
<http://sources.redhat.com/cygwin/>
- Lex on different machines is not created equal.

Homework1 - Requirements

Subset of C Language

- Character Set
ASCII,
but control characters only `'\n'` and `'\t'`

| 二進位 | 十進位 | 十六進位 | 圖形 | 二進位 | 十進位 | 十六進位 | 圖形 | 二進位 | 十進位 | 十六進位 | 圖形 |
|-----------|-----|------|---------|-----------|-----|------|----|-----------|-----|------|----|
| 0010 0000 | 32 | 20 | (space) | 0100 0000 | 64 | 40 | @ | 0110 0000 | 96 | 60 | ` |
| 0010 0001 | 33 | 21 | ! | 0100 0001 | 65 | 41 | A | 0110 0001 | 97 | 61 | a |
| 0010 0010 | 34 | 22 | " | 0100 0010 | 66 | 42 | B | 0110 0010 | 98 | 62 | b |
| 0010 0011 | 35 | 23 | # | 0100 0011 | 67 | 43 | C | 0110 0011 | 99 | 63 | c |
| 0010 0100 | 36 | 24 | \$ | 0100 0100 | 68 | 44 | D | 0110 0100 | 100 | 64 | d |
| 0010 0101 | 37 | 25 | % | 0100 0101 | 69 | 45 | E | 0110 0101 | 101 | 65 | e |
| 0010 0110 | 38 | 26 | & | 0100 0110 | 70 | 46 | F | 0110 0110 | 102 | 66 | f |
| 0010 0111 | 39 | 27 | ' | 0100 0111 | 71 | 47 | G | 0110 0111 | 103 | 67 | g |
| 0010 1000 | 40 | 28 | (| 0100 1000 | 72 | 48 | H | 0110 1000 | 104 | 68 | h |
| 0010 1001 | 41 | 29 |) | 0100 1001 | 73 | 49 | I | 0110 1001 | 105 | 69 | i |
| 0010 1010 | 42 | 2A | * | 0100 1010 | 74 | 4A | J | 0110 1010 | 106 | 6A | j |
| 0010 1011 | 43 | 2B | + | 0100 1011 | 75 | 4B | K | 0110 1011 | 107 | 6B | k |
| 0010 1100 | 44 | 2C | , | 0100 1100 | 76 | 4C | L | 0110 1100 | 108 | 6C | l |
| 0010 1101 | 45 | 2D | - | 0100 1101 | 77 | 4D | M | 0110 1101 | 109 | 6D | m |
| 0010 1110 | 46 | 2E | . | 0100 1110 | 78 | 4E | N | 0110 1110 | 110 | 6E | n |
| 0010 1111 | 47 | 2F | / | 0100 1111 | 79 | 4F | O | 0110 1111 | 111 | 6F | o |
| 0011 0000 | 48 | 30 | 0 | 0101 0000 | 80 | 50 | P | 0111 0000 | 112 | 70 | p |
| 0011 0001 | 49 | 31 | 1 | 0101 0001 | 81 | 51 | Q | 0111 0001 | 113 | 71 | q |
| 0011 0010 | 50 | 32 | 2 | 0101 0010 | 82 | 52 | R | 0111 0010 | 114 | 72 | r |
| 0011 0011 | 51 | 33 | 3 | 0101 0011 | 83 | 53 | S | 0111 0011 | 115 | 73 | s |
| 0011 0100 | 52 | 34 | 4 | 0101 0100 | 84 | 54 | T | 0111 0100 | 116 | 74 | t |
| 0011 0101 | 53 | 35 | 5 | 0101 0101 | 85 | 55 | U | 0111 0101 | 117 | 75 | u |
| 0011 0110 | 54 | 36 | 6 | 0101 0110 | 86 | 56 | V | 0111 0110 | 118 | 76 | v |
| 0011 0111 | 55 | 37 | 7 | 0101 0111 | 87 | 57 | W | 0111 0111 | 119 | 77 | w |
| 0011 1000 | 56 | 38 | 8 | 0101 1000 | 88 | 58 | X | 0111 1000 | 120 | 78 | x |
| 0011 1001 | 57 | 39 | 9 | 0101 1001 | 89 | 59 | Y | 0111 1001 | 121 | 79 | y |
| 0011 1010 | 58 | 3A | : | 0101 1010 | 90 | 5A | Z | 0111 1010 | 122 | 7A | z |
| 0011 1011 | 59 | 3B | ; | 0101 1011 | 91 | 5B | [| 0111 1011 | 123 | 7B | { |
| 0011 1100 | 60 | 3C | < | 0101 1100 | 92 | 5C | \ | 0111 1100 | 124 | 7C | |
| 0011 1101 | 61 | 3D | = | 0101 1101 | 93 | 5D |] | 0111 1101 | 125 | 7D | } |
| 0011 1110 | 62 | 3E | > | 0101 1110 | 94 | 5E | ^ | 0111 1110 | 126 | 7E | ~ |
| 0011 1111 | 63 | 3F | ? | 0101 1111 | 95 | 5F | _ | | | | |

Subset of C Language

- Keywords

void int double bool char null for while do if else switch return break
continue const true false struct case default

Library functions(from stdio.h) -

https://www.tutorialspoint.com/c_standard_library/stdio_h.htm

Subset of C Language

- Identifiers (casesensitive) - C variable naming rules

- Operators

+ - * / % ++ -- < <= > >= == != = && || !

*(Pointer to a variable) &(Returns the address of a variable)

- Punctuation characters

: ; , . [] () {}

Subset of C Language

- Integer constant
- Floating point constant
- Scientific notation (E notation: aEb, aeb)
- String constant
- Variable declaration(Only consider illegal variable names)/initialization
- Types
- Simple statements
- Control structures (goto)
- Functions
- Library functions(from stdio.h)
- Arrays
- Pointers
- Comments
- Pragma directives
 - pragama source on
 - pragama source off
 - pragama token on
 - pragama token off

Note: Lexeme will not exceed 256.

Output Format

- Token type
 - Keywords **(key)** : refer to the slide page 36
 - Identifiers **(id)** : refer to the slide page 37
 - Operators **(op)** : refer to the slide page 37
 - Punctuation characters **(punc)** : refer to the slide page 37
 - Integer **(integer)**: Eg. 10, 234, ...
 - Double **(double)** : Eg. 0.9, 34.56, +.123, -.222, -0.1, ...
 - Char **(char)** : Eg. 's', 'a', ...
 - Scientific notation **(sci)** Eg. 1.23E4, 1.23E+4, 147e-1, ...
 - String **(string)** : Eg. "apple", "ddef", ...

You must follow the rules to classify the tokens and print the token types with the type names we gave in the parentheses!

Output Format

- (Each line) Legal code
 - **You must print the results with the formats described below**
 - If it is a **comment** or a **pragma directive**, just print the line number and the line.
linenum:line_content
 - Others, print token type first, then, print the line number and the line.
#token type:token1
#token type:token2
...
linenum:line_content

Output Format Examples: Test Case 0_1

char a = 'i';

Output Format Examples: Result

#key:char

#id:a

#op:=

#char:'i'

#punc:;

1:char a = 'i';

Output Format Examples: Test Case 0_2

```
1.  //This test case is only for homework explanation
2.  int main () {
3.      double a = 6.0;
4.      int i;
5.      int b[2];
6.      for (i = 0; i < 2; i++) {
7.          b[i] = i;
8.      }
9.      printf("b[1]=%d\n", b[1]);
10.     if (b[0] > 1){
11.         a = a * 1.23e-1;
12.     }
13.     return 0;
14.
15. }
```

Output Format Examples: Result

1://This test case is only for homework explanation

#key:int

#id:main

#punc:(

#punc:)

#punc:{

2:int main () {

#key:double

#id:a

#op:=

#double:6.0

#punc;;

3: double a = 6.0;

#key:int

#id:i


#punc;;

4: int i;

```
#key:int
#id:b
#punc:[
#integer:2
#punc:]
#punc;;
5:    int b[2];
#key:for
#punc:(
#id:i
#op:=
#integer:0
#punc;;
#id:i
#op:<
#integer:2
#punc;;
#id:i
#op:++
#punc:)
#punc:{
6:    for (i = 0; i < 2; i++) {
```

```
#id:b
#punc:[
#id:i
#punc:]
#op:=
#id:i
#punc;;
7:      b[i] = i;
#punc;}
8:    }
#key:printf
#punc:(
#string:b[1]=%d\n
#punc;,
#id:b
#punc:[
#integer:1
#punc:]
#punc:)
#punc;;
9:    printf("b[1]=%d\n", b[1]);
```

```
#key:if
#punc:(
#id:b
#punc:[
#integer:0
#punc:]
#op:>
#integer:1
#punc:)
#punc:{
10:    if (b[0] > 1){
#id:a
#op:=
#id:a
#op:*
#sci:1.23e-1
#punc;;
11:        a = a * 1.23e-1;
#punc;}
12:    }
```

```
#key:return  
#integer:0  
#punc;;  
13:    return 0;  
14:   
#punc:}  
15:}
```

Please use command **diff** or **vimdiff** to check whether your output format is correct or not!

Pragma Directives: Source

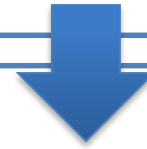
default

```
#pragma source on  
char a = 'i';
```



```
1:#pragma source on  
#key:char  
#id:a  
#op:=  
#char:'i'  
#punc;;  
2:char a = 'i';
```

```
#pragma source off  
char a = 'i';
```



```
#key:char  
#id:a  
#op:=  
#char:'i'  
#punc;;
```

Pragma Directives: Token

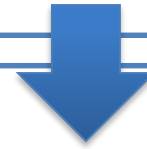
default

#pragma token on
char a = 'i';



1:#pragma token on
#key:char
#id:a
#op:=
#char:'i'
#punc;;
2:char a = 'i';

#pragma token off
char a = 'i';



1:#pragma token off
2:char a = 'i';

Grading Policies

- For all the homework,
 - Warnings: minus 20 points
 - Late homework: minus 10 points **per day**
 - Not executable (include not following the output format and not following the submission rules): **You can still get 30 points if you turn in your codes and the report.**
 - **Cheating: You will receive zero credit!**

Grading Policies

- If your scanner can classify and print the token

- Keywords, identifiers, +15
- operators, punctuation characters, +15
- integer, double, character, +15
- pragma +15
- Scientific notation and string +10
- Comment +20
- Advance Test Cases +10

Pass testcase_basic will get score in red area

Report

- For student who **can not finish** the homework,
 - Describe the abilities of your scanner
 - Describe the difficulties you faced
 - Describe the methods you tried to solve your problems

Submission

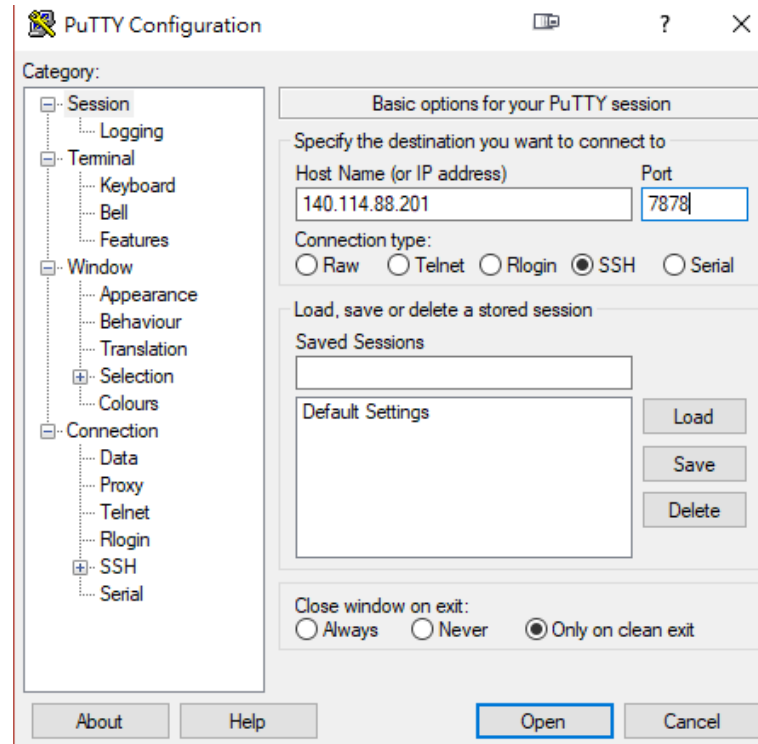
- You must upload 2 items: your source code, a makefile in server or you will get zero credit!
- Server: Source code
 - **Must** create “hw1” under your home directory
 - e.g. Student ID = 104062634
Your home directory is /home/s104062634/hw1
 - In your home directory/hw1, you **must** provide
 - A source code file named ‘**scanner.l**’.
 - A **makefile** for TAs to compile your code.
 - Expect just type make will compile source code into executable file
 - The makefile in which the name of the output executable file **must** be named ‘**scanner**’.
 - **Make sure your program work correctly in server environment**
- iLMS: Report (if you can’t finish your homework)
 - **Upload the report in PDF format to iLMS.**
 - e.g. file name is “100062801_report1.pdf”

How to Connect to our Server?

- PuTTY
 - <https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html>
- MobaXterm
 - <http://mobaxterm.mobatek.net/download.html>
- Terminal : for MAC OS/ Linux-based
 - `ssh -p 7878 your_account@140.114.88.201`

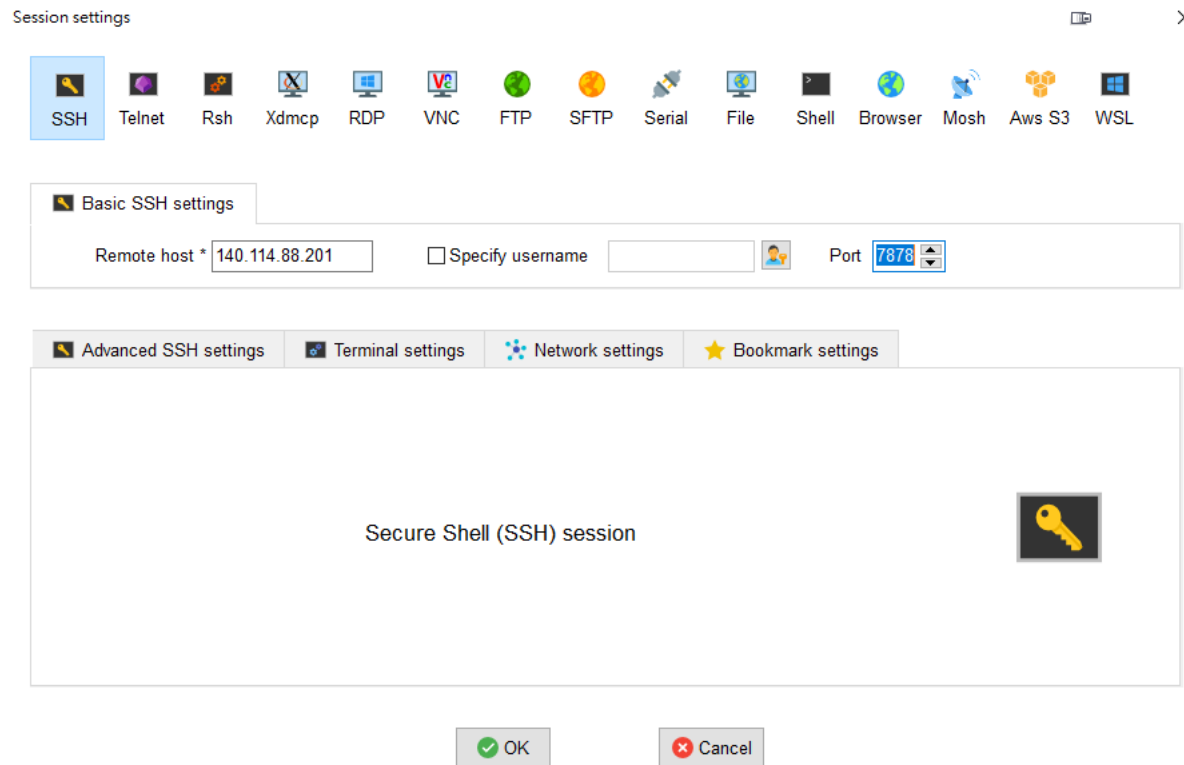
PuTTY

- SSH
- IP: 140.114.88.201 Port: 7878



MobaXTerm

- Toolbar > Session > SSH
- Remote host: 140.114.88.201 Port: 7878



Account

- **Account: s+student ID (lowercase letter)**
 - Example: s107123456
- **Password: same as account**
- Please change the password when you first login!
 - Steps:
 - Enter old password
 - Enter new password
 - Enter new password again
 - Connect to our server again
 - If you want to change your password afterwards,
 - \$ passwd

Tutorials

- **Linux command**
 - http://linux.vbird.org/linux_basic/0220filemanager.php
- **Vim**
 - http://linux.vbird.org/linux_basic/0310vi.php
- **Shell script**
 - http://linux.vbird.org/linux_basic/0320bash.php
- **Makefile**
 - <http://www.csie.nuk.edu.tw/.../Embedded%20Syst.../8-Makefile.pdf>
 - <http://www.cprogramming.com/tutorial/makefiles.html>
 - <http://jimmynuts.blogspot.tw/2010/12/gnu-makefile.html>

Reference

- **lex & yacc**
 - by John R. Levine, Tony Mason & Doug Brown
 - O' Reilly
 - ISBN: 1-56592-000-7
- **Mastering Regular Expressions**
 - by Jeffrey E.F. Friedl
 - O' Reilly
 - ISBN: 1-56592-257-3



Contact Us

- Facebook Group

- <https://www.facebook.com/groups/389309568282500/>
- Search “編譯器設計” , first item

- TAs

- 陳丕祐 piyochen@pllab.cs.nthu.edu.tw
- 陳泰良 tlchen@pllab.cs.nthu.edu.tw