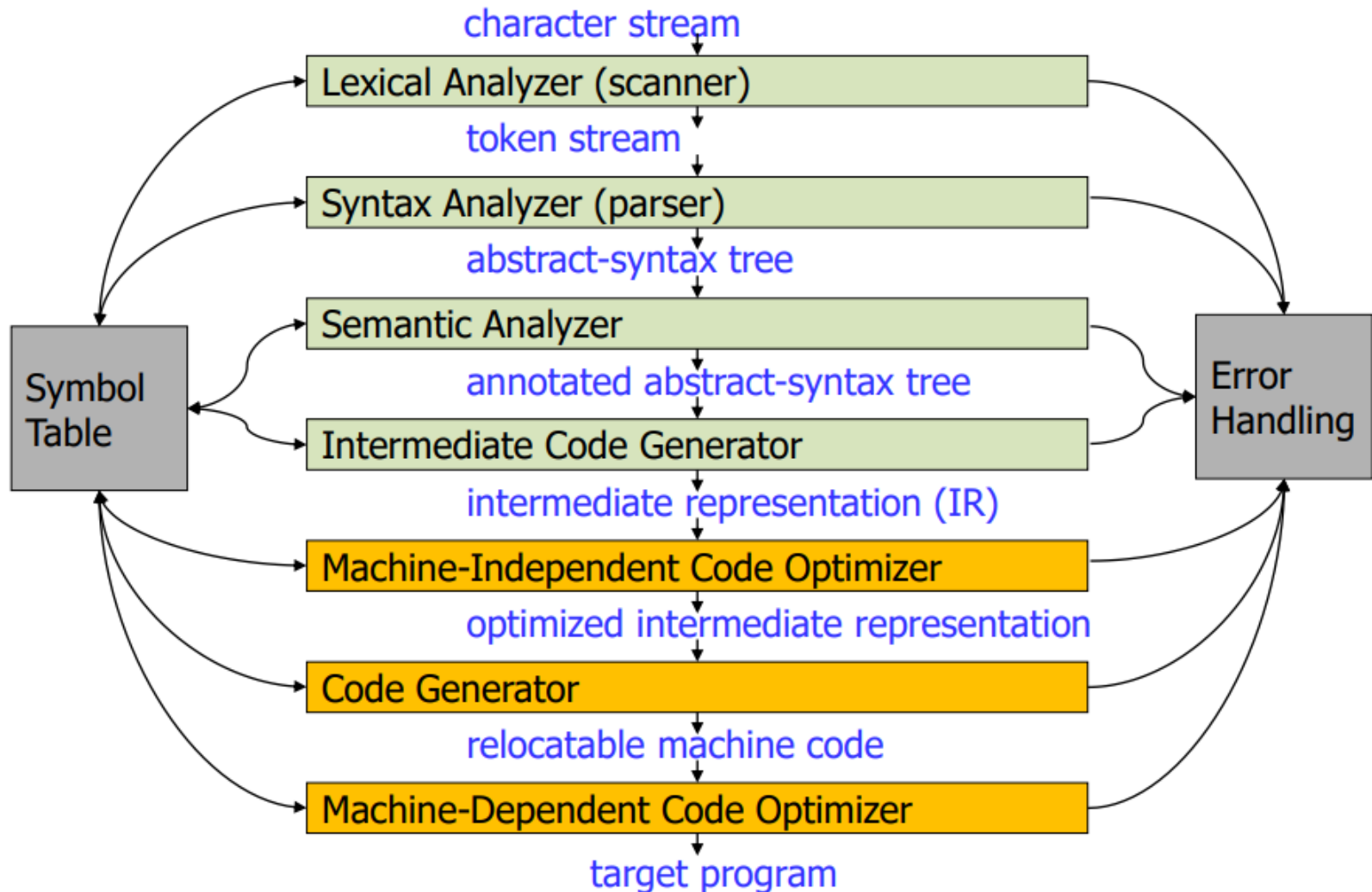




# YACC Tutorial

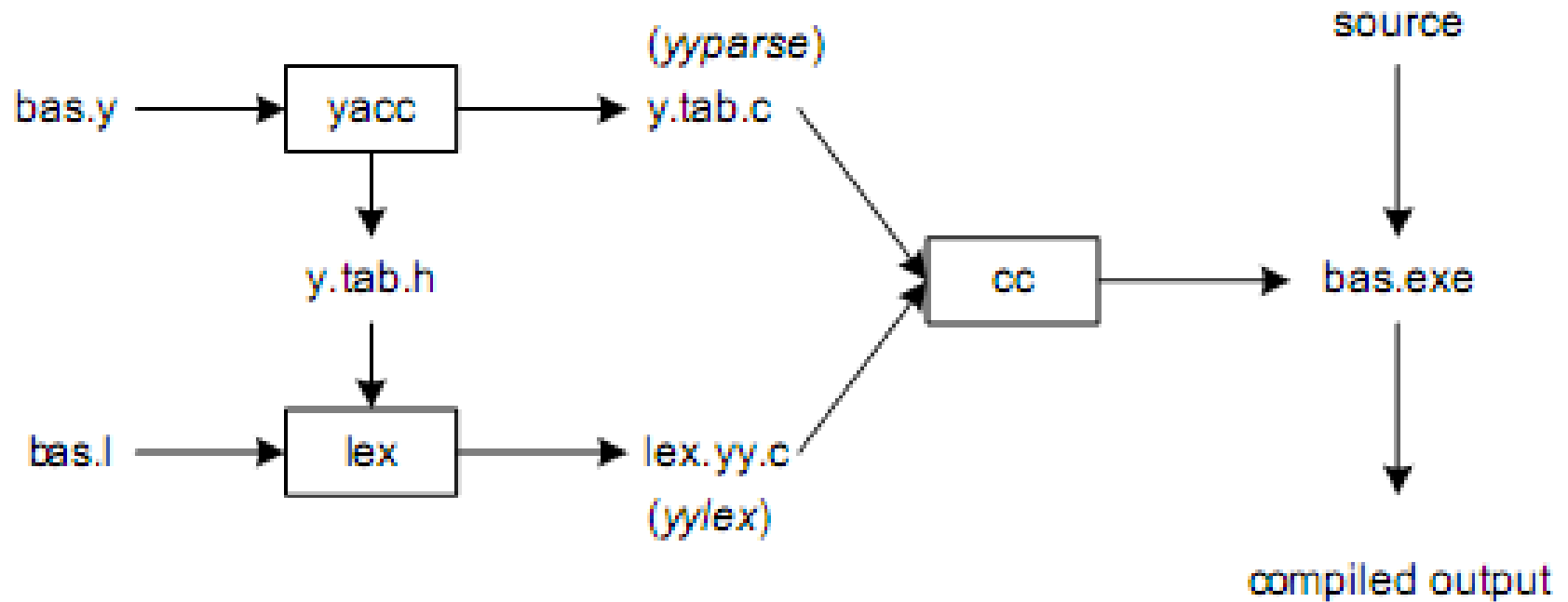
# The structure of a compiler



# YACC的工作

- YACC會把input當作 a sequence of tokens
  - 一個以上連續的token可以被表示成一個grammar(語法)
- YACC的目的是檢查語法是否合法
- Lex只是YACC的一個routine
  - 負責回傳token給YACC

# YACC的工作



# YACC如何表示語法

- 假設現在要做一個簡單的計算機的parser
- 設計語法，其中NUMBER為lex抓到的token:
  - $\text{expression} \rightarrow \text{NUMBER}$
  - $\text{expression} \rightarrow \text{expression} + \text{NUMBER}$
  - $\text{expression} \rightarrow \text{expression} - \text{NUMBER}$
- 同一個LHS可以合併在一起，再用| 隔開每個RHS
- 以上語法，在YACC會被表示成
  - $\text{expression} : \text{NUMBER}$   
 $\quad \quad \quad | \text{expression} + \text{NUMBER}$   
 $\quad \quad \quad | \text{expression} - \text{NUMBER}$

# YACC格式

- 總共分成三個部分

- definition

%%

grammars

%%

user code

- 每個部分以%%區隔開來

# 第一部分：Definition

```
% {  
#include <stdio.h>  
int yylex();  
double ans = 0;  
void yyerror(const char* message) {  
    printf("Invaild format\n");  
};  
% }
```

```
%union {  
    float  floatVal;  
    int    intVal;  
}  
%type <floatVal> NUMBER  
%type <floatVal> expression term  
                        factor group  
%token PLUS MINUS MUL DIV  
%token LP RP  
%token NUMBER NEWLINE
```

## 第二部分：Grammars

%%

lines : /\* empty (epsilon) \*/

| lines expression NEWLINE

{ printf("%lf\n", \$2); }

;

expression : term { \$\$ = \$1; }

| expression PLUS term { \$\$ = \$1 + \$3; }

| expression MINUS term { \$\$ = \$1 - \$3; }

;

term: factor { \$\$ = \$1; }

| term MUL factor { \$\$ = \$1 \* \$3; }

| term DIV factor { \$\$ = \$1 / \$3; }

;



## 第二部分：Grammars

```
factor: NUMBER { $$ = $1;}  
      | group { $$ = $1; }  
      ;  
group: LP expression RP { $$ = $2; }  
      ;  
%%
```

## 第二部分：Grammars

expression : | expression PLUS term { \$\$ = \$1 + \$3; }

\$\$

\$1

\$2

\$3

expression : | expression {...} PLUS term { \$\$ = \$1 + \$4; }

\$\$

\$1

\$2

\$3

\$4

# 第三部分：User Code

```
int main()
{
    yyparse();
    return 0;
}
```

# Lex 内容 -- Definition

```
% {  
#include "y.tab.h"  
#include <stdio.h>  
% }  
Digit [0-9]+  
%%
```

# Lex 內容 -- Rules

{Digit}	{ sscanf(yytext, "%f", &yy1val.floatVal); return NUMBER;}
\+	{return PLUS;}
\-	{return MINUS;}
\*	{return MUL;}
\/	{return DIV;}
\(	{return LP;}
\)	{return RP;}
\n	{return NEWLINE;}
.	{return yytext[0];}
%%	

# Lex 內容 – User Code

```
int yywrap(){  
    return 1;  
}
```

# 編譯流程

- 請先安裝flex和bison
  - `sudo apt-get install flex`
  - `sudo apt-get install bison`
- 編譯cau.y (產生 y.tab.c 及 y.tab.h )
  - `bison -y -d cau.y`
- 編譯cau.lex (產生 lex.yy.c)
  - `flex cau.l`
- 透過gcc產生可執行檔 (產生calc這個執行檔)
  - `gcc lex.yy.c y.tab.c -ly -lfl -o calc`
- 執行方式
  - `./calc < testfile`

# 執行結果

```
testfile x
1 5/2
2 9*3
3 ((3+5)*(8-2))
4 4+3*9-10*8
5
```

```
likems@DESKTOP-6BNKECN:~/yacc$ make
rm -f calc lex.yy.c y.tab.c y.tab.h
bison -y -d calc.y
flex calc.l
gcc lex.yy.c y.tab.c -ly -lfl -o calc
likems@DESKTOP-6BNKECN:~/yacc$ ./calc < testfile
2.500000
27.000000
48.000000
-49.000000
```



# Error Recovery

```

51 ~ term: term MUL {strcat(msg, " * ");} factor {
52     /* printf("6\n"); */
53     $$ = $1 * $4;
54 }
55 ~ | term DIV {strcat(msg, " / ");} factor {
56     /* printf("7\n"); */
57     $$ = $1 / $4;
58 }
59 ~ | factor {
60     /* printf("8\n"); */
61     $$ = $1;
62 }
63 ~ | error NUMBER { /* Error happened, discard token until it find NUMBER. */
64     /* printf("9\n"); */
65     yyerrok; /* Error recovery. */
66 }
67 ;

```

# Error Recovery

```
testfile_err x
1 3++9
2 5/2
3 9*3
4 3+5
5 4**6
6 5+***6+*6
7 4+3*9-10*8
8
```

```
likems@DESKTOP-6BNKECN:~/yacc_calc_err$ ./calc < testfile_err
syntax error at line 1
line 2: 5 / 2      (ans = 2.500000)
line 3: 9 * 3      (ans = 27.000000)
line 4: 3 + 5      (ans = 8.000000)
syntax error at line 5
syntax error at line 6
syntax error at line 6
line 7: 4 + 3 * 9 - 10 * 8      (ans = -49.000000)
likems@DESKTOP-6BNKECN:~/yacc_calc_err$
```

# 編譯流程

- 在example中，助教有幫大家寫好makefile，如下

```
all: clean y.tab.c lex.yy.c
```

```
    gcc lex.yy.c y.tab.c -ly -lfl -o calc
```

```
y.tab.c:
```

```
    bison -y -d cau.y
```

```
lex.yy.c:
```

```
    flex cau.l
```

```
clean:
```

```
    rm -f calc lex.yy.c y.tab.c y.tab.h
```

- 執行「make all」即可編譯產生「calc」

# 作業繳交注意事項

- **due: 5.22 11:59 p.m.**
- YACC的設計要比Lex要複雜很多，因此請馬上開始撰寫。
- 程式Demo環境是**Ubuntu 18.04**，因此請保證你們的程式碼能夠在Ubuntu上面編譯執行
- 作業說明有提供input file，可自行驗證
- 請準時繳交作業，作業遲交一天打七折
- 請把作業包成一個壓縮包，**上傳至網大，檔名命為「學號\_hw2」**，**學號輸錯，此項作業分數-10，沒輸學號分數-50**，請同學注意
- 作業繳交之後，在繳交截止隔周會安排時間，到EC5023找助教Demo。