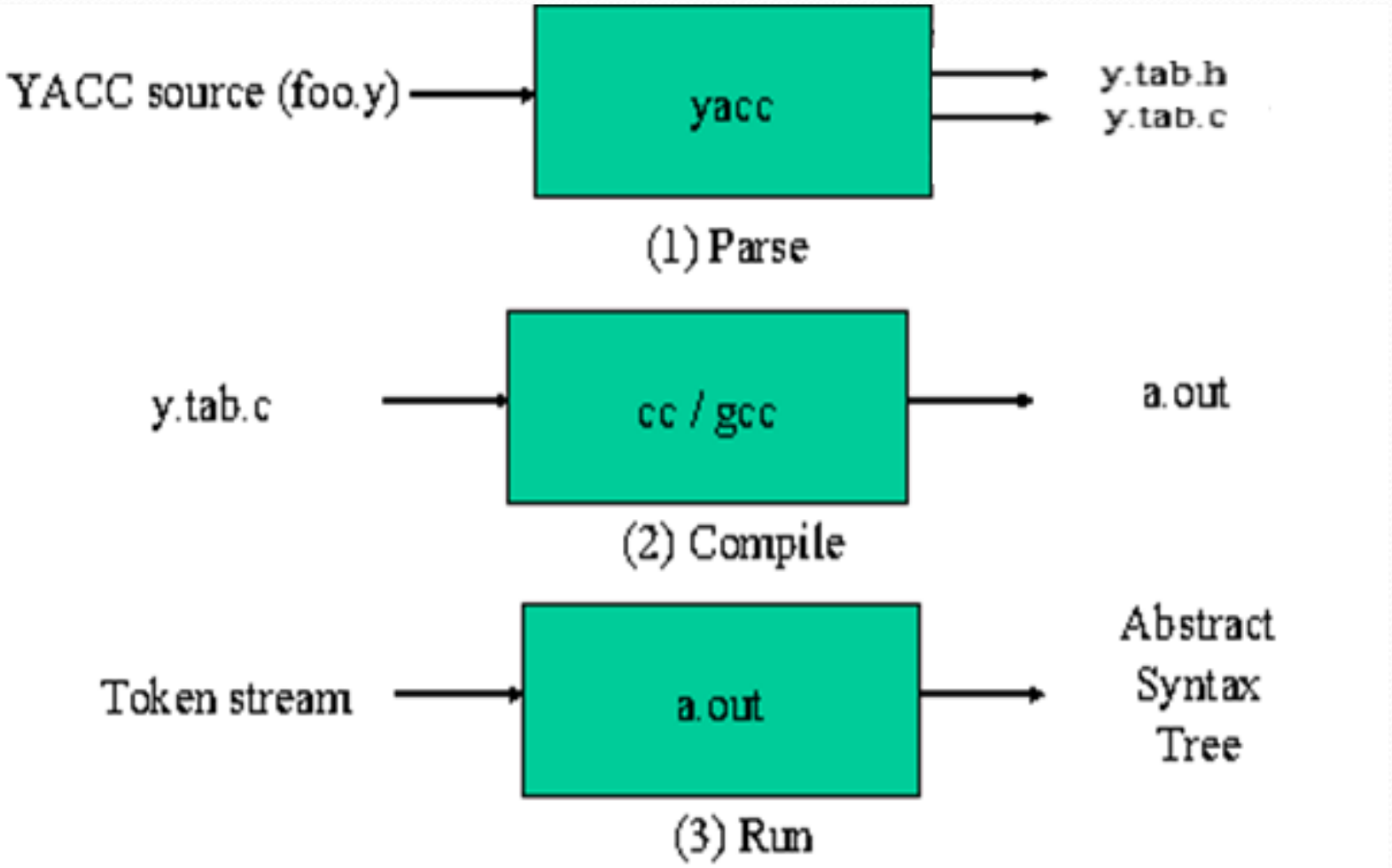
Bison/ YACC

Yacc

- **Yacc** stands for yet another compiler compiler.
- Designed to compile grammar given as input.
- Generate **LALR** parser to recognize sentences in the grammar.
- **Bison** is the updated version of **Yacc**.

How Yacc Works



How Yacc Works (Cont…)

- Designed for use with **C** code.
- Generates a parser written in **C**.
- Parser configured for use in conjunction with a **lex**-generated scanner.
- Relies on shared features (token types, **yylval**, etc.).
- Calls the function **yylex** as a scanner co-routine.
- Provide grammar specification file with **.y** extension.

How Yacc Works (Cont…)

- Invoke **Yacc** on the **.y** file.
- Creates the **y.tab.h** and **y.tab.c** files.
- Implements efficient **LALR** parser for grammar.
- Including code for the actions specified in .y file.
- File provides an extern function **yyparse().**
- **yyparse()** successfully parse a valid sentence.
- Compile that C file normally.
- Link with the rest of the code and have a parser.

Yacc File Format

- Input to Yacc is divided into three sections.

... definitions ...

%%

... production rules ...

%%

... subroutines...

Yacc File Format (Cont…)

- **The definitions section consists of:**
  - token declarations .
  - C code bracketed by “%{“ and “%}”.
- **The rules section consists of**:
  - BNF grammar .
- **The subroutines section consists of:**
  - user subroutines .

The Grammar

- **The grammar:**

<program> : <program> <expr>
| ε
;

< expr > : <expr> <+> <expr>
| <expr> < * > <expr>
| <ID>
;

*

The Grammar (Cont…)

- **Program and expr are nonterminals.**
- **Id is terminal** (tokens returned by lex)**.**
- **Expression may be :**
  - sum of two expressions .
  - product of two expressions .
  - or an identifier

Yacc Declaration

- **'%start':** Specifies the grammar's start symbol.
- **'%token':** Declare a terminal symbol with no precedence or associativity specified.
- **'%left':** Declare a terminal symbol that is left-associative .
- **'%right':** Declare a terminal symbol that is right-associative .
- **'%nonassoc':** Declare a terminal symbol that is nonassociative .

Shift and Reducing

- Perform Shift/reduce parsing
- Maintains set of states, reflecting one or more partially parsed rules
- After reading a token it may take two possible actions
  - **Shift:** If the token cannot complete any rule, shift the token in internal stack
  - **Reduce:** If a rule can be completed, then pop all R.H.S. symbol from the stack and push L.H.S. symbol

Lex File Example

```
%option noyywrap
%{
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "simplecalc.tab.h"
%}
WS [ \t]+
DIGIT [0-9]
NUMBER [-]?{DIGIT}+(\.{DIGIT}+)?
%%
```

Lex File Example (Cont…)

```
{NUMBER}  { sscanf(yytext,"%lf", &yylval);
                    return NUMBER; }
"+"   { return PLUS; }
"-"   { return MINUS; }
"/"   { return SLASH; }
"*"   { return ASTERISK; }
"("   { return LPAREN; }
")"   { return RPAREN; }
"\n"  { return NEWLINE; }
{WS} { /* No action and no return */}
```

Yacc File Example

```
%{
#include <stdio.h>
#include <stdlib.h>
#define YYSTYPE double          /* yyparse() stack type */
void yyerror(const char *s)       { printf("%s\n",s); }
int yylex(void);
%}
%token NEWLINE NUMBER LPAREN RPAREN
%left PLUS MINUS
%left ASTERISK SLASH
%%
```

Yacc File Example (Cont…)

```
input:        /* empty string */
| input line ;
line: NEWLINE
| expr NEWLINE                { printf("\t%.10g\n",$1); } ;
expr: expr PLUS expr          { $$ = $1 + $3; }
| expr MINUS expr             { $$ = $1 - $3; }
| expr ASTERISK expr          { $$ = $1 * $3; }
```

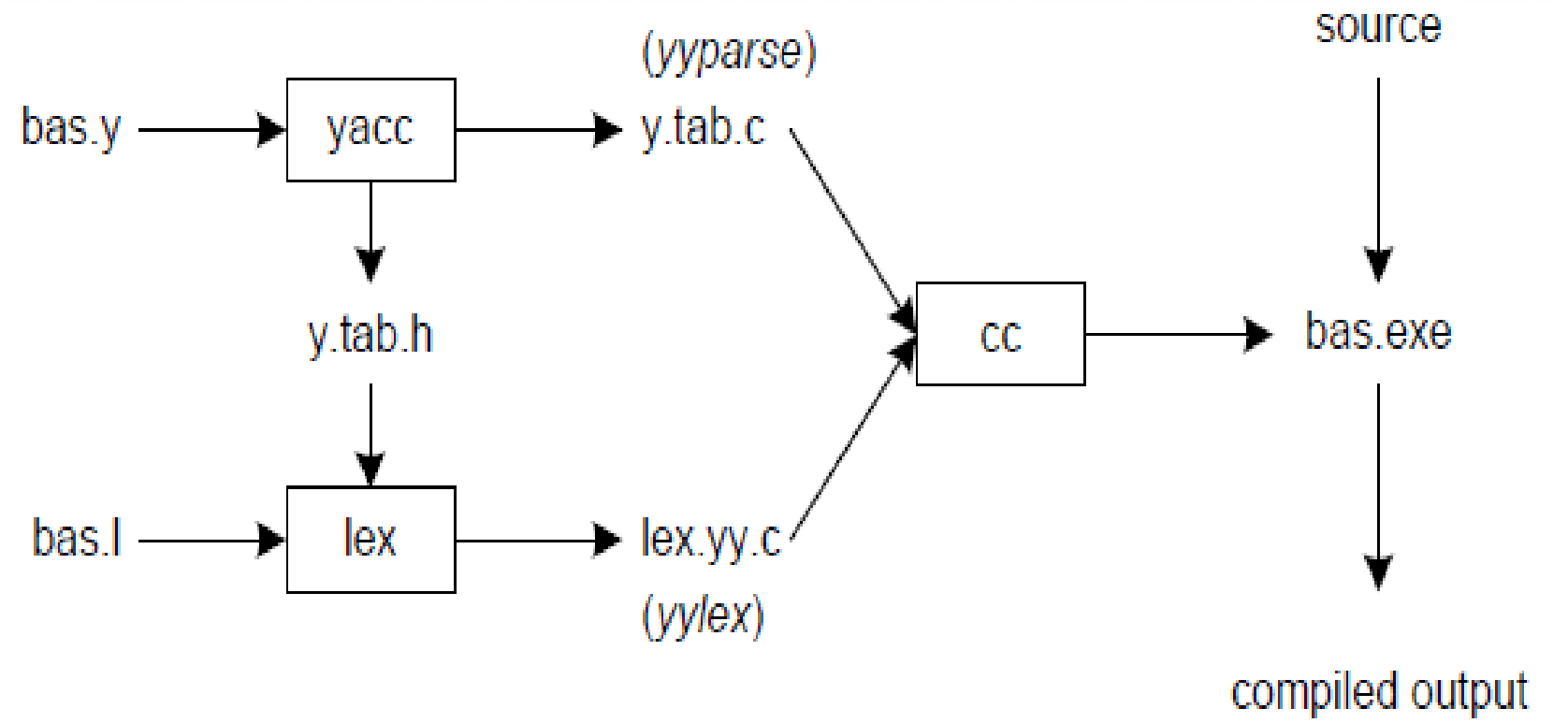
Yacc File Example (Cont…)

```
| expr SLASH expr  {
                        if($3 == 0)
                                {
                                        yyerror("Div error");
                                }
                         else
                                {
                                        $$ = $1 / $3;
                                 }
                        }
```

Yacc File Example (Cont…)

```
| MINUS expr                          { $$ = -$2; }
| factor ;
factor:  LPAREN expr RPAREN  { $$ = $2; }
| NUMBER ;
%%
main()
{
yyparse();
exit(0);
}
```

Linking Lex & Yacc



Command

- Compile Lex File to generate lex.yy.c file:

  flex simplecalc.l

- Compile YACC File to generate y.tab.c and y.tab.h file:

  bison -d simplecalc.y

- Use GCC Compiler to link lex.yy.c and y.tab.c files:

  gcc lex.yy.c y.tab.c -o simplecalc.exe

Offline Assignment

Design a Simple calculator that can support the following operations:

- "+" : plus operation: Example: 5 + 8 := 13
- "-" : minus operation: Example: 5 - 8 := -3
- "++" : increment: Example: 3 + 6++ := 10 ; 3 + ++6 := 10
- "--" : decrement: Example: 3 + 6-- := 8 ; 3 + --6 := 8
- "<<" : LEFTSHIFT operation: Example: 6 << 2 := 24
- ">>" : RIGHTSHIFT operation: Example: 6 >> 1 := 3

Offline Assignment (Cont…)

- "/" : DIV operation: Example: 6 / 1 := 6 [note that 6 / 0 := Infinity, you have to handle this.]
- "*" : MUL operation: Example: 6 * 2 := 12
- "%" : MOD operation: Example: 6 % 10 := 6 [only on integer operands, otherwise handle the error]
- "**" : DOUBLESTERIK operation: Example: 6 ** 2 := 36 [power: 6^2]

[Note: 1.You also have to handle UNARY MINUS operation!!!!!. Example: 33 - -2:= 35

2.You also have to handle UNARY PLUS operation!!!!!. Example: 33 - +2:= 31]

Some Resourceful Link

- Operator Precedence Link:

http://en.cppreference.com/w/c/language/operator_precedence

- Shift Reducing Simulation Video Link:

https://youtu.be/yTXCPGAD3SQ

- Bison Software Download Link:

http://gnuwin32.sourceforge.net/packages/bison.htm

Thank You