

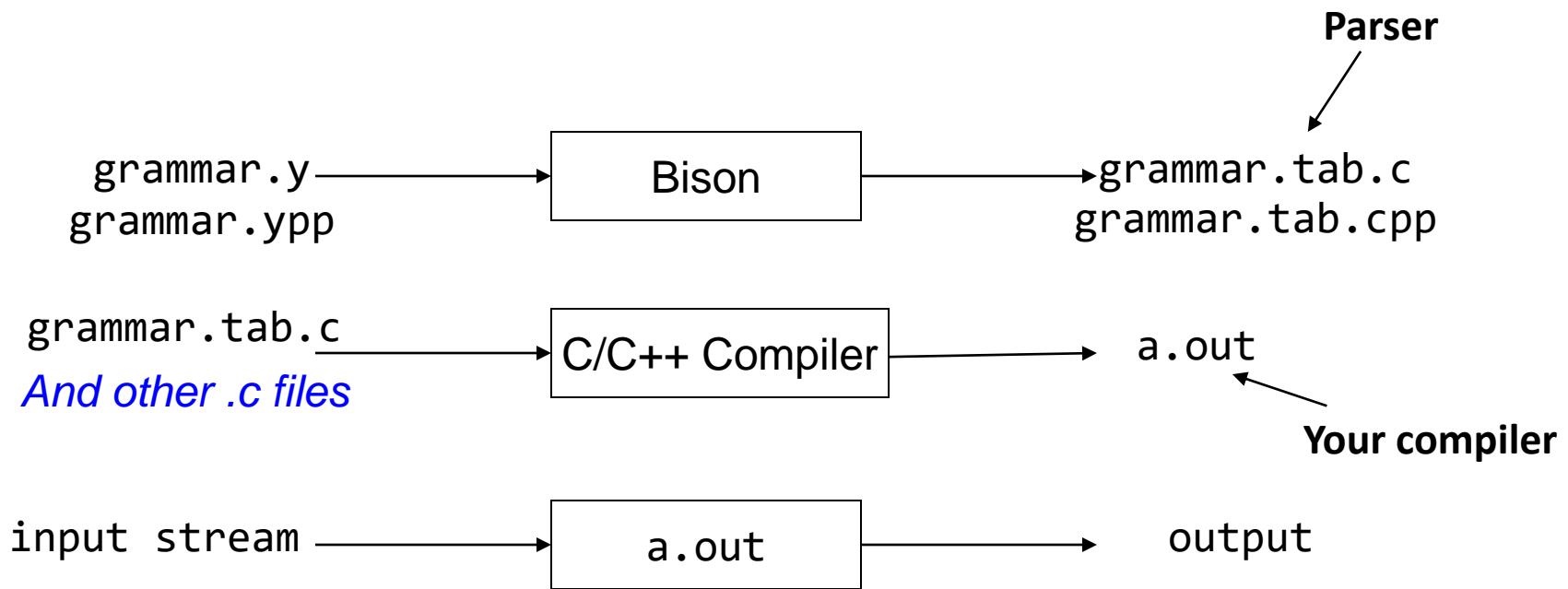
# Parsers - Bison

CS316 Spring 2021

# Bison (YACC)

- Specify the grammar
- Write a lexical analyzer to process input programs and pass the tokens to parser
- Call `yyparse()` from `main`
- Write error-handlers (what happens when the compiler encounters invalid programs?)

# Bison (YACC)



# Bison (YACC) – Input Format

```
%{  
Prologue  
%}  
Bison declarations  
%%  
Grammar rules  
%%  
Epilogue
```

# Bison (YACC) – Grammar Rules

```
%{  
Prologue  
%}  
Bison declarations  
%%  
E: E PLUS E {}  
| INTEGER_LITERAL {}  
;  
%%  
Epilogue
```

# Bison (YACC) - Prologue

```
%{  
Prologue  
%}  
%token PLUS INTEGER_LITERAL  
%left PLUS  
%%  
E: E PLUS E {}  
| INTEGER_LITERAL {}  
;  
%%  
Epilogue
```

# Bison (YACC) - Actions

```
%{  
Prologue  
%}  
%token PLUS INTEGER_LITERAL  
%left PLUS  
%%  
E: E PLUS E { $$ = $1 + $3; }  
| INTEGER_LITERAL { $$ = $1; }  
;  
%%  
Epilogue
```

A blue arrow points from the text "Legal C/C++ code" to the assignment statement "\$\$ = \$1 + \$3;" in the E production rule.

# Bison (YACC) – Semantic Values

```
%{  
Prologue  
%}  
%token PLUS INTEGER_LITERAL  
%left PLUS  
%%  
E: E PLUS E { $$ = $1 + $3; }  
| INTEGER_LITERAL { $$ = $1; }  
;  
%%  
Epilogue
```

The diagram illustrates the semantic value substitution process in a Bison rule. The rule E: E PLUS E { \$\$ = \$1 + \$3; } is shown. Three blue arrows originate from the tokens PLUS, E, and E respectively, and point to the semantic values \$1, \$2, and \$3. These semantic values are then used in the assignment statement \$\$ = \$1 + \$3;.

# Bison (YACC) – Helper Functions

```
%{  
int yylex();  
void yyerror(char *s);  
}  
%token PLUS INTEGER_LITERAL  
%left PLUS  
%%  
E: E PLUS E { $$ = $1 + $3; }  
| INTEGER_LITERAL { $$ = $1; }  
;  
%%  
Epilogue
```

# Bison (YACC) – Helper Functions

```
%{  
#include<stdlib.h>  
#include<stdio.h>  
int yylex();  
void yyerror(char const *s);  
%}  
%token PLUS INTEGER_LITERAL  
%left PLUS  
%%  
E: E PLUS E { $$ = $1 + $3; }  
| INTEGER_LITERAL { $$ = $1; };  
%%  
void yyerror(char const* s) {  
    fprintf(stderr,"%s\n",s);  
    exit(1);  
}
```

# Bison (YACC) – Integrating

- Recall that terminals are tokens
- Lexer produces tokens
  - How do the parser and lexer have a common understanding of tokens?
  - How should the Lexer return tokens?

```
//grammar.y file
...
%token PLUS INTEGER_LITERAL
%%
E: E PLUS E { $$ = $1 + $3; }
| INTEGER_LITERAL { $$ = $1; };
%%
...
```

```
bison -d grammar.y
```

```
grammar.tab.h
```

```
//scanner.l file
#include "grammar.tab.h"
extern YYSTYPE yylval
%%
\+      {return PLUS;}
[0-9]+   { yylval=atoi(yytext);
            return INTEGER_LITERAL;}
.\n      {}
%%
...
```

# Bison(YACC) - More..

- %union
- %define
- error