

# Bison (Objectives)

---

- Given an LALR(1) grammar, the student will be able to write a bison specification for the grammar.

# Bison

---

- Bison is a parser generator.
- It takes a restricted form of a context-free grammar,  $G$ , and produces C code that recognizes strings in  $L(G)$ .
- We will discuss how bison works in this class.

# Bison File Layout

---

```
%{  
    // C++ definitions  
}%  
// bison definitions like %token  
%%  
// grammar rules and actions  
%%  
// C++ support code
```

# Bison Definitions

---

- %token ...
  - define the tokens returned by yylex()
  - will be turned into #defines if '-d' option used

%token BEGIN

%token END

%token ID

%token NUM

# Bison Definitions

---

- %left, %right and %nonassoc
  - declare the associativity of tokens
  - lexical order in file gives precedence
  - all tokens on the same line have the same precedence

%left PLUS MINUS

%left TIMES DIVIDE

%right EXP

- PLUS and MINUS have lowest precedence, EXP has the highest

# Bison Definitions

---

➤ `%union { ... }`

- declare a union type for the attributes associated with terminals and non-terminals
- allows you to pass information up the parse tree as rules are reduced
- may have different type of information for different grammar symbols

```
%union {  
    int SymbolTableIndex;  
    char *AssemblyInstruction;  
}
```

# Bison Definitions

---

- `%start`
  - declare the start symbol of the grammar

```
%start Program
```
- `%type <type> symbol`
  - declare the type of the attribute for a grammar symbol

```
%type <SymbolTableIndex> ID
```

```
%type <AssemblyInstruction> Expression
```

# Bison Productions

---

## ➤ Form

```
lhs      : alt1    { //code for action }  
         | alt2    { //code for action }  
         ;
```

```
Expr    : Expr PLUS Expr      { // generate and ADD  
  inst}  
         | ID                  { // load an var in a reg }  
         ;
```

# Attributes

---

- Attributes allow you to pass information up the tree as reductions are performed.

```
Expr    : Expr PLUS Expr
         { $$ = generateAddInstruction($1,$3);}
        | ID
         { $$ = generateLoad($1);}
```

- The %type declaration for a symbol tells bison which item in the %union to use for a particular attribute
- Assign to yylval in yylex() for tokens to get initial attribute of terminals

# Example

---

```
%union { int Register;  
         int SymIndex}  
%token BEGIN END ID EQ  
%token NUM PLUS  
%type <SymIndex> ID NUM  
%type <Register> Expr  
%start Program  
  
%%  
Program : BEGIN Stmt END  
        ;
```

```
Stmt  : ID EQ Expr  
      { doStore($1,$3);} ;  
Expr  : Expr PLUS Expr  
      { $$ = doAdd  
        ($1,$3);}  
      | ID  
      { $$ = doLoad($1);} ;  
      | NUM  
      { $$ = doLoadI($1);}
```

# Example

---

- Parse the following program

```
BEGIN
```

```
  x = y + 3 + 5
```

```
END
```

# Errors

---

- define a function to emit an error message

```
void yyerror (char *msg)
{
    cerr << msg << endl;
}
```

- use error productions