Syntax-Directed Translation Example

- Using bison
- Creating an LL(1) parser using predictive recursive descent
- AST Representation

Bison Notes

- Call bison with flag –d to create header file containing token numbers (used in flex or other scanner implementation)
  - Since bison creates token numbers, bison should be called *before* flex
- Call bison with –v flag to dump output file showing resulting LR state machine and any shift-reduce, reduce-reduce errors

Simple Grammar

stmts $\rightarrow$ expr ; stmts

stmts $\rightarrow \varepsilon$

expr $\rightarrow$ IntConst

expr $\rightarrow$ (expr)

expr $\rightarrow$ expr + expr

expr $\rightarrow$ expr * expr

Bison Parser File

- Define tokens, precedence in first part of bison file (note, no EOF token):

```
%{
#include "parse_util.h"
%}

%token T_SEMI

%left T_PLUS
%left T_TIMES

%token T_LPAREN
%token T_RPAREN
%token T_INTCONST
%%
```

Defining AST Node Types

```
class GeneralNode {
 public:
  virtual void print
   (ostream& os = cout) {}
};
```

`#define YYSTYPE GeneralNode*`

```
class StmtsNode :
  public GeneralNode {
 private:
  StmtsNode() {}
 public:
  GeneralNode *stmt;
  GeneralNode *next;
  StmtsNode(GeneralNode *e,
   GeneralNode *nxt);
  void print (ostream& os =
   cout); };
```

```
class BinaryNode :
  public GeneralNode {
 private:
  BinaryNode() {}
 public:
  int theop;
  GeneralNode *leftarg;
  GeneralNode *rightarg;
  BinaryNode(int op, GeneralNode
   *arg1, GeneralNode *arg2);
  void print(ostream& os = cout);
};
```

Defining AST Node Types (cont)

```
class UnaryNode :
  public GeneralNode {
 private:
  UnaryNode() {}
 public:
  int theop;
  GeneralNode *arg;
  UnaryNode(int op,
    GeneralNode *thearg);
  void print
    (ostream& os = cout);
};
```

```
class IntNode :
  public GeneralNode {
 private:
  IntNode() {}
 public:
  int ivalue;
  IntNode(int ival);
  void print(ostream& os = cout);
};
```

`extern GeneralNode *parse_root;`

Corresponding Scanner File

- Define tokens, plus return appropriate AST nodes when needed:

Initialize and include header files

```
\+       { char_num++; return T_PLUS; }
\*       { char_num++; return T_TIMES; }
\(       { char_num++; return T_LPAREN; }
\)       { char_num++; return T_RPAREN; }
\;       { char_num++; return T_SEMI; }
{DIGIT}+ { char_num += strlen(yytext);
           yylval = new IntNode(atoi(yytext));
           return T_INTCONST; }
```

Handle white space and errors

Remainder of Parser File

- Create actions for building AST:

```
%%
stmts: e T_SEMI stmts {
  $$ = new StmtsNode($1,$3); parse_root = $$; }
        | { $$ = 0; parse_root = $$; }
;
e: e T_PLUS e {
     $$ = new BinaryNode(T_PLUS,$1,$3); }
        | e T_TIMES e {
            $$ = new BinaryNode(T_TIMES,$1,$3); }
        | T_INTCONST { $$ = $1; }
        | T_LPAREN e T_RPAREN {
            $$ = new UnaryNode(T_LPAREN,$2); }
;
%%
```

Calling the bison Parser

- Open file to be read and then call yyparse:

```
GeneralNode *parse_root = 0;
GeneralNode *do_parse(const char* filename) {
  if ((yyin = fopen(filename,"r")) == NULL) {
    cout << "Error!  Unable to open file "
         << filename << endl;
    return 0;
  }
  if (yyparse())
    return 0;
  else
    return parse_root;
}
```

Creating LL Parser

- Change grammar (keep track of actions as needed):

	Use if next token is
stmts $\rightarrow$ expr ; stmts	IntConst (
stmts $\rightarrow \varepsilon$	$
expr $\rightarrow$ t e’	IntConst (
e’ $\rightarrow \varepsilon$	;)
e’ $\rightarrow$ + t e’	+
t $\rightarrow$ f t’	IntConst (
t’ $\rightarrow \varepsilon$	;) +
t’ $\rightarrow$ * f t’	*
f $\rightarrow$ (expr)	(
f $\rightarrow$ IntConst	IntConst

Match Function

- Match next token or throw exception:

```
void match (int token_num) {
  if (token_num == nexttok)
    nexttok = yylex();
  else
    throw tokentostring(token_num);
}
```

Recursive Functions for Nonterminals

```
GeneralNode *do_Stmts () {
  if ((nexttok == T_INTCONST) ||
      (nexttok == T_LPAREN)) {
    GeneralNode *first = do_E();
    match(T_SEMI);
    GeneralNode *rst= do_Stmts();
    return new
      StmtsNode(first,rst);
  }
  else if (nexttok == T_EOF)
    return 0;
  else
    throw "integer constant, left
   parenthesis (() or end of
   file";
}
```

```
GeneralNode *do_E () {
  GeneralNode *left = do_T();
  return do_EPrime(left);
}
GeneralNode *do_Eprime
                (GeneralNode *left) {
 if ((nexttok == T_SEMI) ||
     (nexttok == T_RPAREN))
  return left;
 else if (nexttok == T_PLUS) {
  match(T_PLUS);
  GeneralNode *right = do_T();
  return do_EPrime(new
    BinaryNode(T_PLUS,left,right));
 }
 else
  throw "semi-colon (;) or plus (+)";
}
```

Recursive Functions for Nonterminals

```
GeneralNode *do_T () {
  GeneralNode *left = do_F();
  return do_TPrime(left);
}
GeneralNode *do_Tprime
                (GeneralNode *left) {
  if ((next_token == T_SEMI) ||
      (next_token == T_PLUS) ||
      (next_token == T_RPAREN))
    return left;
  else if (next_token == T_TIMES) {
    match(T_TIMES);
    GeneralNode *right = do_F();
    return do_TPrime(new
      BinaryNode(T_TIMES,left,right));
  }
  else
    throw "semi-colon (;), plus (+),
   or times (*)"; }
```

```
GeneralNode *do_F () {
 if (next_token == T_INTCONST) {
   GeneralNode *result = yylval;
   match(T_INTCONST);
   return result;
 }
 else if (next_token == T_LPAREN){
   match(T_LPAREN);
   GeneralNode *expr = do_E();
   match(T_RPAREN);
   return new
     UnaryNode(T_LPAREN,expr);
 }
 else
   throw "integer constant or left
   parenthesis (()";
}
```

Creating yyparse

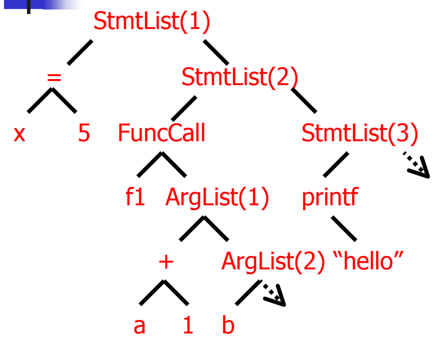
- LL(1) version of yyparse:

```
int yyparse () {
  next_token = yylex();
  try {
    parse_root = do_Stmts();
  }
  catch (const char *string) {
    cout << "Error: expected " << string << ", found "
         << token_num_to_string(nexttok) <<
         " at line number " << line_num <<
         " character " << char_num << endl;
    return 1;
  }
  return 0;
}
```

AST Implementation

- In Object-Oriented languages, generally implement AST as general node class and specializations
- Many later processes involve traversals of the resulting tree
- Traversals built using recursive methods defined for each node
- Example: printing resulting code using print methods
  - Other examples:
    - Type checking - type emerges as result of function call
    - Interpreters - interpret and evaluate operations
    - Intermediate code generation

Traversal Example



```
root(StmtList(1))->print
  stmt(=)->print
    ident(x)->print
      cout << string for x
    cout << "="
    expr(5)->print
      cout << string for 5
    cout << ";"
  rest(StmtList(2)->print
    stmt(FuncCall)->print
      fname(f1)->print
        cout << string for f1
      arguments(ArgList(1)->print
…
```