Compiler Lab

Part #2

12.01.2015

Building an expression tree

Semantic actions to build an expression tree

E --> E1 + E2 { E.ptr = *mkNode* (+, E1.ptr, E2.ptr) }

E --> NUM { E.ptr = *mkLeafNode* (NUM.lexval) }

YACC code

```
pgm  : ID '=' expr '\n' {printf("%d\n", evaluate($1));}
     ;
expr :   expr '+' expr {$$=mkNode('+', $1, $3);}
     ;
expr :   NUMBER {$$=mkLeafNode($1);}
```

Expression Tree

```
%union{       //defines YYSTYPE
    int   ival;
    struct tree_node *nptr;
};

%token    <ival> NUMBER
%type     <nptr> expr
```

```
%{
#include <stdio.h>
#include "exprtree.h"
   void yyerror(char *);
%}

%union{
       int ival;
       struct tree_node *nptr;
};

%token  <ival> NUMBER
%type    <nptr> expr

%%
pgm: ID '=' expr '\n' {printf("%d\n", evaluate($1));}
       ;
expr:      expr '+' expr {$$=mkOperatorNode('+', $1, $3);}
       ;
expr:      NUMBER {$$=mkLeafNode($1);};
%%
```

SIL: Introduction

A sample Program

```
decl
integer   x,   y1;
enddecl
x = 10;
y1 =  2;
write(x+y1);
```

CFG

pgm -> gdecl stmtlist

gdecl -> DECL decllist ENDDECL

stmtlist -> stmt stmtlist | ϵ

CFG

declist -> decl declist | decl

decl -> INTEGER idlist ;

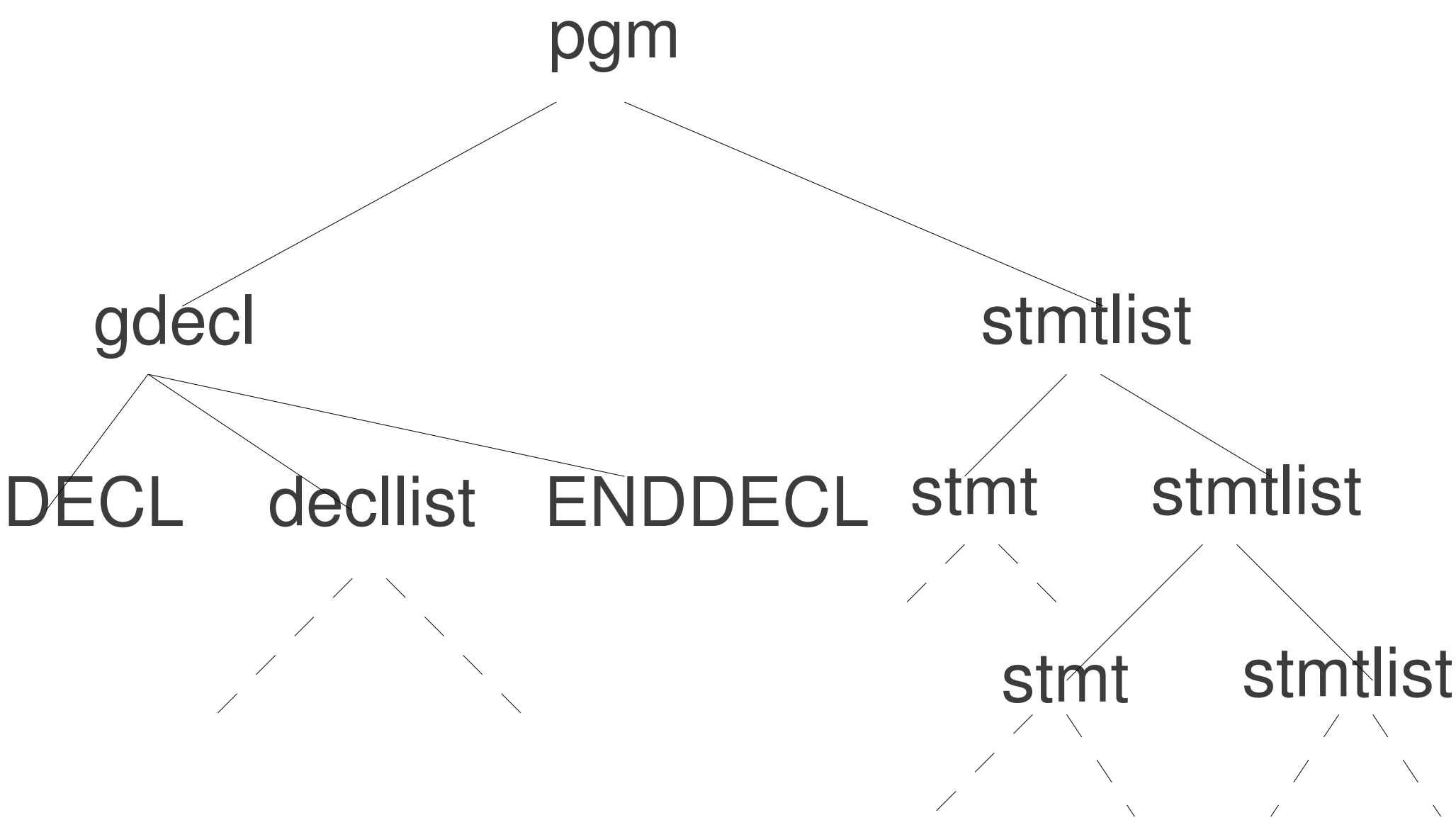
idlist -> ID ',' idlist | ID

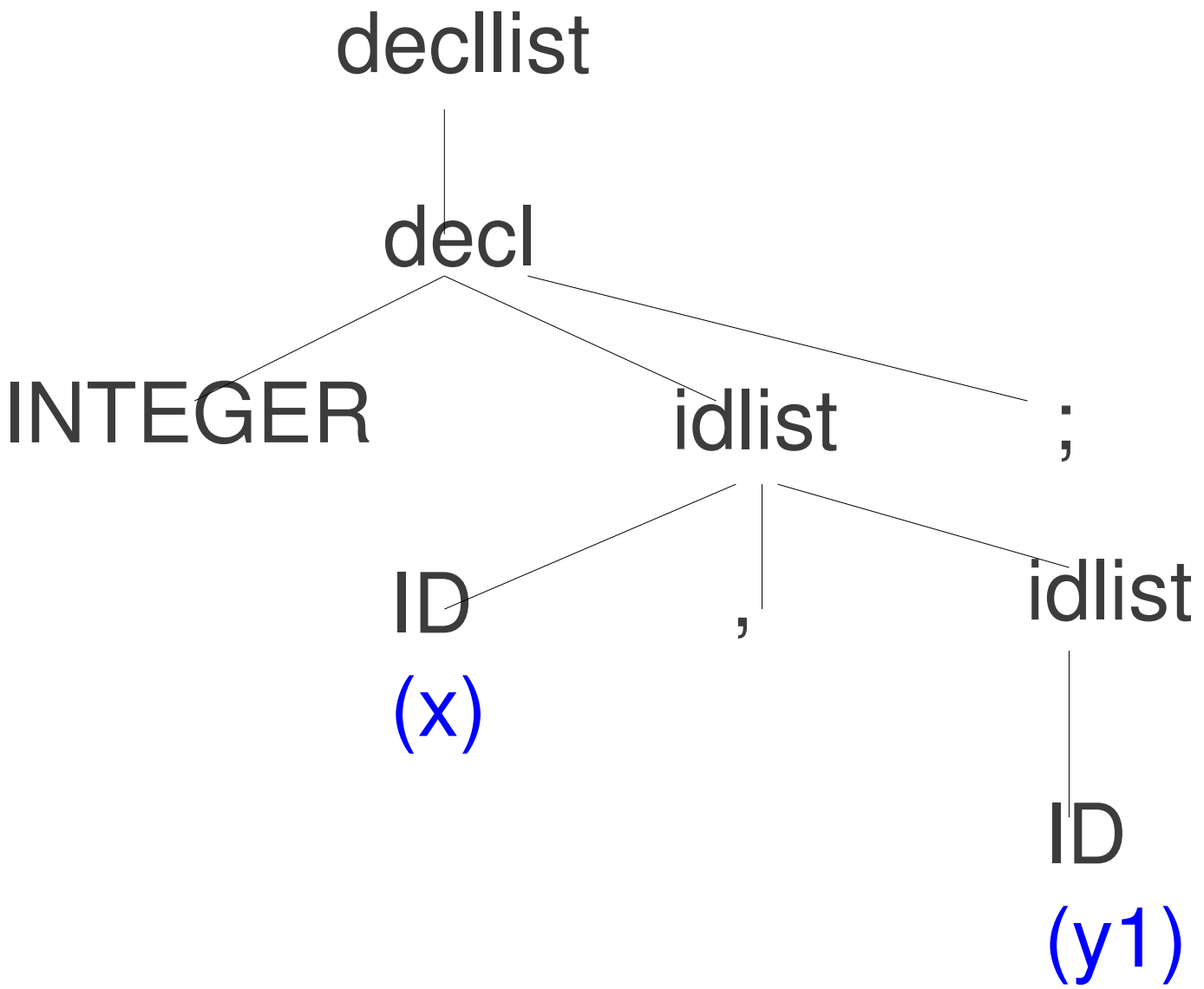
CFG

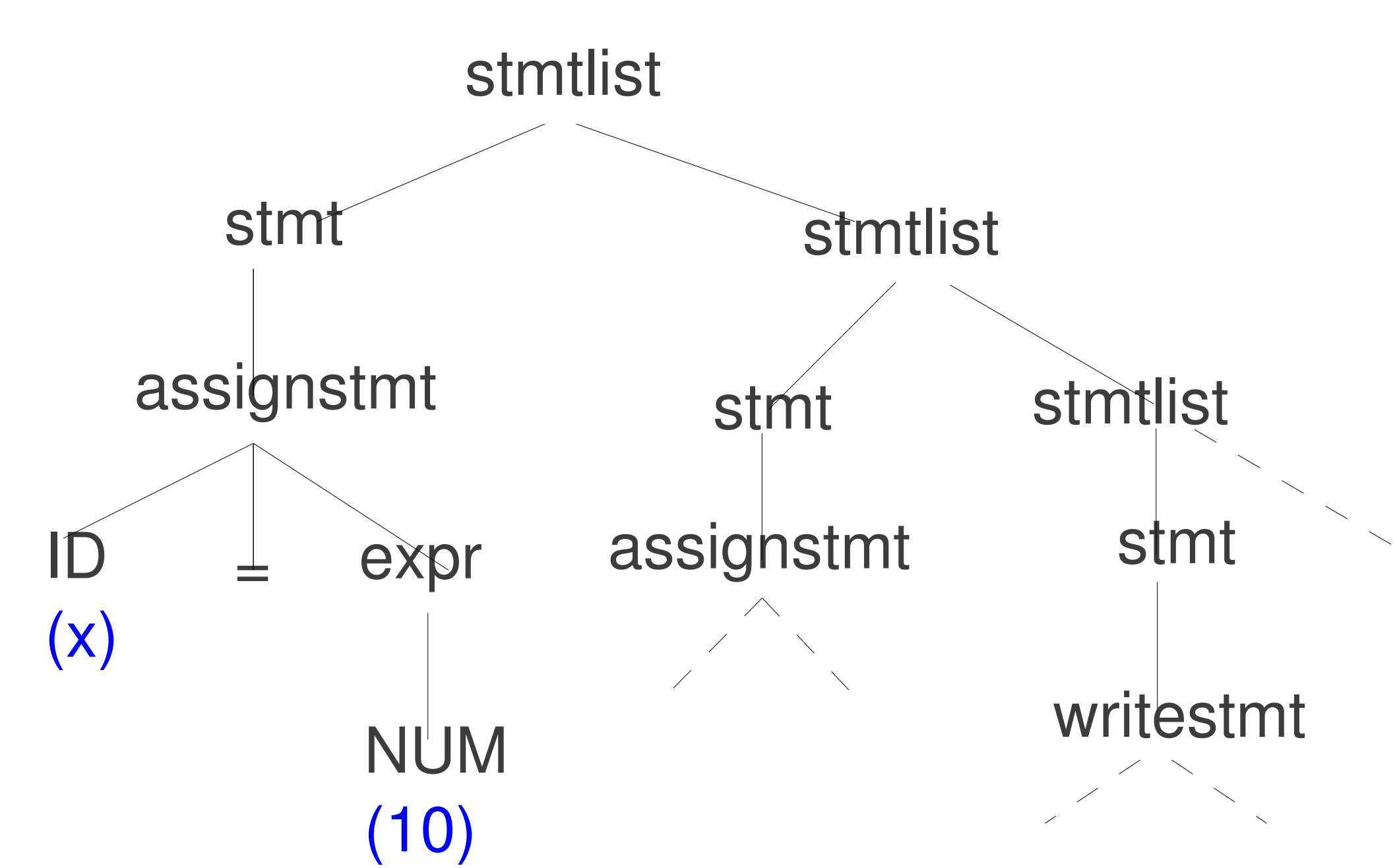
stmt -> assignstmt | writestmt

assignstmt -> ID '=' expr ';'

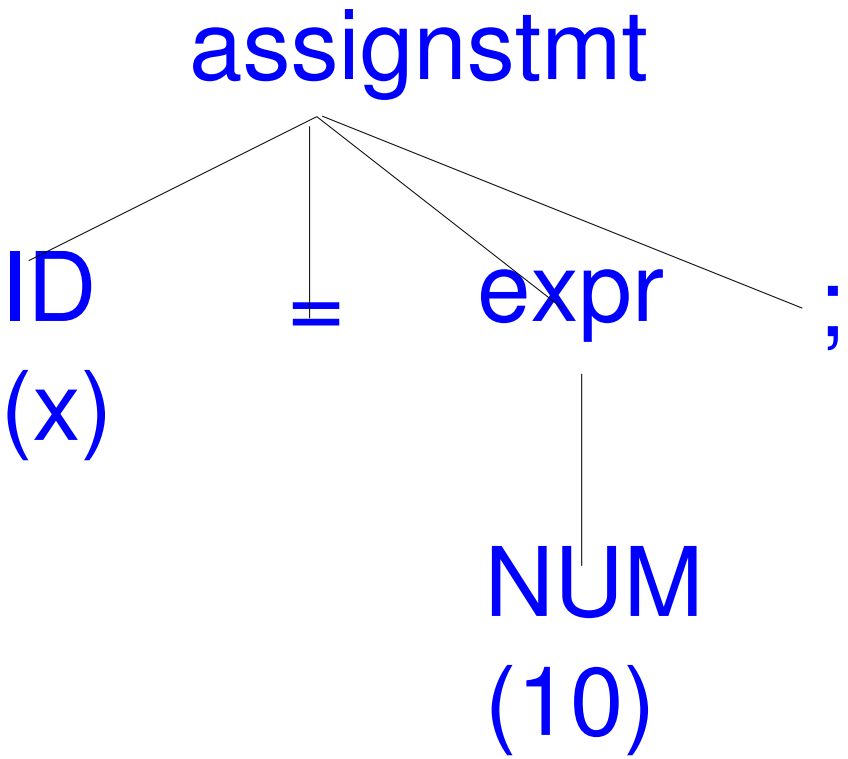
writestmt -> WRITE '(' expr ')' ';'


pgm
gdecl
stmtlist
DECL
decllist
ENDDECL
stmt
stmtlist
stmt
stmtlist

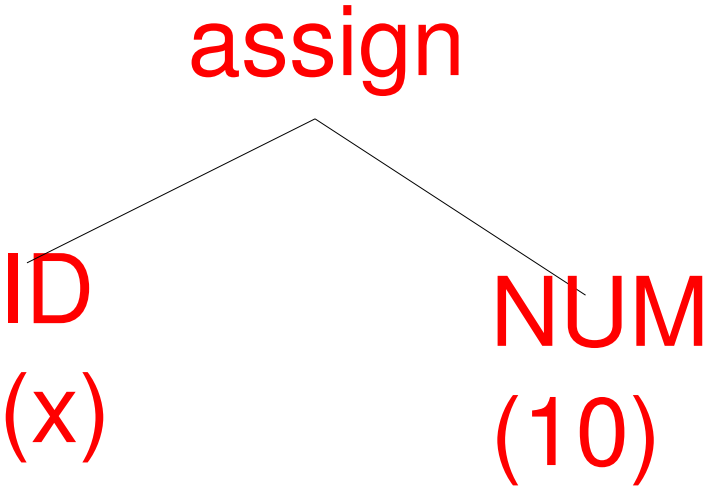

decllist
decl
INTEGER
idlist
;
ID
(x)
,
idlist
ID
(y1)


stmtlist
stmt
stmtlist
assignstmt
stmt
stmtlist
ID
(x)
=
expr
assignstmt
stmt
NUM
(10)
writestmt

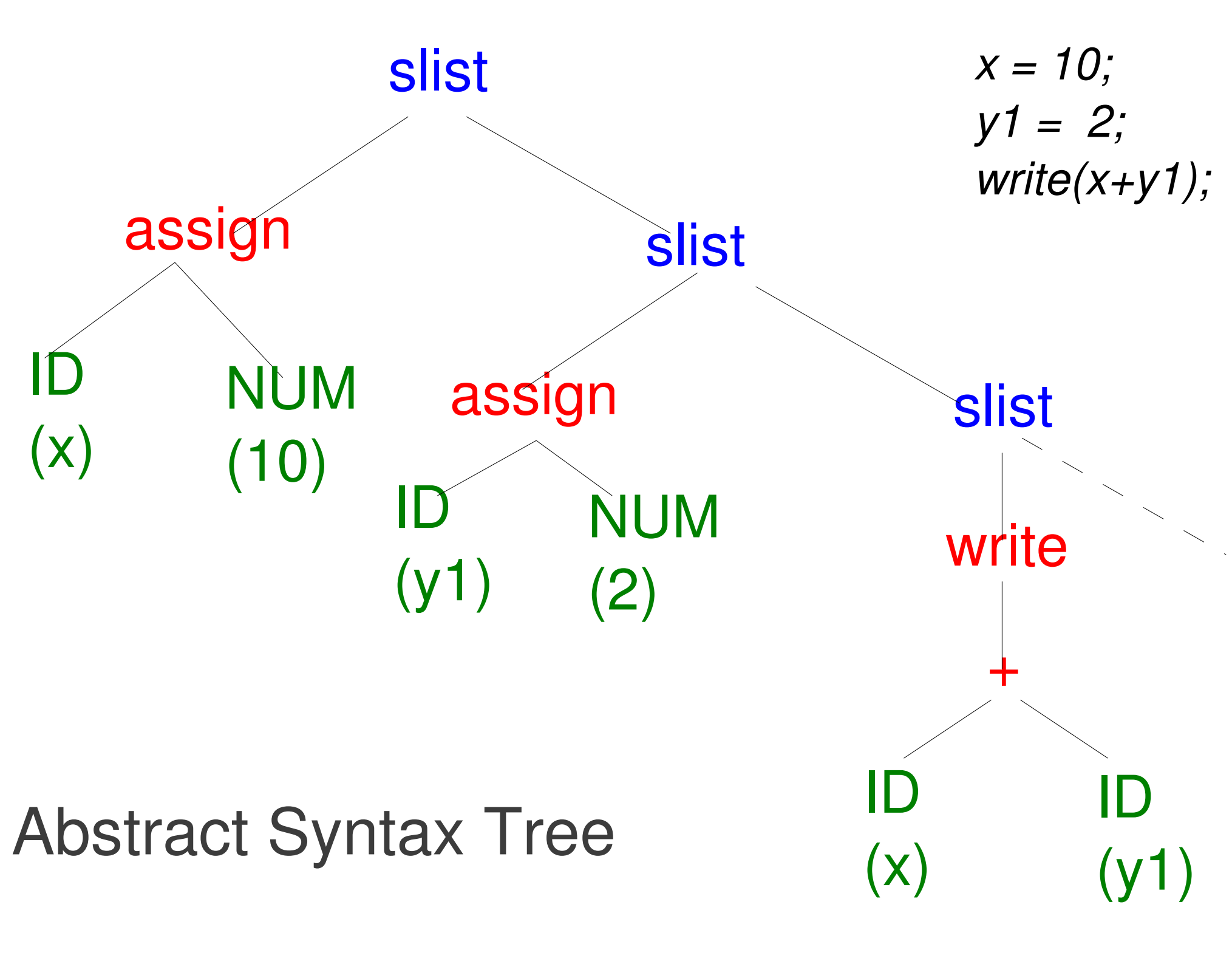
Parse Tree



Abstract Syntax Tree



Statement: x=10



Abstract Syntax Tree

Symbol Table

Symbols : x, y1
Attributes: name, type, scope ....

- A data structure to store these attributes
- Use a linked list

Symbol table entry : fields

```
struct Gsymbol {
        char *NAME; // Name of the Identifier
         int TYPE; // TYPE can be INTEGER or BOOLEAN
         int SIZE; // Size field for arrays
        int BINDING; // Address of the Identifier in Memory
        .............
        struct Gsymbol *NEXT; // Pointer to next Symbol Table Entry */
}
```

Use the binding field to store the value (for interpretation)

Symbol Table entry

- One entry per symbol
- New entry created upon processing a declaration
- Set attributes
  - Name, Type
- Binding field can be used for storing its value
- x=10
  - enter value 10 in the binding field of x

Separate files

- For AST,

  tree.h - declarations alone

  tree.c - definition of functions

- For Symbol Table,

  symtable.h,

  symtable.c

#include "tree.h"

#include "symtable.h"

To Compile:

cc lex.yy.c   y.tab.c   tree.c   symtable.c -ll

Input from file

```
int main(int argc, char *argv[ ])
{
    yyin=fopen(argv[1],"r");
    yyparse();
    fclose (yyin);
}
```