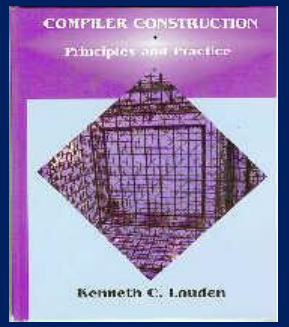
CS 445: Assignment 2

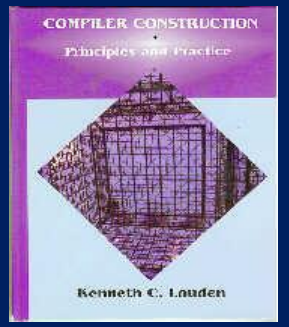
Bison – Your parser.y file

Assignment 2: Abstract Syntax Tree



- Sibling links and up to 3 children.
- Basically, you can imagine a node that has multiple children and one sibling.
- Sibling: Collections of things that come in a list:
  - Parameters
  - Multiple declarations: int x,y,z
  - Multiple statements:
    - Char yy;
    - X=3;
    - Y = ;

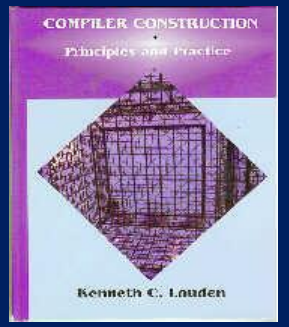
Recommended workflow:



- Download the files from the shared folder.
- In the top %{ %} section of parser.y add:

```
int numErrors;
int numWarnings;
extern int line;
extern int yylex();

TreeNode *addSibling(TreeNode *t, TreeNode *s)
{
   // make sure s is not null. If it is this s a major error. Exit the program!
   // Make sure t is not null. If it is, just return s
   // look down t's sibling list until you fin with with sibblin = null (the end o f the lsit) and add s there.
   return s;
}
// pass the static and type attribute down the sibling list
void setType(TreeNode *t, ExpType type, bool isStatic)
{
   while (t) {
      // set t->type and  t->isStatic
      // t = t->sibling;
   }
}
// the syntax tree goes here
TreeNode *syntaxTree;
```

Recommended workflow:



- Update %union to include:

```
%union {
   TokenData *tokenData;
   TreeNode *tree;
   ExpType type;   // for passing type spec up the tree
}
```

- Bring in the productions from the bC grammar for this term.
  - For each one you will have to add a type:
    %type <tree> program
  - The one exception:
    %type <type> typeSpec
  - The tokenData should remain unchanged except you need to add the operators:

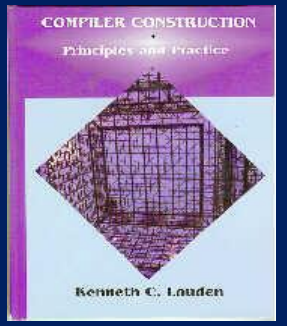
%token <tokenData> '(' ')' ',' ';' '[' '{' '}' ']' ':'

  - Keep them in the order <tree> <type> <tokenData> and add to the beginning/end of the sections:

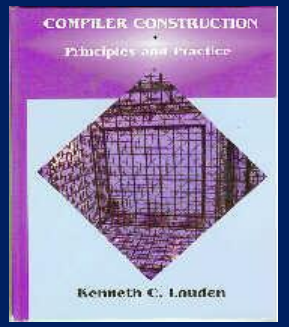
```
%token <tokenData> FIRSTOP
%token <tokenData> LASTOP
%token <tokenData> LASTTERM
```

```
char *largerTokens[LASTTERM+1];       // used in the utils.cpp file printing routines

// create a mapping from token class enum to a printable name in a
// way that makes it easy to keep the mapping straight.
void initTokenStrings()
{
   largerTokens[ADDASS] = (char *)"+=";
   largerTokens[AND] = (char *)"and";
   largerTokens[BOOL] = (char *)"bool";
   largerTokens[BOOLCONST] = (char *)"boolconst";
   largerTokens[BREAK] = (char *)"break";
   largerTokens[BY] = (char *)"by";
   largerTokens[CHAR] = (char *)"char";
   largerTokens[CHARCONST] = (char *)"charconst";
   largerTokens[CHSIGN] = (char *)"chsign";
   largerTokens[DEC] = (char *)"--";
   largerTokens[DIVASS] = (char *)"/=";
   largerTokens[DO] = (char *)"do";
   largerTokens[ELSE] = (char *)"else";
   largerTokens[EQ] = (char *)"==";
   largerTokens[FOR] = (char *)"for";
   largerTokens[GEQ] = (char *)">=";
   largerTokens[ID] = (char *)"id";
   largerTokens[IF] = (char *)"if";
   largerTokens[INC] = (char *)"++";
   largerTokens[INT] = (char *)"int";
   largerTokens[LEQ] = (char *)"<=";
   largerTokens[MAX] = (char *)":>:";
   largerTokens[MIN] = (char *)":<:";
   largerTokens[MULASS] = (char *)"*=";
   largerTokens[NEQ] = (char *)"!=";
   largerTokens[NOT] = (char *)"not";
   largerTokens[NUMCONST] = (char *)"numconst";
   largerTokens[OR] = (char *)"or";
   largerTokens[RETURN] = (char *)"return";
   largerTokens[SIZEOF] = (char *)"sizeof";
   largerTokens[STATIC] = (char *)"static";
   largerTokens[STRINGCONST] = (char *)"stringconst";
   largerTokens[SUBASS] = (char *)"-=";
   largerTokens[THEN] = (char *)"then";
   largerTokens[TO] = (char *)"to";
   largerTokens[WHILE] = (char *)"while";
   largerTokens[LASTTERM] = (char *)"lastterm";
}
```

Recommended workflow: Add an array to your .y file

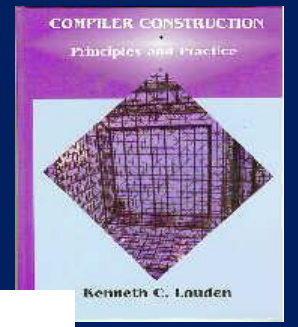


Recommended workflow:



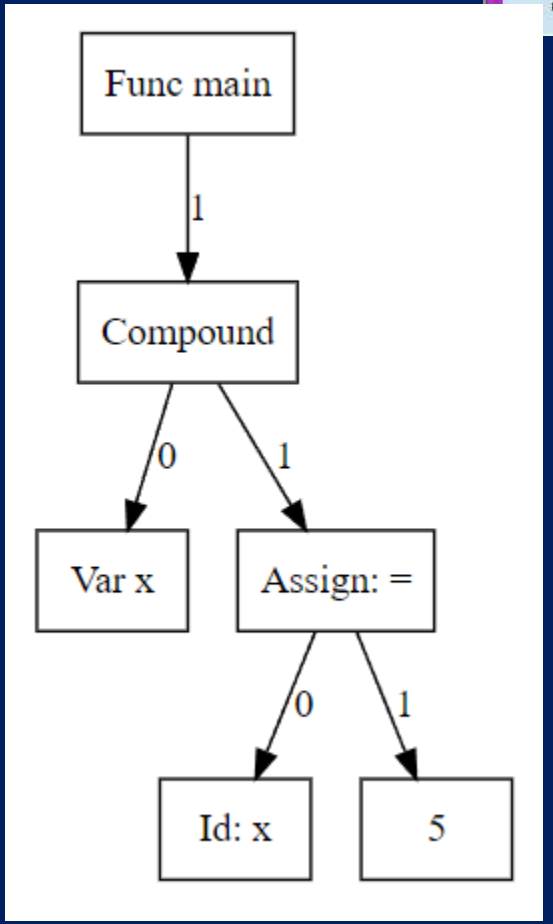
- Start each production with empty { } Fill them in after the next step.
- Write the functions in treeUtils.cpp then start filling in the {}

A simple bC file

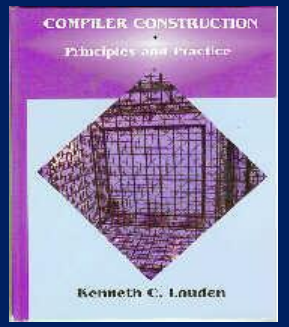



```
#DRBC This program is as simple as I can get.
#DRBC This should compile without errors
main ()
{
  int x;
  x = 5;
}
```

```
Func: main returns type void [line: 3]
.   Child: 1  Compound [line: 4]
.   .   Child: 0  Var: x of type int [line: 5]
.   .   Child: 1  Assign: = [line: 6]
.   .   .   Child: 0  Id: x [line: 6]
.   .   .   Child: 1  Const 5 [line: 6]
```



bC




```
program     : precomList declList                  { syntaxTree = $2;}
            ;

precomList  : precomList PRECOMPILER       { $$ = $1;}
            | PRECOMPILER                   { printf("%s\n", yylval.tokenData->tokenstr);}
            | /* empty */                   { $$ = NULL;}
            ;
```

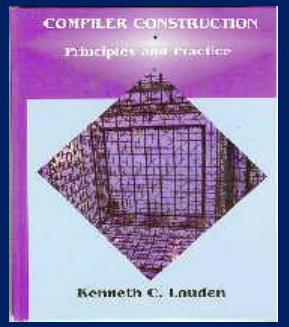
Tiny

$$ $1 $3

```
stmt_seq    : stmt_seq SEMI stmt
              { YYSTYPE t = $1;
                if (t != NULL)
                { while (t->sibling !=
NULL)
                     t = t->sibling;
                  t->sibling = $3;
                  $$ = $1; }
                  else $$ = $3;
              }
            | stmt  { $$ = $1; }
            ;
```

bC

Lists contain siblings




```
declList    : declList decl   { $$ = addSibling( $1, $2); }
            | decl       { $$ = $1;}
            ;

decl        : varDecl              { $$ = $1;}
            | funDecl   { $$ = $1;}
            ;
```

Tiny

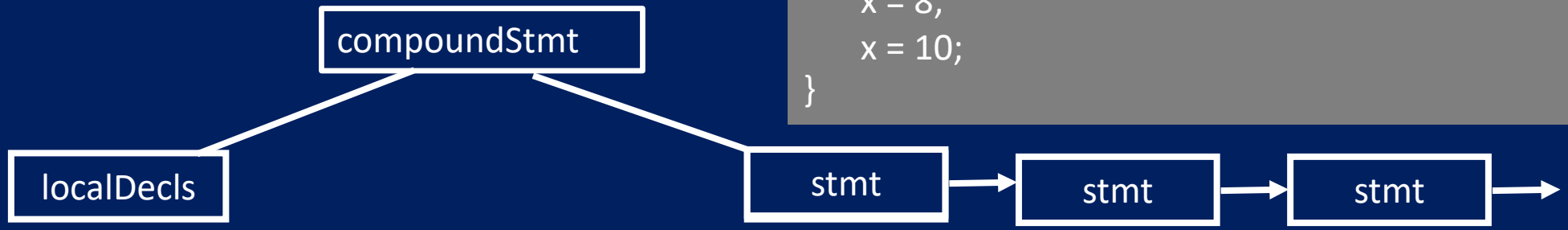
```
if_stmt     : IF exp THEN stmt_seq END
              { $$ = newStmtNode(IfK);
                $$->child[0] = $2;
                $$->child[1] = $4;
              }
            | IF exp THEN stmt_seq ELSE
stmt_seq END
              { $$ = newStmtNode(IfK);
                $$->child[0] = $2;
                $$->child[1] = $4;
                $$->child[2] = $6;
              }
          ;
```

bC


```
decl        : varDecl             { $$ = $1;}
            | funDecl   { $$ = $1;}
            ;

varDecl     : typeSpec varDeclList ';'
              { $$ = $2; setType( $2, $1, false); yyerrok; }
              ;
funDecl     : typeSpec ID '(' parms ')' stmt
                { $$ = newDeclNode(FuncK, $1, $2, $4, $6);}
            | ID '(' parms ')' stmt
                 { $$ = newDeclNode(FuncK, Void, $1, $3, $5);}
            ;
```

Compound Statement

```
#DRBC This program is as simple as I can get.
#DRBC This should compile without errors
main ()
{
  int x;
  x = 5;
     x = 8;
     x = 10;
}
```



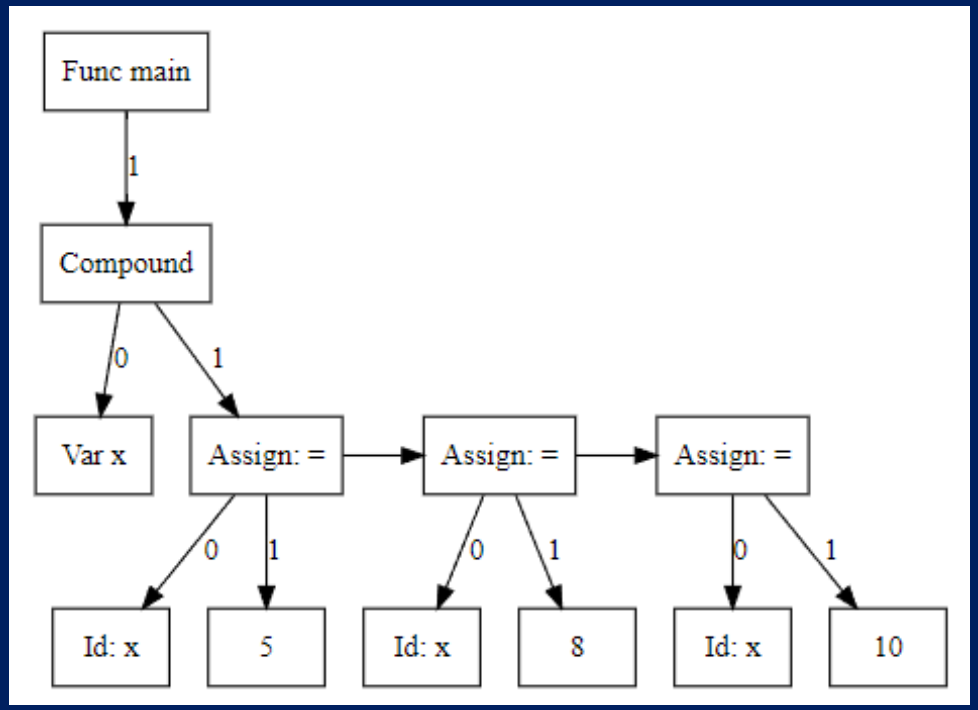
- Declarations and statements are sibling lists ->
- (Declarations could be null)

```
compoundStmt : '{' localDecls stmtList '}'          { $$ = newStmtNode(CompoundK, $1, $2, $3); yyerrok;}
          ;

stmtList     : stmtList stmt          { $$ = ($2==NULL ? $1 : addSibling($1, $2)); } // empty stmt test
          | /* empty */               { $$ = NULL; }
          ;
```

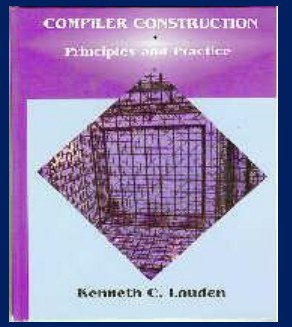
Compound Statement

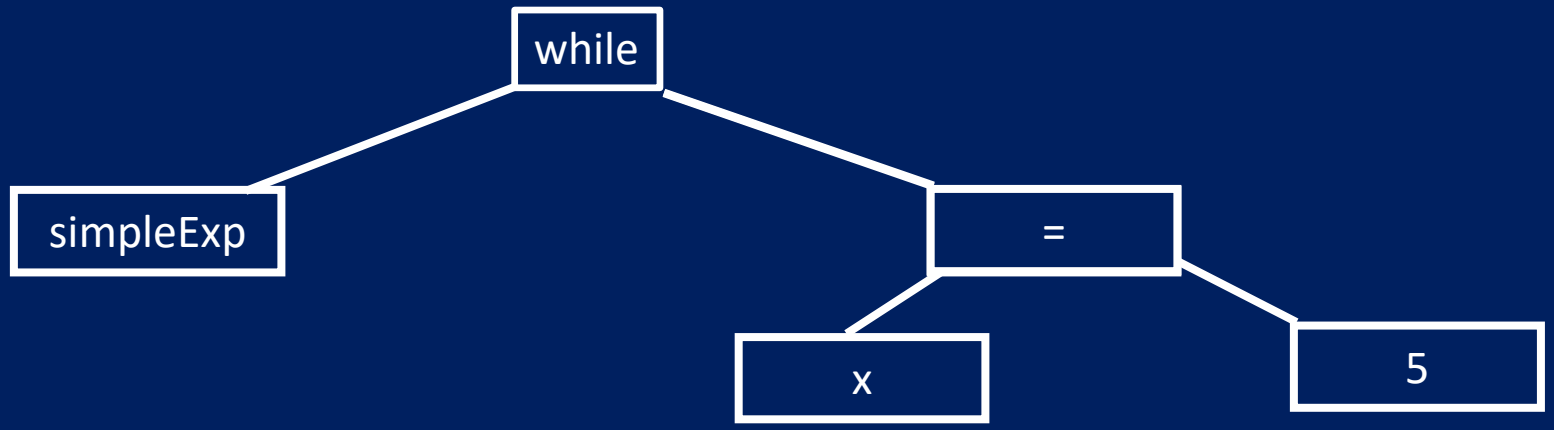
```
#DRBC This program is as simple as I can get.
#DRBC This should compile without errors
main ()
{
  int x;
  x = 5;
     x = 8;
     x = 10;
}
```

```
Func: main returns type void [line: 3]
.  Child: 1  Compound [line: 4]
.  .  Child: 0  Var: x of type int [line: 5]
.  .  Child: 1  Assign: = [line: 6]
.  .  .  Child: 0  Id: x [line: 6]
.  .  .  Child: 1  Const 5 [line: 6]
.  .  Sibling: 1  Assign: = [line: 7]
.  .  .  Child: 0  Id: x [line: 7]
.  .  .  Child: 1  Const 8 [line: 7]
.  .  Sibling: 2  Assign: = [line: 8]
.  .  .  Child: 0  Id: x [line: 8]
.  .  .  Child: 1  Const 10 [line: 8]
```

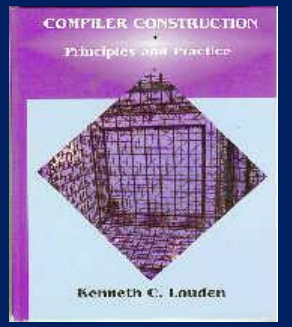


Children






```
matched     : IF simpleExp THEN matched ELSE matched     { $$ = newStmtNode(IfK, $1, $2, $4, $6);}
            | WHILE simpleExp DO matched                 { $$ = newStmtNode(WhileK, $1, $2, $4);}
            | FOR ID '=' iterRange DO matched            { $$ = newStmtNode(ForK, $1, NULL, $4, $6);…
```



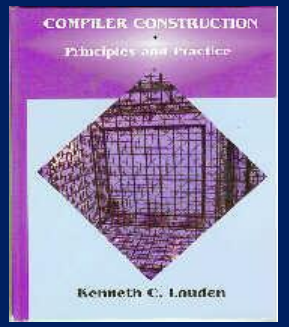
IF <simpleExp> THEN <matched>

Matched means that I don't have an if waiting to be picked up by an else.

IF <simpleExp> THEN <unmatched>

Unmatched means that somewhere in here there is a pending if that doesn't have an else attached to it.

6 +1 possible cases



If(test if test do else do

What can go in 'do'?

Match <nolfs>

No Ifs.

UnMatch IF <simpleExp> THEN <matched>

Matched so OK (but no else)

UnMatch IF <simpleExp> THEN <unmatched>

If it's unmatched at the end, that means there's a trailing if in here that would pick up this else. (But is it legal)

Match IF <simpleExp> THEN <matched> ELSE <matched>

Matched so OK

UnMatch IF <simpleExp> THEN <matched> ELSE <unmatched>

Also Safe

IF <simpleExp> THEN <unmatched> ELSE <matched>

That has if statements that don't have else's attached.

IF <simpleExp> THEN <unmatched> ELSE <unmatched>

This has got unmatched parts before an else.

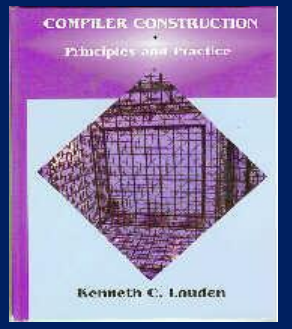
We want to require that this be matched

In treeUtils.cpp

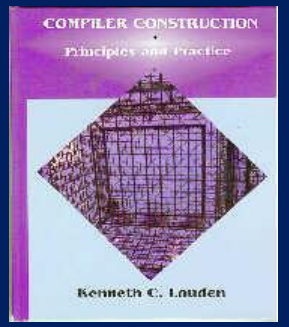
newStmtNode

- compoundStmt
- matched
- unmatched
- iterRange
- returnStmt
- breakStmt

```
treeUtils.cpp
TreeNode *newStmtNode(StmtKind kind,
TokenData *token, TreeNode *c0, TreeNode *c1,
TreeNode *c2)
{
   TreeNode *newNode;
   int i;

   newNode = new TreeNode;
   newNode->nodeNum = nodeNum++;

            <more code>

  return newNode;
}
```



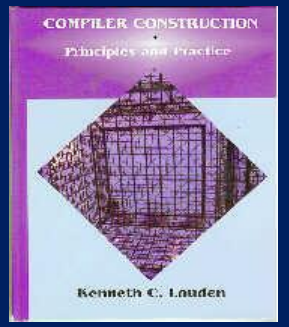
Also in treeUtils.cpp



newExpNode

- exp
- simpleExp
- andExp
- unaryRelExp
- relExp
- minmaxExpn
- sumExp
- mulExp
- unaryExp
- mutable
- call
- constant

Also in treeUtils.cpp



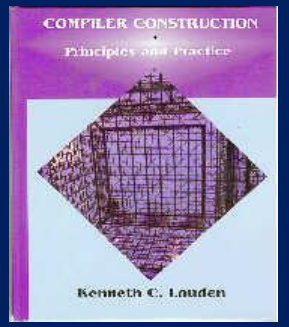
newDeclNode

- varDeclId
- funDecl
- matched : FOR ID '=' iterRange DO matched
- unmatched : FOR ID '=' iterRange DO unmatched

printTree

- Put a call to printTree at the end of your main

In main.cpp



- Include the dot code generator:

  ```
  #include "dot.h"
  ```

- To print the dot tree add this line after your tree is created(Assuming your tree is called syntaxTree):

  ```
  printDotTree(astDot, syntaxTree, false, false);
  ```