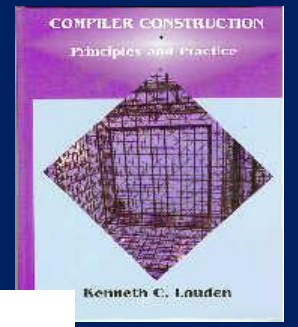
CS 445: Assignment 3

symatnics.cpp

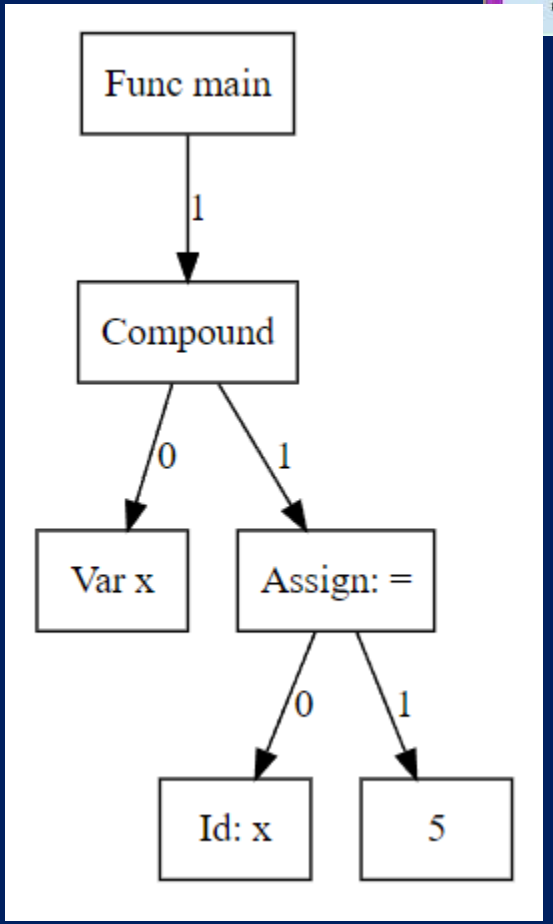
semantics.h

symbolTable.cpp

A simple bC file in assignment 2

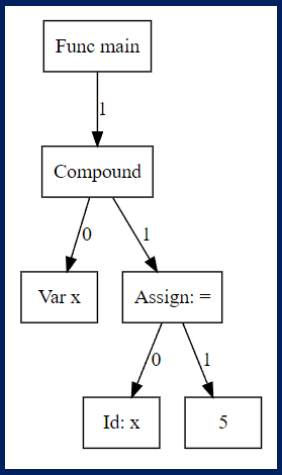


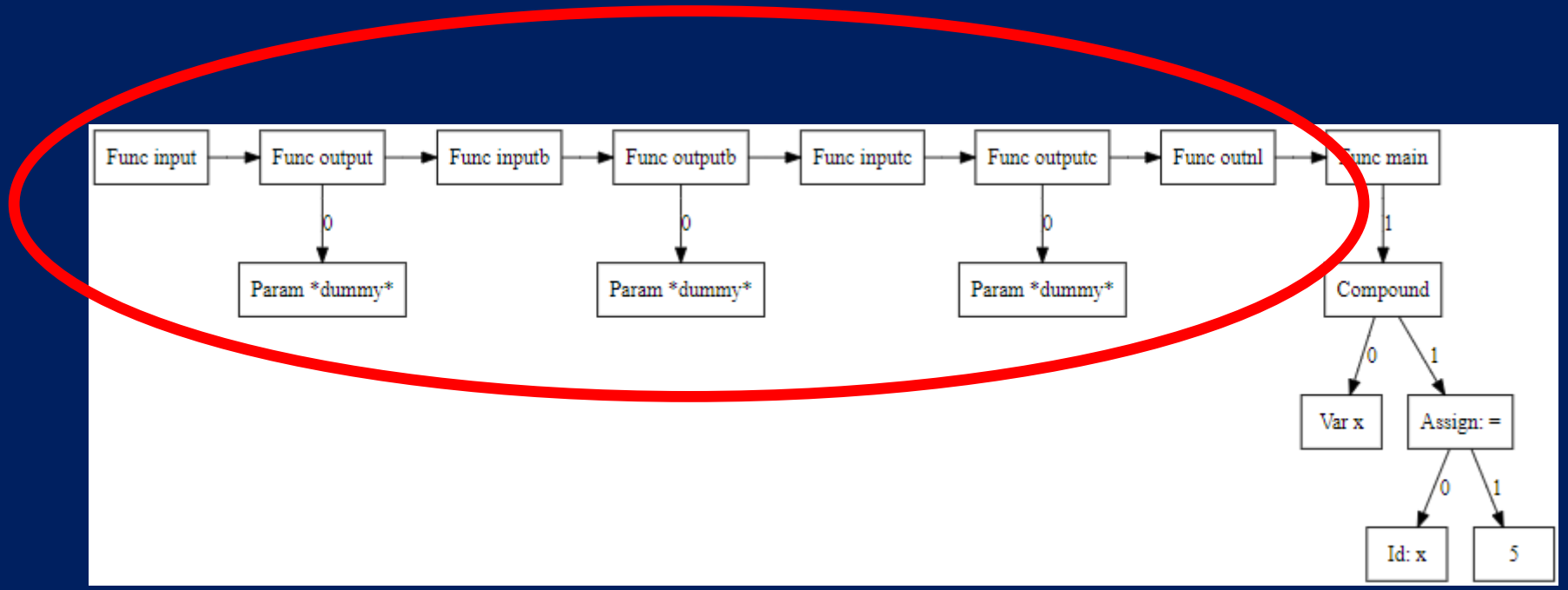

```
#DRBC This program is as simple as I can get.
#DRBC This should compile without errors
main ()
{
  int x;
  x = 5;
}
```




```
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]
```

A simple bC file in assignment 3: 10 new nodes for IOLib



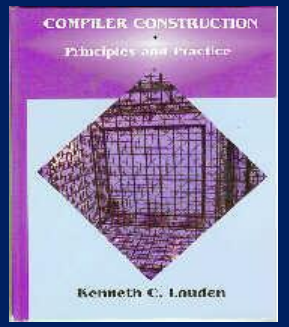


A simple bC file in assignment 3

```
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]
```

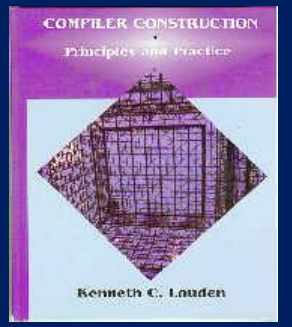
```
Func: input returns type int [line: -1]
Sibling: 1  Func: output returns type void [line: -1]
.  Child: 0  Parm: *dummy* of type int [line: -1]
Sibling: 2  Func: inputb returns type bool [line: -1]
Sibling: 3  Func: outputb returns type void [line: -1]
.  Child: 0  Parm: *dummy* of type bool [line: -1]
Sibling: 4  Func: inputc returns type char [line: -1]
Sibling: 5  Func: outputc returns type void [line: -1]
.  Child: 0  Parm: *dummy* of type char [line: -1]
Sibling: 6  Func: outnl returns type void [line: -1]
Sibling: 7  Func: main returns type void [line: 3]
.  Child: 1  Compound [line: 4]
.  .  Child: 0  Var: x of type int [line: 5]
.  .  Child: 1  Assign: = of type int [line: 6]
.  .  .  Child: 0  Id: x of type int [line: 6]
.  .  .  Child: 1  Const 5 of type int [line: 6]
```

Outline



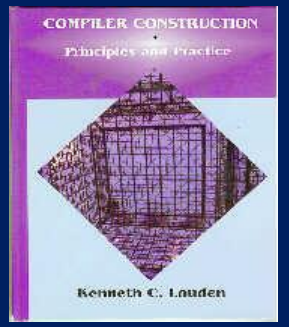
- Changes to main
- 10 new IOLib nodes
- Types for everything
  - Symbol table to find types in scope
  - Dealing with type errors

main.cpp




```
SymbolTable *symtab;
 symtab = new SymbolTable();
 symtab->debug(debugSymTab);
 syntaxTree = semanticAnalysis(syntaxTree,
                               shareCompoundSpace,  // true
                               noDuplicateUndefs,       // false unless u option
                               symtab,
                               globalOffset);
```

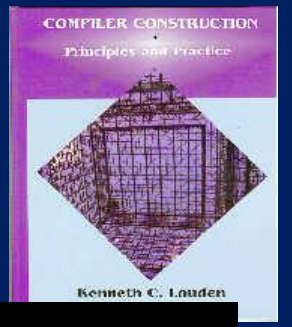
semanticAnalysis(...)




```
syntree = loadIOLib(syntree);

treeTraverse(syntree, symtab);
...
```

Printing the tree

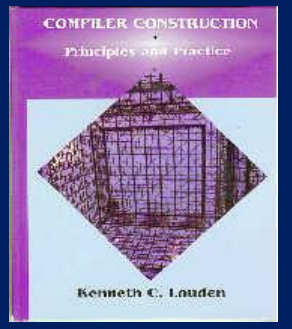
printTree(stdout, syntaxTree, true, false);



In Assignment 2:

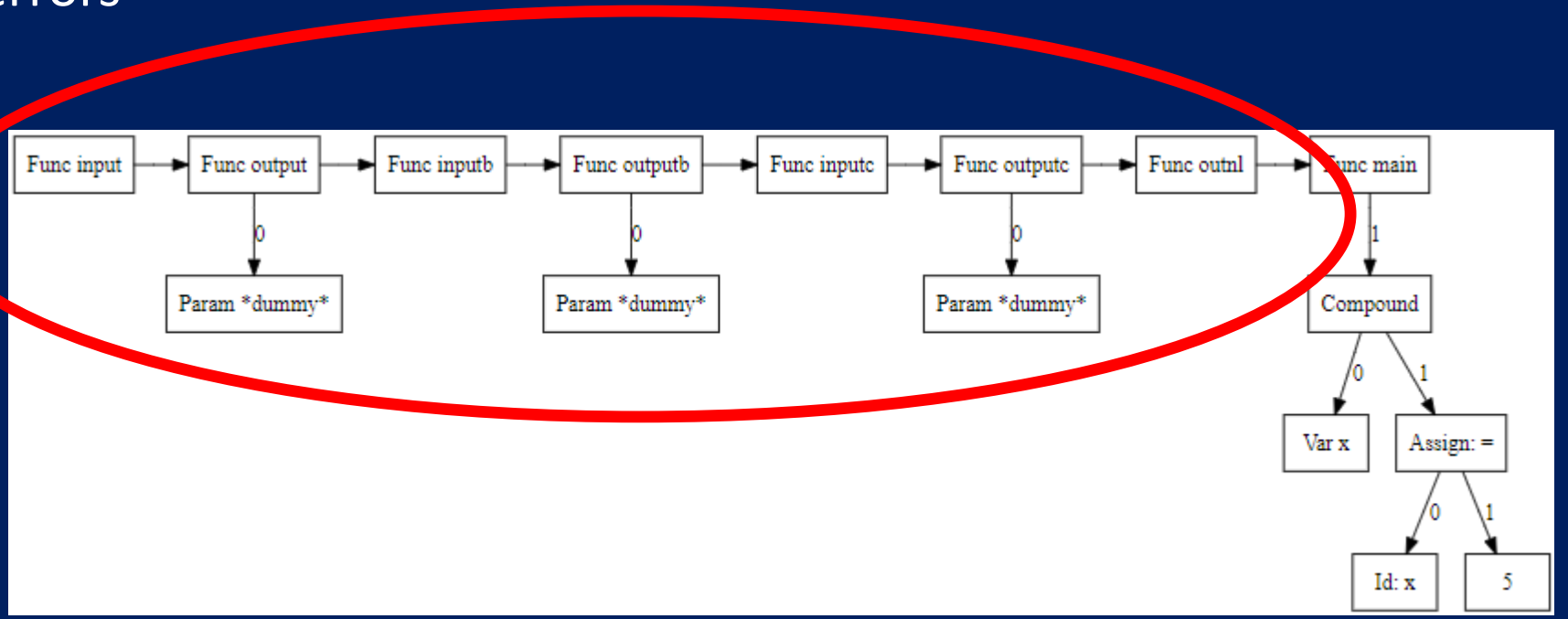
```
Func: main returns type void [line: 3]
.  Child: 1  Compound [line: 4]
.  .  Child: 0  Var: x of type int [line: 5]
.  .  Child: 1  Assign: = [line: 6]
.  .  .  Child: 0  Id: x [line: 6]
.  .  .  Child: 1  Const 5 [line: 6]
Number of warnings: 0
Number of errors: 0
```

```
Func: input returns type int [line: -1]
Sibling: 1  Func: output returns type void [line: -1]
.  Child: 0  Parm: *dummy* of type int [line: -1]
Sibling: 2  Func: inputb returns type bool [line: -1]
Sibling: 3  Func: outputb returns type void [line: -1]
.  Child: 0  Parm: *dummy* of type bool [line: -1]
Sibling: 4  Func: inputc returns type char [line: -1]
Sibling: 5  Func: outputc returns type void [line: -1]
.  Child: 0  Parm: *dummy* of type char [line: -1]
Sibling: 6  Func: outnl returns type void [line: -1]
Sibling: 7  Func: main returns type void [line: 3]
.  Child: 1  Compound [line: 4]
.  .  Child: 0  Var: x of type int [line: 5]
.  .  Child: 1  Assign: = of type int [line: 6]
.  .  .  Child: 0  Id: x of type int [line: 6]
.  .  .  Child: 1  Const 5 of type int [line: 6]
Number of warnings: 0
Number of errors: 0
```

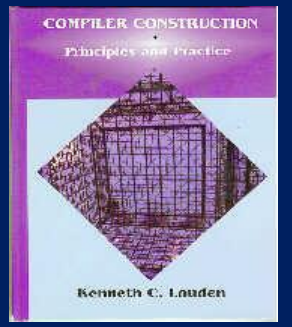


Outline

- Changes to main
- 10 new IOLib nodes
- Types for everything
  - Symbol table to find types in scope
  - Dealing with type errors

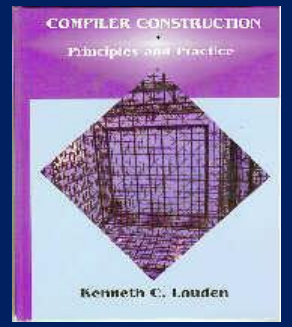


TreeNode *loadIOLib(TreeNode *syntree)




```
TreeNode *input, *output, *param_output;
TreeNode *inputb, *outputb, *param_outputb;
TreeNode *inputc, *outputc, *param_outputc;
TreeNode *outnl;

//////// Stuff from next slides

// link them and prefix the tree we are interested in traversing.
// This will put the symbols in the symbol table.
input->sibling = output;
output->sibling = inputb;
inputb->sibling = outputb;
outputb->sibling = inputc;
inputc->sibling = outputc;
outputc->sibling = outnl;
outnl->sibling = syntree;  // add in the tree we were given
return input;
```

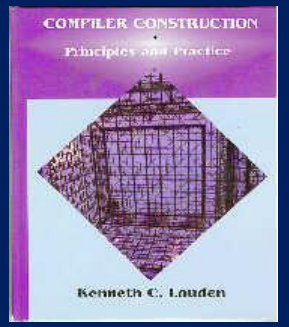
TreeNode *loadIOLib(TreeNode *syntree)



1. input = newDeclNode(FuncK, Integer);
   input->lineno = -1; // all are -1
   input->attr.name = strdup("input"); //We named the variables well
   input->type = Integer;
2. inputb = newDeclNode(FuncK, Boolean);
3. inputc = newDeclNode(FuncK, Boolean);
4. param_output = newDeclNode(ParamK, Void);
5. output = newDeclNode(FuncK, Void);
6. param_outputb = newDeclNode(ParamK, Void);
7. outputb = newDeclNode(FuncK, Void);
8. param_outputc = newDeclNode(ParamK, Void);
9. outputc = newDeclNode(FuncK, Void);
10. outnl = newDeclNode(FuncK, Void);

TreeNode *loadIOLib(TreeNode *syntree)

Exceptions



3. inputc = newDeclNode(FuncK, Boolean);

```
inputc->type = Char;
```

4. param_output = newDeclNode(ParamK, Void);

```
//lineno not needed
param_output->attr.name = strdup("*dummy*");
param_output->type = Integer;
```

6. param_outputb = newDeclNode(ParamK, Void);

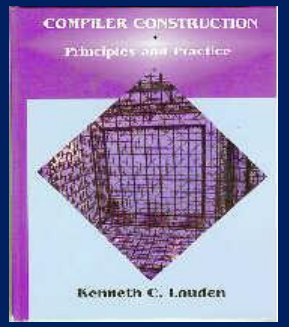
```
//lineno not needed
param_outputb->attr.name = strdup("*dummy*");
param_outputb->type = Boolean;
```

8. param_outputc = newDeclNode(ParamK, Void);

```
// lineno not needed
param_outputc->attr.name = strdup("*dummy*");
param_outputc->type = Char;
```

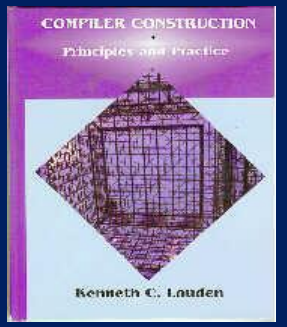
TreeNode *loadIOLib(TreeNode *syntree)

Linking



5. output = newDeclNode(FuncK, Void);
   output->child[0] = param_output;
6. param_outputb = newDeclNode(ParamK, Void);
7. outputb = newDeclNode(FuncK, Void);
   outputb->child[0] = param_outputb;
8. param_outputc = newDeclNode(ParamK, Void);
9. outputc = newDeclNode(FuncK, Void);
   outputc->child[0] = param_outputc;
10. outnl = newDeclNode(FuncK, Void);
    outnl->child[0] = NULL;

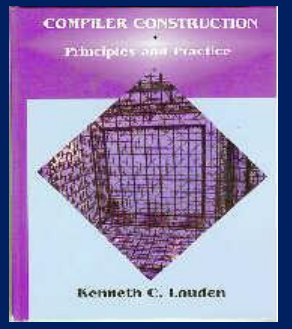
Outline



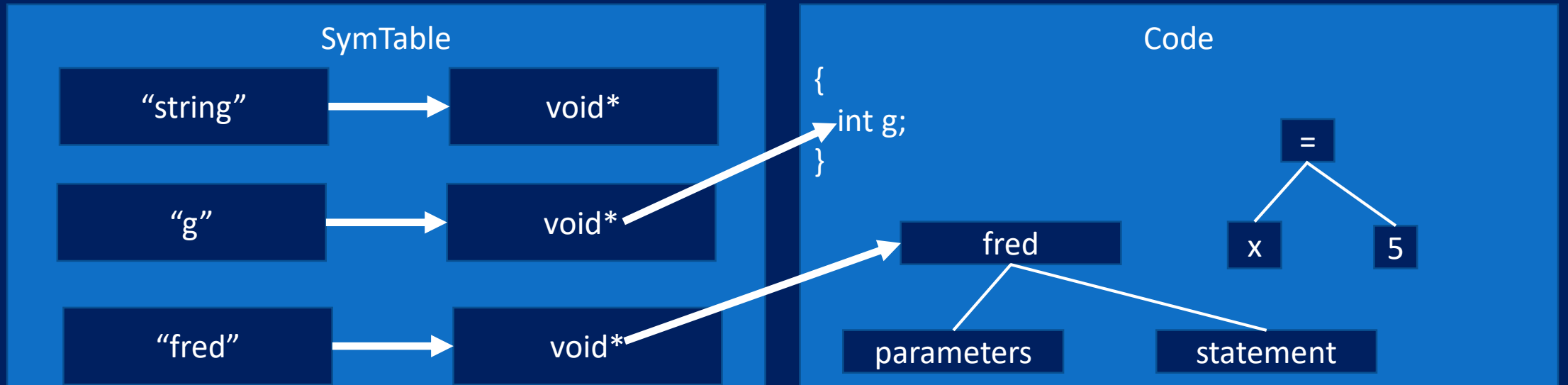
- Changes to main
- 10 new IOLib nodes
- Types for everything
  - Symbol table to find types in scope
  - Dealing with type errors

```
Func: input returns type int [line: -1]
Sibling: 1  Func: output returns type void [line: -1]
.   Child: 0  Parm: *dummy* of type int [line: -1]
Sibling: 2  Func: inputb returns type bool [line: -1]
Sibling: 3  Func: outputb returns type void [line: -1]
.   Child: 0  Parm: *dummy* of type bool [line: -1]
Sibling: 4  Func: inputc returns type char [line: -1]
Sibling: 5  Func: outputc returns type void [line: -1]
.   Child: 0  Parm: *dummy* of type char [line: -1]
Sibling: 6  Func: outnl returns type void [line: -1]
Sibling: 7  Func: main returns type void [line: 3]
.   Child: 1  Compound [line: 4]
.   .   Child: 0  Var: x of type int [line: 5]
.   .   Child: 1  Assign: = of type int [line: 6]
.   .   .   Child: 0  Id: x of type int [line: 6]
.   .   .   Child: 1  Const 5 of type int [line: 6]
```

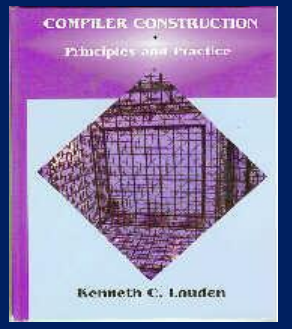
SymbolTable



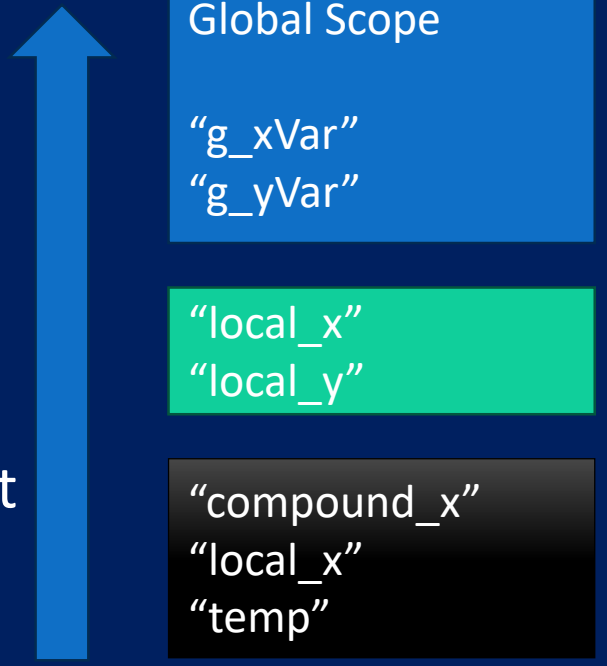
- A mapping from a string to void*
  - The void* points to the declaration of the variable.
  - Anytime you have an identifier (like the name of a procedure or a variable) something that you need to tag with what its type is, then
    - we'll put that name into the as the string part and
    - the void pointer will point into the code for the declaration



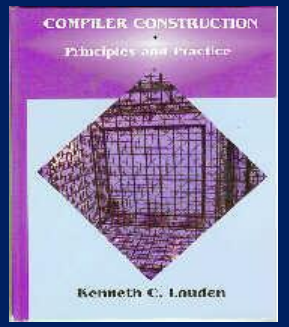
We need to manage scope



- The symbol table is a stack of symbol tables.
  - when you look up the symbol table code, you'll find that in that class there are two types.
    - One is the symbol table type, which is a stack of scopes,
    - and these are listed as scopes.
- The global scope is always there, and the symbol table will yell at you if you try to get rid of it.
  - But you can push and pop scopes and those are the two next procedures you need, enter and leave.
- When I look things up, I start at the top of the stack and return the most recent occurrence.

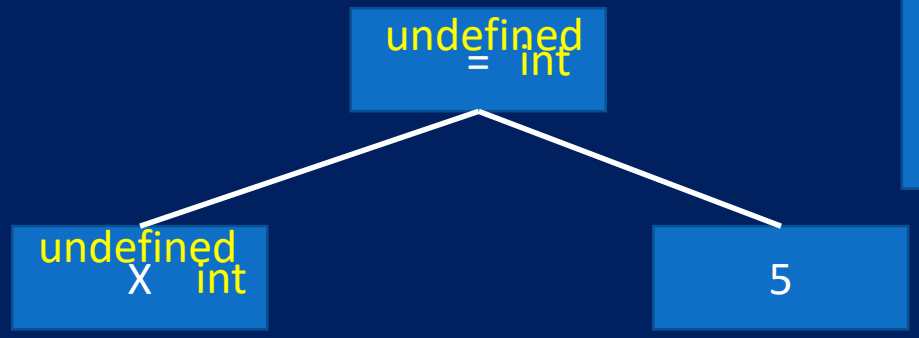


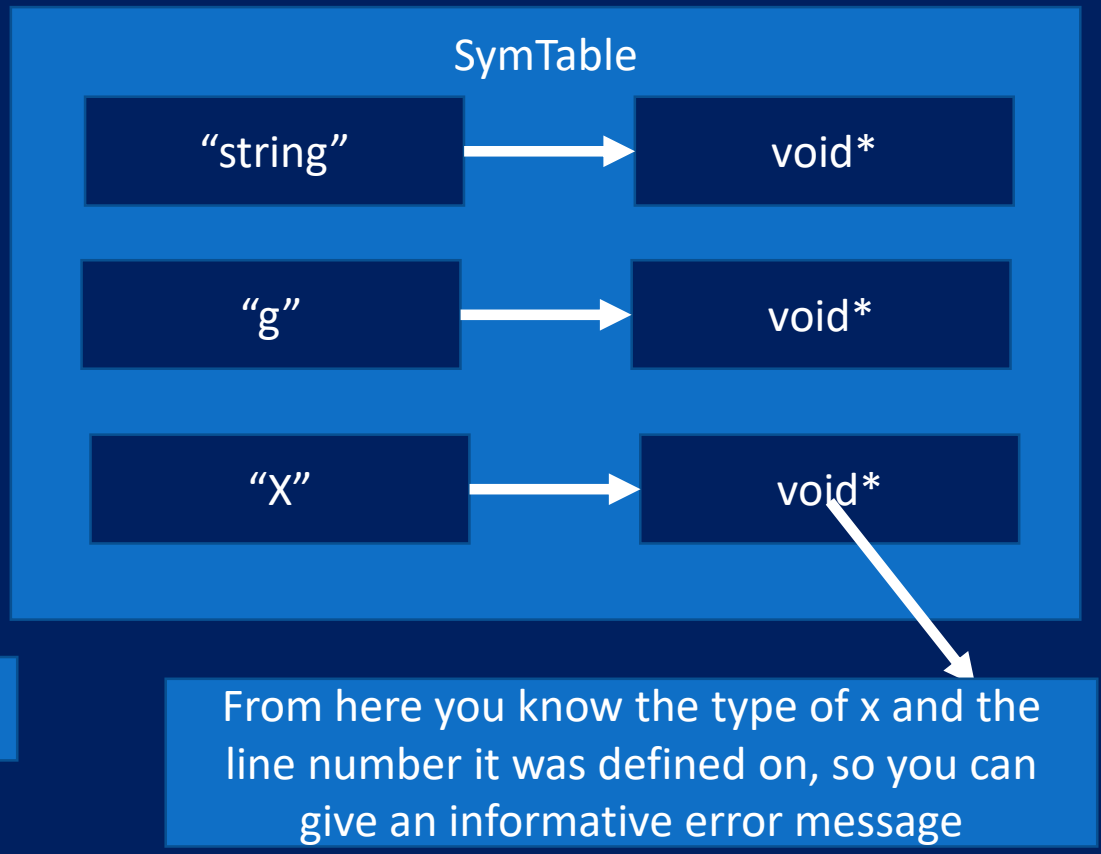
Symbol Table methods



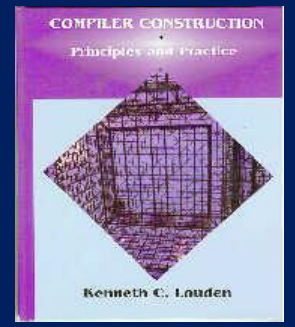
- 4 routines
  - insert into symbol table (Any time you have a declaration)
    - Saves info like the type
    - Gives an error on multiple declarations of the same variable in the same scope
  - lookup (any time you get an id, like x=5, we need to look up x to see if we can put a 5 in it)
    - In x=5 we look up the type of the left and right side to see if they match. (We have no coercion)
    - If they don't match, we have an error for that.
    - X=5 the type of an assignment is the type of the left hand side of the assignment in bC
  - enter scope
  - leave scope

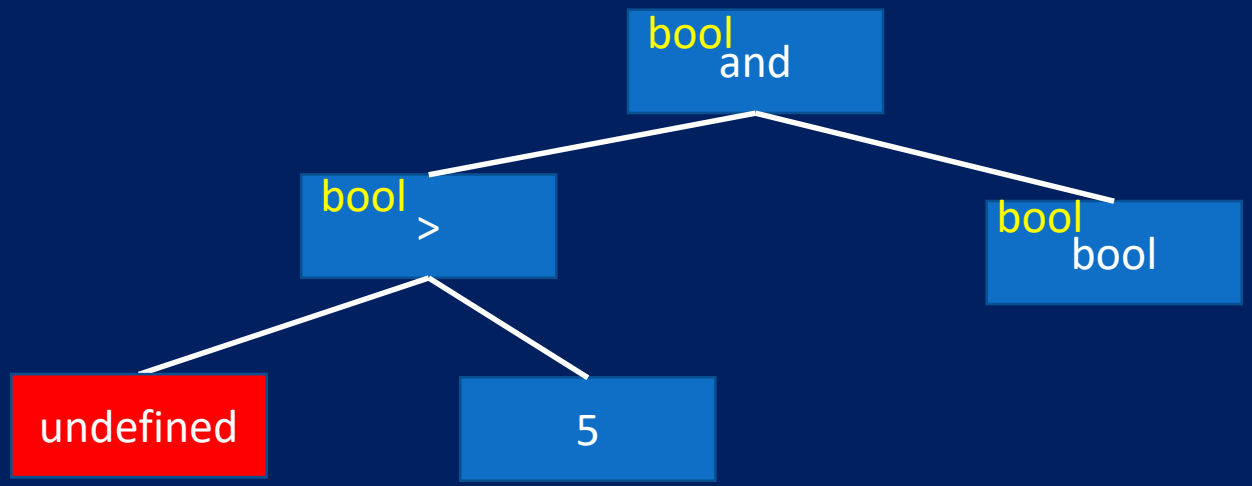
Assignment





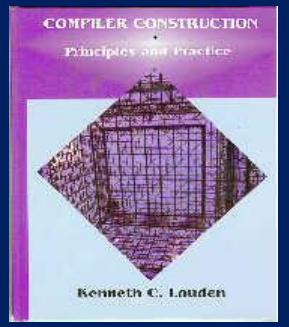
- X=5 the type of an assignment is the type of the left hand side of the assignment in bC
  - Look up X in the symbol table.
    - If X is an int, then = is int.
    - If X is undefined, then = is undefined.
- undefined = 5 sends undefined up the tree.
- int x = undefined sends int up the tree





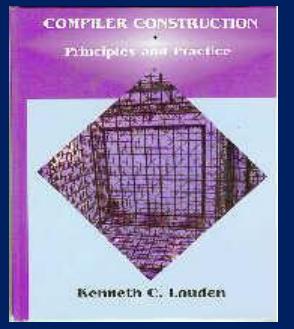
- Because we do not want the error to cascade up the tree
  - Even if both values in the > are undefined the result is a bool

Useful global variables

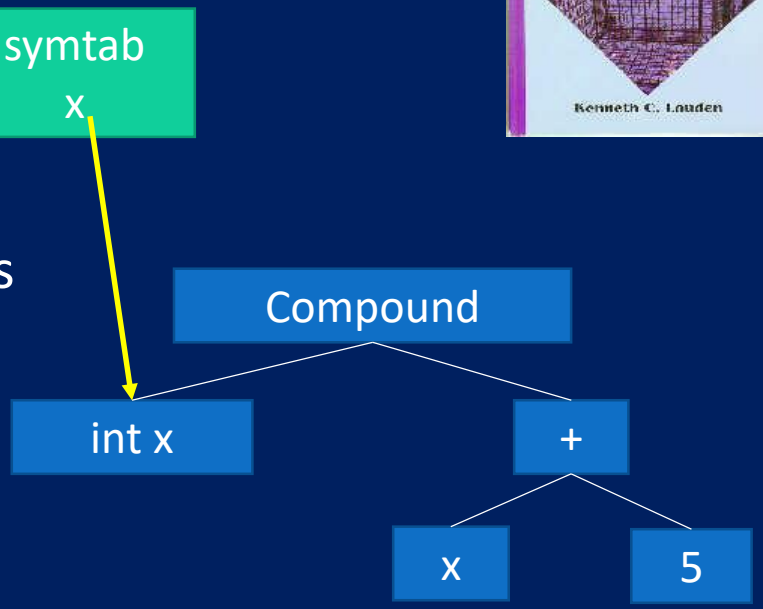



```
// memory offsets are GLOBAL
static int goffset;                 // top of global space
static int foffset;                 // top of local space
```

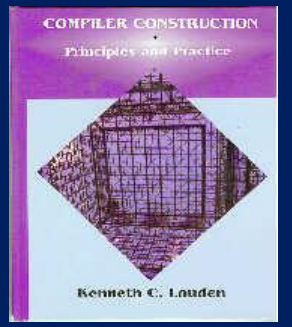
What YOU will write for the compound statement (roughly)



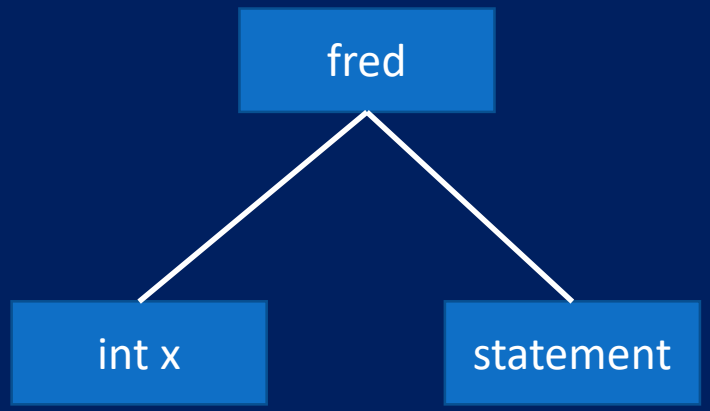

```
symtab->enter((char *)"compoundStmt");
    treeTraverse(current->child[0], symtab); // process declarations
   // More stuff
    current->size = foffset;
    treeTraverse(current->child[1], symtab);
    // More stuff
symtab->leave(); //end of processing of the compound statement
                 //because you can throw away the scope.
```



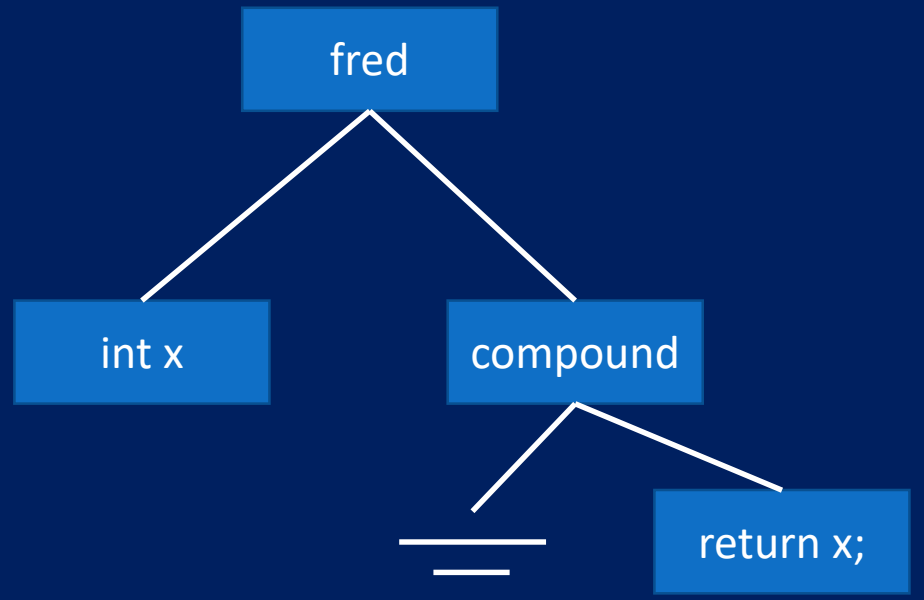
Remember: compound statements occurring right after a function are treated differently.



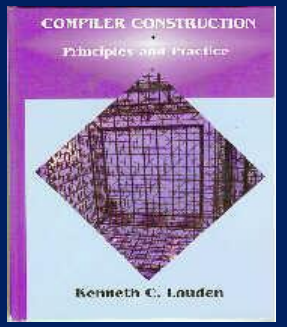
fred(int x) return x;



fred(int x) {return x;}



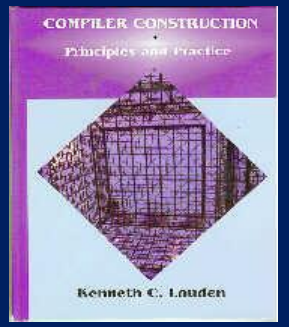
Outline



- Changes to main
- 10 new IOLib nodes
- Types for everything
  - Symbol table to find types in scope
  - Dealing with type errors

```
Func: input returns type int [line: -1]
Sibling: 1  Func: output returns type void [line: -1]
.   Child: 0  Parm: *dummy* of type int [line: -1]
Sibling: 2  Func: inputb returns type bool [line: -1]
Sibling: 3  Func: outputb returns type void [line: -1]
.   Child: 0  Parm: *dummy* of type bool [line: -1]
Sibling: 4  Func: inputc returns type char [line: -1]
Sibling: 5  Func: outputc returns type void [line: -1]
.   Child: 0  Parm: *dummy* of type char [line: -1]
Sibling: 6  Func: outnl returns type void [line: -1]
Sibling: 7  Func: main returns type void [line: 3]
.   Child: 1  Compound [line: 4]
.   .   Child: 0  Var: x of type int [line: 5]
.   .   Child: 1  Assign: = of type int [line: 6]
.   .   .   Child: 0  Id: x of type int [line: 6]
.   .   .   Child: 1  Const 5 of type int [line: 6]
```

Redefinition of variables



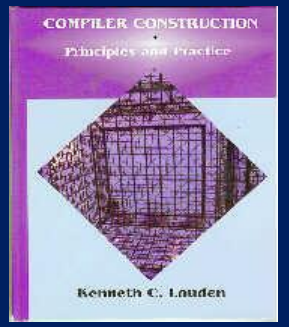

```
int x;
int x; // x is already defined

y = 73; //y is undefined

myFunction(int x) {
  int x; // Even in C this is not OK, It is a weird special case.
  {
    int x; // This is OK
  }
}
```

basicAll

Kenneth C. Louden

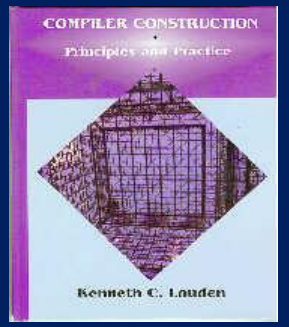
```
//
char dog(char x)
{
   char x;   // SPECIAL CASE: params are in function compound statement
             // Symbol %s is already defined at line %d
   int b;

   cat(10);  // Symbol 'cat' is not defined

   return b; // Expecting return type of %s but got type %s
}

int dog(int x)  // Symbol %s is already defined at line %d
{
   {
      int x;
   }
}             // missing return warning

char cat(char x)
{
   int b;

   return; // Expecting a return type of %s but got none
}
```

Order of traversal



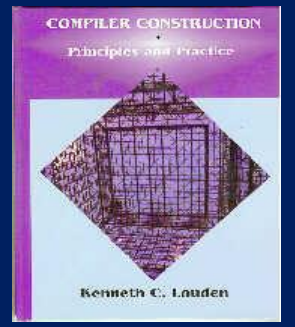
- child[0]
- Current node
- child[1]
- child[2]
- sibling

New scope if:



- DeclK
  - FuncK
    - Also set foffset = -2;
- StmtK
  - CompoundK
  - ForK
  - IfK
  - WhileK

Scope



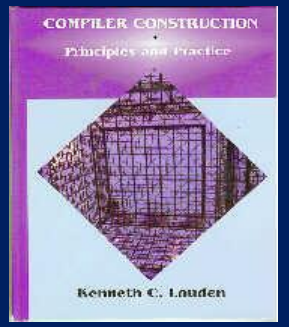

```
if(isCompound) {
      char *id = strdup("{");
      symtab->enter("NewScope from " + (std::string)id);
  }
```

- child[0]
- Current node
- child[1]
- child[2]

```
if(isCompound) {
      symtab->leave();
}
```

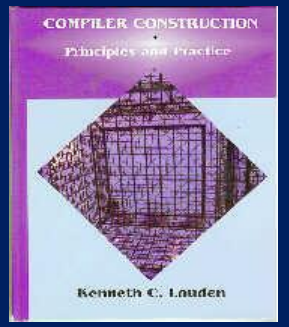
- sibling

Types



- ExpK
  - AssignK & OpK
    - Boolean
      - AND
      - OR
      - EQ
      - NEQ
      - LEQ
      - <
      - GEQ
      - >
      - NOT
    - Integer
      - Any value < LASTOP
    - Whatever child[0]’s type is (and if it is an array)
      - =
      - [

Types

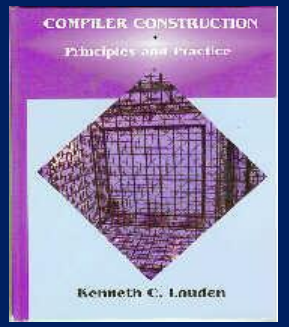


- ExpK
  - IdK
    - Look up the type in the symtab.

```
if ((tmp = (TreeNode *)(symtab->lookup(current->attr.name)))) {
        current->type = tmp->type;
        current->isStatic = tmp->isStatic;
        current->isArray = tmp->isArray;
        current->size = tmp->size;
        current->varKind = tmp->varKind;
        current->offset = tmp->offset;
    }
```

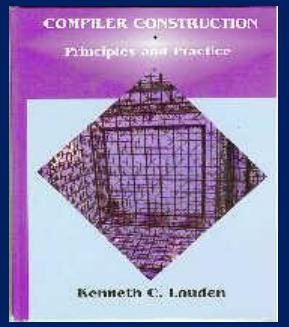
  - CallK
    - Look up like for IdK and set type and size

TMBC Offsets (See the tmDiscription)



- goffset - the global offset is the relative offset of the next available space in the global space.
- foffset - the frame offset is the relative offset of the next available space in the frame being built
- toffset - the temp offset is the offset from the frame offset next available temp variable.

Offset calculations



- DeclK
  - VarK & ParamK

if (symtab->depth()==1) // This is a global variable since it is not in a function

// Set the varKind to Global for VarK (Parameter for ParamK)

current->offset = goffset;

goffset -= current->size;

Otherwise, if current->isStatic // This is a static variable

// Set varKind to LocalStatic for VarK (Parameter for ParamK)

current->offset = goffset;

goffset -= current->size;

//symtab->insertGlobal with a unique name. (Keep a static int variable and append it to the end of the name)

Otherwise, treat it as normal

// Set varKind to Local for VarK (Parameter for ParamK)

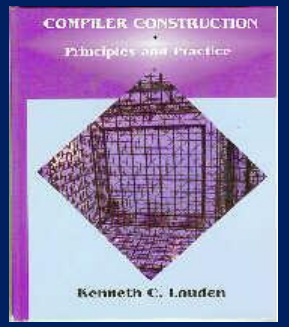
current->offset = foffset;

foffset -= current->size;

For VarK only

if (current->isArray) current->offset--; // pt to array after size

Offset calculations



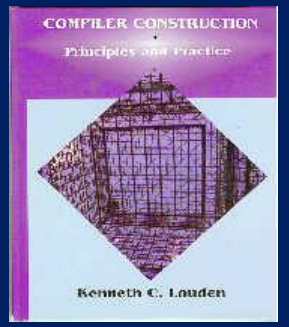
- DeclK
  - FuncK

    foffset = -2;

    Traverse child[0]’s tree (where the parameters are)

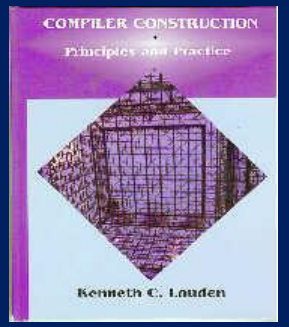
    current->size = foffset;

Offset calculations



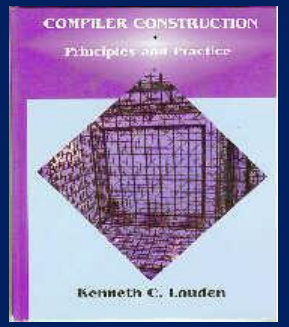
- StmtK
  - CompoundK
    - If newScope
      - Remember the current Offset
      - Traverse child[0]’s tree
      - current->size = foffset;
      - Traverse child[1]’s tree
      - Remember the current Offset
    - Otherwise
      - Traverse child[0]’s tree
      - current->size = foffset;
      - Traverse child[1]’s tree

Offset calculations



- StmtK
  - ForK
    - Remember the current Offset
    - Traverse child[0]'s tree
    - foffset-=2; // Make space for the for loop var
    - current->size = foffset;
    - Traverse child[1]'s tree
    - Traverse child[2]'s tree
    - Remember the current Offset

Offset calculations



- ExpK
  - ConstantK

```
if (current->type == Char && current->isArray) {  // Deal with strings
        current->varKind = Global;
        current->offset = goffset - 1;
        goffset -= current->size;
    }
```

  - IdK

```
tmp = (TreeNode *)(symtab->lookup(current->attr.name) // Look up in the symbol table
current->offset = tmp->offset;
```