```
////////////////////////////////////////////////////////////////////////////
////Class:          CS 445
////Semester:       Fall 2011
////Assignment:     Homework 4
////Author:         Dr. Robert Heckendorn, modified by Colby Blair
////File name:      symtab.c
////////////////////////////////////////////////////////////////////////////

#include "symtabc.h"
static int initMaxTable_=100;  // this is the initial size of the symbol tables
static int maxTable_;          // this is how big the table is now since it can grow
static SymTabEntry *table_;    // the table is simply a fancy stack of SymTabEntries
static SymTabEntry *top_;      // this is where the next new data will be added
static char *scopeName_;       // this is the current scope name
static int scopeDepth_;        // this is the currect depth of the scopes
static void (* elemPrint_)(void *);  // this is a print routine to print your TreeNo
de *
static int debug_;             // this holds debug flags defined above

//
// Class SymTab
//
// A general simple stack of symbol tables that maps
// a char * to a void *.  Provides a user definable
// print routine for the objects stored in the symbol table.
// The print rouinte is defined when the constructor is called.
//
// debug flags setable by the debug method:
//   DEBUG_TABLE - announce entry to a scope and prints the symbol
//       table on exit from a scope.
//   DEBUG_PUSH - print everything that is pushed on the stack (uses
//       the print routine for printing the ptr value (treeNode *?)
// these flags are bit masks and so can be ored together to turn
// on multiple affects.  For example debug(DEBUG_TABLE | DEBUG_PUSH) would
// turn on both the DEBUG_PUSH and DEBUG_TABLE flags.
//
//  The four most important operations are insert, lookup, enter, leave.
//


static void xPrint(void *p)
{
    printf("0x%08x", p);
}


// constructor
void SymTab_init(void (* elemPrint)(void *))
{
    int i;

    maxTable_ = initMaxTable_;
    table_ = (SymTabEntry *)malloc(sizeof(SymTabEntry)*initMaxTable_);
    for (i=0; i<initMaxTable_; i++) table_[i].name="";
    top_ = table_;
    elemPrint_ = elemPrint;
    scopeDepth_ = 0;
    debug_ = 0x0;
    SymTab_enter_scope("globals");
};


// destructor
void SymTab_free()
{
    maxTable_ = 0;
    free(table_);
};

// set the debug flags defined in symtab.h
void SymTab_debug(int newDebugValue)
{
    debug_ = newDebugValue;
```

```
}

// push the sym and ptr on the stack
static void SymTab_push(char *sym, char *type, char *aux_flag,
                        int scopeDepth, void *ptr)
{
    // if you run out of memory then add some
    if (top_>=table_+maxTable_) {
        int offset;
        SymTabEntry *newt;

        // could have used realloc but this is a quick hand translation
        newt = malloc(sizeof(SymTabEntry)*(maxTable_*13)/8);
        memcpy(newt, table_, sizeof(SymTabEntry)*maxTable_);
        free(table_);

        maxTable_=(maxTable_*13)/8;
        printf("SYMTAB: size fault.  Increase to %d elements\n", maxTable_);
        fflush(stdout);
        offset = top_-table_;
        table_ = newt;
        top_ = newt+offset;

    }
    top_->name = sym;
    top_->type = type;
    top_->aux_flag = aux_flag;
    top_->scope = scopeName_;
    top_->depth = scopeDepth;  // note that this is passed in
    top_->ptr = ptr;
    top_++;
};


// prints the symbol table with each element printed using
// the print routine supplied in the constuctor.  New line is supplied
// by this print routine.
void SymTab_print()
{
    SymTabEntry *p;

    printf("\nSymbol Stack:\n");
    for (p=table_; p<top_; p++) {
        // print a regular entry
        if (p->depth) {
//debug     printf("%10s %10s %d 0x%08x ", p->name, p->scope, p->depth, p);
//old       printf("%10s %10s %d ", p->name, p->scope, p->depth);
            printf("%10s %10s %10d %10s %10s", p->name, p->scope, p->depth,
                                               p->type, p->aux_flag);
            elemPrint_(p->ptr);
            printf("\n");
        }
        // print the scope divider
        else {
            printf("%10s %10s ---- \n", p->name, p->scope);
        }
    }
    fflush(stdout);
};


// inserts an element into the symbol table
bool SymTab_insert(char *sym, char *type, char *aux_flag, void *ptr)
{
    SymTabEntry *p;

    for (p=top_-1; p->depth; p--) {
        if (strcmp(p->name, sym)==0) return false;
    }

    if (debug_ & DEBUG_PUSH) {
        printf("SymTab: Pushing this node: ");
```

```
        elemPrint_(ptr);
        printf("\n");
        fflush(stdout);
    }

    SymTab_push(sym, type, aux_flag, scopeDepth_, ptr);
    return true;
};


// lookup the name in the SymTabEntry
// returning the pointer to the thing stored with the symbol
// or NULL if it could not be found
void *SymTab_lookup(char *sym)
{
    SymTabEntry *p;

    for (p=top_-1; p>=table_; p--) {
        if (strcmp(p->name, sym)==0) {
            if (debug_ & DEBUG_LOOKUP) {
                printf("SymTab: looking up: %s and found data: ", sym);
                elemPrint_(p->ptr);
                printf("\n");
                fflush(stdout);
            }

            //return p->ptr; //what the heck is p->ptr suppose to point to?
            return p;
        }
    }
    if (debug_ & DEBUG_LOOKUP) {
        printf("SymTab: looking up: %s and did not find it.\n", sym);
        fflush(stdout);
    }
    return NULL;
};


// lookup the entry in the symbol table
SymTabEntry *lookupSymTabEntry(char *sym)
{
    SymTabEntry *p;

    for (p=top_-1; p>=table_; p--) {
        if (strcmp(p->name, sym)==0) {
            return p;
        }
    }
    return NULL;
};


// create a new scope on the stack
void SymTab_enter_scope(char *funcname)
{
    scopeName_ = funcname;
    if (debug_ & DEBUG_TABLE) printf("SymTab: Entering scope %s\n", scopeName_);
    scopeDepth_++;
    SymTab_push("", "", "", 0, NULL);
};


// leave a scope
bool SymTab_leave_scope()
{
    SymTabEntry *newTop;

    if (debug_ & DEBUG_TABLE) {
        SymTab_print();
        printf("SymTab: Leaving scope %s ", scopeName_);
        fflush(stdout);
    }
```

```
    newTop = lookupSymTabEntry("");
    if (newTop>table_) {
        top_ = newTop;
        scopeName_ = (top_-1)->scope;
        if (debug_ & DEBUG_TABLE) {
            printf("and entering scope %s\n", scopeName_);
            fflush(stdout);
        }
        if (scopeDepth_>1) scopeDepth_--;
        return true;
    }
    if (debug_ & DEBUG_TABLE) printf("\n");
//debug    printf("ERROR(symbol table): You cannot leave global scope.\n");
    return false;
};


// the depth of the scope stack with the first real scope (probably
// globals) numbered 1
int SymTab_depth()
{
    return scopeDepth_;
}

// number of real entries in the whole table
int SymTab_numEntries()
{
    return (top_-table_)-scopeDepth_;
}
```