```
/***********************************************
 * File: var.c          Project by: David Hodgdon *
 * Class: ECE476        Instructor: Bruce Land    *
 ***********************************************/
#include "scan.h"
#include "var.h"
#include "lcd.h"

//#define VARMEMSIZE 34
#define VARMEMSIZE 20
static struct VarDesc freeTable[VARMEMSIZE];
static struct VarDesc stringHeadDesc;   Var stringHead=&stringHeadDesc;
static struct VarDesc stringTailDesc;   Var stringTail=&stringTailDesc;
static struct VarDesc realHeadDesc;     Var realHead=&realHeadDesc;
static struct VarDesc realTailDesc;     Var realTail=&realTailDesc;
static struct VarDesc intHeadDesc;              Var intHead=&intHeadDesc;
static struct VarDesc intTailDesc;              Var intTail=&intTailDesc;

static struct VarDesc stringArrayHeadDesc;              Var stringArrayHead=&str
ingArrayHeadDesc;
static struct VarDesc stringArrayTailDesc;              Var stringArrayTail=&str
ingArrayTailDesc;
static struct VarDesc realArrayHeadDesc;                Var realArrayHead=&realA
rrayHeadDesc;
static struct VarDesc realArrayTailDesc;                Var realArrayTail=&realA
rrayTailDesc;
static struct VarDesc intArrayHeadDesc;         Var intArrayHead=&intArrayHeadDe
sc;
static struct VarDesc intArrayTailDesc;         Var intArrayTail=&intArrayTailDe
sc;

static struct VarDesc freeHeadDesc;     Var freeHead=&freeHeadDesc;
static struct VarDesc freeTailDesc;     Var freeTail=&freeTailDesc;

static char freeVars=VARMEMSIZE;
#if 1
#define CHARMEMSIZE 80
#define NUMSTRINGS 6
//#define CHARMEMSIZE 15
//#define NUMSTRINGS 3
struct MallocDesc{
        char* begin;
        char size;
};
char charMem[CHARMEMSIZE];
struct MallocDesc charMemLog[NUMSTRINGS];


void  initMalloc() {
 char i;
 for(i=0;i<NUMSTRINGS;i++) {
        charMemLog[i].begin=NULL;
        charMemLog[i].size=0;
 }
}

#define MAX(a, b) (a > b ? a : b)

char * xmalloc(char size) {
  char i;
  char unused=-1;
  unsigned char free;
  char freeIdx;
// precondition array sorted so that free are in
// the front and used are sorted by begin address
  unused=-1;
  freeIdx=-1;
  for(i=0;i<NUMSTRINGS;i++) {
        if(charMemLog[i].begin==0) { // look at empty
```



```
                unused=i;
        }
        if(charMemLog[i].begin!=0) { // non empty
                if(i!=NUMSTRINGS-1) {
                        free=(charMemLog[i].begin)-(charMemLog[i].begin+charMemL
og[i].size);
                        if(free>=size) {
                                freeIdx=i;
                        }
                }
        }
  }

        free=(charMem+CHARMEMSIZE)-MAX((charMemLog[NUMSTRINGS-1].begin+charMemLo
g[NUMSTRINGS-1].size), charMem);
        if(free>=size) {
                freeIdx=NUMSTRINGS-1;
        }
        free=(charMemLog[unused+1].begin-charMem);
        if(free>=size) {
                freeIdx=unused;
        }

  if(unused==-1) ScanError("No String Memory Left\n");
  if(freeIdx==-1) ScanError("No Character Memory Left\n");
  // move all from unused+1 to freeIdx down one
  for(i=unused;i<freeIdx;i++) {
        charMemLog[i] = charMemLog[i+1];
  }
  charMemLog[freeIdx].begin=MAX(charMemLog[freeIdx-1].begin+charMemLog[freeIdx-1
].size,charMem);
  charMemLog[freeIdx].size=size;
  return charMemLog[freeIdx].begin;
}

void xfree(char * ptr) {
 char i;
 char pos;
 char freeIdx;
 for(i=0;i<NUMSTRINGS && charMemLog[i].begin!=ptr;i++) {
         if(charMemLog[i].begin==0) freeIdx=i;
 }
 pos=i;
 if(pos==NUMSTRINGS) ScanError("ptr not in table\n");
 for(i=pos;i>freeIdx;i--) {
        charMemLog[i]=charMemLog[i-1];
 }
 charMemLog[i].begin=NULL;
}
#endif

#define INITIALIZE_STR_LL(FIRST,LAST) {\
        FIRST->next=LAST;\
        FIRST->prev=NULL;\
        FIRST->str[0]='\0';\
        LAST->next=NULL;\
        LAST->prev=FIRST;\
        LAST->str[0]='\0';\
}

#define INITIALIZE_VAR_LL(FIRST,LAST) {\
        FIRST->next=LAST;\
        FIRST->prev=NULL;\
        FIRST->str=NULL;\
        FIRST->type=InvalidType;\
        FIRST->intVal=0;\
        FIRST->realVal=0.0;\
        LAST->next=NULL;\
        LAST->prev=FIRST;\
```

```
        LAST->str=NULL;\
        LAST->type=InvalidType;\
        LAST->intVal=0;\
        LAST->realVal=0.0;\
}
#define REMOVE_LL(X) {\
        X->prev->next=X->next;\
        X->next->prev=X->prev;\
}
#define APPEND_LL(X, TAIL) {\
        X->prev=TAIL->prev;\
        X->next=TAIL;\
        TAIL->prev->next=X;\
        TAIL->prev=X;\
}

void VarNewStr(Var v, unsigned char size) {
        v->str = (char*)xmalloc(sizeof(char)*((int)size+1));
        v->len=size;
}

Var VarNewVar(enum VarType type, char * ident, char i1, char i2, char i3, char i
sTemp) {
        Var v;
        freeVars--;
#ifdef DEBUG_MSG
        if(freeTail->prev==freeHead) ScanError("VarNewVar: Not Enough Memory\n");
#endif
        v=freeTail->prev;
        REMOVE_LL(v);
        v->type=type;
        switch(type) {
                case String:    case StringArray:               v->type=String;
break;
                case Integer:   case IntegerArray:              v->type=Integer;
break;
                case Real:              case RealArray:                 v->type=
Real;   break;
        }
        v->ident[0]=ident[0];
        v->ident[1]=ident[1];
        v->ident[2]='\0';
        v->intVal=0;
        v->realVal=0;
        v->str=NULL;
        v->i1=i1;
        v->i2=i2;
        v->i3=i3;
        if(v->type==String) {
                v->str=NULL;
        }
        v->isTemp=isTemp;
        return v;
}

void VarDestroyVar(Var v) {
        freeVars++;
        if(v->isTemp==0) {REMOVE_LL(v);}
        if(v->str!=NULL){
                xfree(v->str);
        }
        APPEND_LL(v, freeTail);
}

Var VarGetVar(char* str, char num, char i1, char i2, char i3, Var head, Var tail
, enum VarType varType) {
        Var cur;
        for(cur=head->next;!(cur->ident[0]==str[0] && cur->ident[1]==str[1] && A
RRAYIDENTCHECK(cur, num, i1, i2, i3)) && cur!=tail; cur=cur->next);
```



```
        if(cur!=tail){ // we found our desired variable
                return cur;
        }
        else { // we must create a variable
                cur=VarNewVar(varType, str, i1, i2, i3, 0);
                APPEND_LL(cur, tail);
        }
        return cur;
}

void VarInit() {
        unsigned char i;
        initMalloc();
        INITIALIZE_VAR_LL(stringHead, stringTail);
        INITIALIZE_VAR_LL(intHead, intTail);
        INITIALIZE_VAR_LL(realHead, realTail);
        INITIALIZE_VAR_LL(stringArrayHead, stringArrayTail);
        INITIALIZE_VAR_LL(intArrayHead, intArrayTail);
        INITIALIZE_VAR_LL(realArrayHead, realArrayTail);
        INITIALIZE_VAR_LL(freeHead, freeTail);
        for(i=0;i<VARMEMSIZE;i++) {
                APPEND_LL(((Var)freeTable+i), freeTail);
        }
}
```