```
/************************************************
 * File: parse.c        Project by: David Hodgdon *
 * Class: ECE476        Instructor: Bruce Land    *
 ************************************************/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include "parse.h"
#include "scan.h"
#include "var.h"
#include "input.h"
#include "serial.h"
#include "lcd.h"
#include "snd.h"

#define SWAP(A,B) {\
        A^=B;\
        B^=A;\
        A^=B;\
}

static void ParseLine();
int newPC; // this allows only half of the PC values that target allows
static enum ScanSymbol s, sBackup;
int curPC;
int dataPC=-1;

#define MAXFORLOOPS 10
static struct ForDesc forLoops[MAXFORLOOPS];
static char numForLoops=0;
static char inData=0;
#ifndef ATMEL
static FILE* fTemp;
#endif

static float lastRand=0;

extern Var stringHead;
extern Var stringTail;
extern Var realHead;
extern Var realTail;
extern Var intHead;
extern Var intTail;
extern Var freeHead;
extern Var freeTail;

extern Var stringArrayHead;
extern Var stringArrayTail;
extern Var realArrayHead;
extern Var realArrayTail;
extern Var intArrayHead;
extern Var intArrayTail;



static char interIdent[]=" ";

// 0=index1,0=index2,0=index3,1=a temp
#define VarNewTemp(T) (VarNewVar(T, interIdent, 0,0,0,1))
Var ParseDoOpUnary(Var x, enum ScanSymbol op);

Var ParseDoOp(Var x, Var y, enum ScanSymbol op) {
        Var result;
        if(x->type==Real && y->type==Integer) y=ParseDoOpUnary(y, ScanREAL);
        else if(x->type==Integer && y->type==Real) x=ParseDoOpUnary(x,ScanREAL);
#ifdef DEBUG_MSG
        else if(x->type!=y->type) ScanError("ParseDoOp: Incompatible Types\n");
#endif
```



```
        result=VarNewTemp(x->type);
        switch(op) {
                case ScanPlus: switch(x->type) {
                        case Integer:   result->intVal=x->intVal+y->intVal;break
;
                        case Real:              result->realVal=x->realVal+y->re
alVal;break;
                        case String:
                                VarNewStr(result, x->len+y->len);
                                memcpy(result->str, x->str,x->len);
                                memcpy(result->str+x->len, y->str, y->len);
                                result->str[x->len+y->len]='\0';
                                break;
                }break;
                case ScanMinus: switch(x->type) {
                        case Integer:   result->intVal=x->intVal-y->intVal;break
;
                        case Real:              result->realVal=x->realVal-y->re
alVal;break;
                        case String:
#ifdef DEBUG_MSG
                                ScanError("ParseDoOp: Minus with String not supported\n");
#endif
                                result=NULL;
                                break;
                }break;
                case ScanTimes: switch(x->type) {
                        case Integer:   result->intVal=x->intVal*y->intVal;break
;
                        case Real:              result->realVal=x->realVal*y->re
alVal;break;
                        case String:
#ifdef DEBUG_MSG
                                ScanError("ParseDoOp: Times with String not supported\n");
#endif
                                result=NULL;
                                break;
                }break;
                case ScanDiv: switch(x->type) {
                        case Integer:   result->intVal=x->intVal/y->intVal;break
;
                        case Real:              result->realVal=x->realVal/y->re
alVal;break;
                        case String:
#ifdef DEBUG_MSG
                                ScanError("ParseDoOp: Divide with String not supported\n");
#endif
                                result=NULL;
                                break;
                }break;
                case ScanExpon: switch(x->type) {
                        case Integer:   result->intVal=(int)pow(x->intVal, y->in
tVal);break;
                        case Real:              result->realVal=(float)pow(x->re
alVal, y->realVal);break;
                        case String:
#ifdef DEBUG_MSG
                                ScanError("ParseDoOp: Exponentiation with String not supported\
n");
#endif
                                result=NULL;
                                break;
                }break;
                case ScanAND: switch(x->type) {
                        case Integer:   result->intVal=x->intVal&&y->intVal;brea
k;
                        case Real:              result->realVal=(float)((int)x->
realVal&&(int)y->realVal);break;
                        case String:
```

```
#ifdef DEBUG_MSG
                                ScanError("ParseDoOp: Logical AND with String not supported\n
");
#endif
                                result=NULL;
                                break;
                }break;
                case ScanOR: switch(x->type) {
                        case Integer:   result->intVal=x->intVal||y->intVal;brea
k;
                        case Real:              result->realVal=(float)((int)x->
realVal||(int)y->realVal);break;
                        case String:
#ifdef DEBUG_MSG
                                ScanError("ParseDoOp: Logical OR with String not supported\n"
);
#endif
                                result=NULL;
                                break;
                }break;
                case ScanEqual: switch(x->type) {
                        case Integer:   result->intVal=x->intVal==y->intVal;brea
k;
                        case Real:              result->realVal=(float)(x->realV
al==y->realVal);break;
                        case String:    VarNewStr(result, 1);
                                                if(strcmp(x->str, x->str
)==0) result->str[0]='1';
                                                else result->str[0]='0';
                                                result->str[1]='\0';
                                break;
                }break;
                case ScanGreaterThan: switch(x->type) {
                        case Integer:   result->intVal=x->intVal>y->intVal;break
;
                        case Real:              result->realVal=(float)(x->realV
al>y->realVal);break;
                        case String:    VarNewStr(result, 1);
                                                if(strcmp(x->str, x->str
)>0) result->str[0]='1';
                                                else result->str[1]='0';
                                                result->str[0]='\0';
                                break;
                }break;
                case ScanGreaterThanEqual: switch(x->type) {
                        case Integer:   result->intVal=x->intVal>=y->intVal;brea
k;
                        case Real:              result->realVal=(float)(x->realV
al>=y->realVal);break;
                        case String:    VarNewStr(result, 1);
                                                if(strcmp(x->str, x->str
)>=0) result->str[0]='1';
                                                else result->str[1]='0';
                                                result->str[0]='\0';
                                break;
                }break;
                case ScanLessThan: switch(x->type) {
                        case Integer:   result->intVal=x->intVal<y->intVal;break
;
                        case Real:              result->realVal=(float)(x->realV
al<y->realVal);break;
                        case String:    VarNewStr(result, 1);
                                                if(strcmp(x->str, x->str
)<0) result->str[0]='1';
                                                else result->str[1]='0';
                                                result->str[0]='\0';
                                break;
                }break;
                case ScanLessThanEqual: switch(x->type) {
```



```
                        case Integer:   result->intVal=x->intVal<=y->intVal;brea
k;
                        case Real:              result->realVal=(float)(x->realV
al<=y->realVal);break;
                        case String:    VarNewStr(result, 1);
                                                if(strcmp(x->str, x->str
)<=0) result->str[0]='1';
                                                else result->str[1]='0';
                                                result->str[0]='\0';
                                break;
                }break;
                case ScanNotEqual: switch(x->type) {
                        case Integer:   result->intVal=x->intVal!=y->intVal;brea
k;
                        case Real:              result->realVal=(float)(x->realV
al!=y->realVal);break;
                        case String:    VarNewStr(result, 1);
                                                if(strcmp(x->str, x->str
)!=0) result->str[0]='1';
                                                else result->str[1]='0';
                                                result->str[0]='\0';
                                break;
                }break;
                default:
#ifdef DEBUG_MSG
                        ScanError("ParseDoOp: Operation not supported (yet)\n");
#endif
                        break;
        }
        if(x->isTemp) VarDestroyVar(x);
        if(y->isTemp) VarDestroyVar(y);
        return result;
}

Var ParseDoOpUnary(Var x, enum ScanSymbol op) {
        Var result;
        char num;
        if(op==ScanINT) result=VarNewTemp(Integer);
        else if(op==ScanLEN || op==ScanRAND || op==ScanREAL) result=VarNewTemp(R
eal);
        else result = VarNewTemp(x->type);
        switch(op) {
                case ScanMinus: switch(x->type) {
                        case Integer:   result->intVal=-x->intVal;break;
                        case Real:              result->realVal=-x->realVal;brea
k;
                        default:
#ifdef DEBUG_MSG
                                ScanError("ParseDoOpUnary: Unary minus not implemented for t
ype\n");
#endif
                                break;
                }break;
                case ScanPlus: switch(x->type) {
                        case Integer:   result->intVal=-x->intVal;break;
                        case Real:              result->realVal=-x->realVal;brea
k;
                        default:
#ifdef DEBUG_MSG
                                ScanError("ParseDoOpUnary: Unary plus not implemented for typ
e\n");
#endif
                                break;
                }break;
                case ScanNOT: switch(x->type) {
                        case Integer:   result->intVal=!x->intVal;break;
                        case Real:              result->realVal=(float)(!((int)x
->realVal));break;
                        default:
```

```
#ifdef DEBUG_MSG
                                ScanError("ParseDoOpUnary: Unary NOT not implemented for ty
pe\n");
#endif
                                break;
                }break;
                case ScanINT: switch(x->type) {
                        case Integer:   result->intVal=x->intVal;break;
                        case Real:              result->intVal=(int)x->realVal;b
reak;
                        case String:    result->intVal=atoi(x->str);break;
                        default:
#ifdef DEBUG_MSG
                                ScanError("ParseDoOpUnary: Unary NOT not implemented for ty
pe\n");
#endif
                                break;
                }break;
                case ScanREAL: switch(x->type) {
                        case Integer:   result->realVal=(float)x->intVal;break;
                        case Real:              result->realVal=x->realVal;break
;
                        case String:    result->realVal=(float)atof(x->str);brea
k;
                        default:
#ifdef DEBUG_MSG
                                ScanError("ParseDoOpUnary: Unary NOT not implemented for ty
pe\n");
#endif
                                break;
                }break;
                case ScanLEN: switch(x->type) {
                        case String:    result->realVal=(float)strlen(x->str);br
eak;
                        default:
#ifdef DEBUG_MSG
                                ScanError("ParseDoOpUnary: String LEN not implemented for thi
s type.\n");
#endif
                                break;
                }break;
                case ScanRAND: switch(x->type) {
                        case Integer:   num=(char)x->intVal;break;
                        case Real:              num=(char)x->realVal;break;
                        default:
#ifdef DEBUG_MSG
                                ScanError("ParseDoOpUnary: RAND not implemented for this typ
e.\n");
#endif
                                break;
                }
            if(num<0)   srand(sndClock|(sndClock<<8));
                if(num==0)      result->realVal = lastRand;
                else            result->realVal = ((float)rand())/RAND_MAX;
                break;
                default:
#ifdef DEBUG_MSG
                        ScanError("ParseDoOpUnary: Only plus and minus are valid unary operations\
n");
#endif
                        break;

        }
        if(x->isTemp) VarDestroyVar(x);
        return result;
}

static void ParseREAD(Var x) {
        SWAP(s,sBackup);        // parse symbols
```



```
        SWAP(c,cBackup);        // scan characters
#ifndef ATMEL
        fTemp=fBackup;fBackup= f;f=fTemp;
#else
        SerialUseDATA();
#endif

        s = ScanGetSymbol();
        if(!inData) {
                do {
                        s=ScanGetSymbol();
                        if(s==ScanEOF) {

#ifdef ATMEL
        ScanReopenForATMEL();
#else
                ScanClose();
                ScanInit();
#endif

                        }
                }while(s!=ScanDATA);
                s = ScanGetSymbol();
        }
        switch(x->type) {
                case Real:
                        x->realVal=scanRealValue;
                        x->intVal=(int)scanRealValue;
                        s = ScanGetSymbol();
                        break;
                case Integer:
                        x->realVal=(float)scanIntValue;
                        x->intVal=scanIntValue;\
                        s = ScanGetSymbol();
                        break;
                case ScanStr:
                        if(x->str!=NULL) xfree(x->str);
                        VarNewStr(x, strlen(tempStr));
                        strcpy(x->str, tempStr);
                        s = ScanGetSymbol();
                        break;
        }
        switch(s) {
                case ScanComma: inData=1;break;
                case ScanEOF:
                case ScanEOL:
                case ScanNewStatement:
                        inData=0; break;
        }
        // switch inputs back if necessary
        SWAP(s,sBackup);
        SWAP(c,cBackup);
#ifndef ATMEL
        fTemp=fBackup;
        fBackup= f;
        f=fTemp;
#else
        SerialUseCode();
#endif
}

static Var ParseExpression();

static Var ParseGetVar() {
        Var x;
        enum VarType type;
        char i1=0,i2=0,i3=0,num=0;
        char ident[3];
        switch(s) {
```

```
                case ScanIntegerVar:    type=Integer;break;
                case ScanStringVar:     type=String; break;
                case ScanRealVar:               type=Real;   break;
                default:
#ifdef DEBUG_MSG
                        ScanError("Type Unknown\n");
#endif
                        break;
        }
        ident[0]=tempStr[0];ident[1]=tempStr[1];ident[2]='\0';
        s=ScanGetSymbol();
        if(s==ScanLeftParen) {
                num=1;
                s=ScanGetSymbol();
                x=ParseExpression();
                if(x->type!=Integer) x=ParseDoOpUnary(x, ScanINT);
                i1=x->intVal;
                if(x->isTemp) VarDestroyVar(x);
                if(s==ScanComma) {
                        num=2;
                        s=ScanGetSymbol();
                        x=ParseExpression();
                        if(x->type!=Integer) x=ParseDoOpUnary(x, ScanINT);
                        i2=x->intVal;
                        if(x->isTemp) VarDestroyVar(x);
                        if(s==ScanComma) {
                                num=3;
                                s=ScanGetSymbol();
                                x=ParseExpression();
                                if(x->type!=Integer) x=ParseDoOpUnary(x, ScanINT
);
                                i3=x->intVal;
                                if(x->isTemp) VarDestroyVar(x);
                        }
                }
#ifdef DEBUG_MSG
                if(s!=ScanRightParen) ScanError("Array identifier }, right paren expected\n");
#endif
                s=ScanGetSymbol();
                switch(type) {
                        case Integer:   type=IntegerArray;break;
                        case Real:              type=RealArray;   break;
                        case String:    type=StringArray; break;
                }
        }
        switch(type) {
                case String:            return VarGetVar(ident, num, i1, i2, i3,
 stringHead, stringTail, String);
                case Integer:   return VarGetVar(ident, num, i1, i2, i3, intHead
, intTail, Integer);
                case Real:              return VarGetVar(ident, num, i1, i2, i3,
 realHead, realTail, Real);
                case StringArray:               return VarGetVar(ident, num, i1,
 i2, i3, stringArrayHead, stringArrayTail, StringArray);
                case IntegerArray:      return VarGetVar(ident, num, i1, i2, i3,
 intArrayHead, intArrayTail, IntegerArray);
                case RealArray:         return VarGetVar(ident, num, i1, i2, i3,
 realArrayHead, realArrayTail, RealArray);
                default:
#ifdef DEBUG_MSG
                        ScanError("GetVar: Unexpected Variable Type\n");
#endif
                        return NULL;
        }
}

// Numbers, variables, and expressions in parentheses are "Factors"
static Var ParseFactor()
{
```



```
  Var x;

  switch (s) {
        case ScanRealVar:case ScanIntegerVar:case ScanStringVar:
                x=ParseGetVar();
                break;
    case ScanNum:
                if(scanIsFloat) {
                        x=VarNewTemp(Real);
                        x->realVal=scanRealValue;
                }
                else {
                        x=VarNewTemp(Integer);
                        x->intVal=scanIntValue;
                }
                s = ScanGetSymbol();
                break;
        case ScanStr:
                x=VarNewTemp(String);
                VarNewStr(x, strlen(tempStr));
                strcpy(x->str, tempStr);
                s = ScanGetSymbol();
                break;
    case ScanLeftParen:
      s = ScanGetSymbol();
      x = ParseExpression();
#ifdef DEBUG_MSG
         if (s != ScanRightParen) ScanError("Factor: ')' expected\n");
#endif
      s = ScanGetSymbol();
      break;
        case ScanINT:
                s=ScanGetSymbol();
#ifdef DEBUG_MSG
                if(s!=ScanLeftParen) ScanError("Factor: '(' expected\n");
#endif
                s=ScanGetSymbol();
                x = ParseExpression();
                x = ParseDoOpUnary(x, ScanINT);
#ifdef DEBUG_MSG
                if(s!=ScanRightParen) ScanError("Factor: ')' expected\n");
#endif
                s=ScanGetSymbol();
                break;
        case ScanLEN:
                s=ScanGetSymbol();
#ifdef DEBUG_MSG
                if(s!=ScanLeftParen) ScanError("Factor: '(' expected\n");
#endif
                s=ScanGetSymbol();
                x = ParseExpression();
                x = ParseDoOpUnary(x, ScanLEN);
#ifdef DEBUG_MSG
                if(s!=ScanRightParen) ScanError("Factor: ')' expected\n");
#endif
                s=ScanGetSymbol();
                break;
        case ScanRAND:
                s=ScanGetSymbol();
#ifdef DEBUG_MSG
                if(s!=ScanLeftParen) ScanError("Factor: '(' expected\n");
#endif
                s=ScanGetSymbol();
                x = ParseExpression();
                x = ParseDoOpUnary(x, ScanRAND);
#ifdef DEBUG_MSG
                if(s!=ScanRightParen) ScanError("Factor: ')' expected\n");
#endif
                s=ScanGetSymbol();
```

```
                break;
        default:
#ifdef DEBUG_MSG
                ScanError("ParseFactor: Factor expected\n");
#endif
                break;
  }
  return x;
}

// Handles the ^ exponential
static Var ParseExponTerm()
{
        enum ScanSymbol op;
        Var x, y;
        x = ParseFactor();
        while (s == ScanExpon) {
                op = s;
                s = ScanGetSymbol();
                y = ParseFactor();
                x = ParseDoOp(x, y, op);
        }
        return x;
}


static Var ParseTerm()
{
        enum ScanSymbol op;
        Var x, y;
        x = ParseExponTerm();
        while ((s == ScanTimes) || (s == ScanDiv)) {
                op = s;
                s = ScanGetSymbol();
                y = ParseExponTerm();
                x = ParseDoOp(x, y, op);
        }
        return x;
}


static Var ParseSimpleExpression()
{
  enum ScanSymbol op;
  Var x, y;

  if ((s == ScanPlus) || (s == ScanMinus)) {
    op = s;
    s = ScanGetSymbol();
    x = ParseTerm();
    x = ParseDoOpUnary(x, op);
  } else {
    x = ParseTerm();
  }
  while ((s == ScanPlus) || (s == ScanMinus)) {
    op = s;
    s = ScanGetSymbol();
    y = ParseTerm();
    x = ParseDoOp(x, y, op);
  }
  return x;
}


static Var ParseEqualityExpr()
{
  enum ScanSymbol op;
  Var x, y;
```



```
  x = ParseSimpleExpression();
  if ((s == ScanLessThan) || (s == ScanLessThanEqual) || (s == ScanEqual) || (s
== ScanNotEqual) || (s == ScanGreaterThanEqual) || (s == ScanGreaterThan)) {
    op = s;
    s = ScanGetSymbol();
    y = ParseSimpleExpression();
    x = ParseDoOp(x, y, op);
  }
  return x;
}

static Var ParseNOTExpr()
{
  enum ScanSymbol op;
  Var x;

  if (s == ScanNOT) {
    op = s;
    s = ScanGetSymbol();
    x = ParseEqualityExpr();
    x = ParseDoOpUnary(x, op);
  } else {
    x = ParseEqualityExpr();
  }
  return x;
}

static Var ParseANDExp()
{
  enum ScanSymbol op;
  Var x, y;

  x = ParseNOTExpr();
  if (s == ScanAND) {
    op = s;
    s = ScanGetSymbol();
    y = ParseNOTExpr();
    x = ParseDoOp(x, y, op);
  }
  while ((s == ScanAND)) {
    op = s;
    s = ScanGetSymbol();
    y = ParseNOTExpr();
    x = ParseDoOp(x, y, op);
  }
  return x;
}

static Var ParseExpression()
{
  enum ScanSymbol op;
  Var x, y;

  x = ParseANDExp();
  if (s == ScanOR) {
    op = s;
    s = ScanGetSymbol();
    y = ParseANDExp();
    x = ParseDoOp(x, y, op);
  }
  while ((s == ScanOR)) {
    op = s;
    s = ScanGetSymbol();
    y = ParseNOTExpr();
    x = ParseDoOp(x, y, op);
  }
  return x;
}
```

```
static void ParseExecuteLine();

static void ParseExecuteStatement() {
        Var exp, v;
        char * str;
        char cond;
        equalCount=0;
        switch(s) {
                // note fall through here
                case ScanHOME:
#ifdef ATMEL
                        ClearLCD();
#else
                        printf("\n\n\n\n\n\n\n\n\n\n\n\n");
#endif
                case ScanREM: case ScanDATA: case ScanDIM:
                        do{
                                s=ScanGetSymbol();
                        }while(s!=ScanNewStatement && s!=ScanEOL && s!=ScanEOF);

                        break;
                case ScanSND:
                        s=ScanGetSymbol();
                        if(s!=ScanLeftParen) ScanError("ScanSND: ( expected\n");
                        s=ScanGetSymbol();
                        exp=ParseExpression();
                        if(s!=ScanComma) ScanError("ScanSND: , expected\n");
                        s=ScanGetSymbol();
                        v=ParseExpression();
                        if(s!=ScanRightParen) ScanError("ScanSND: ) expected\n");
                        s=ScanGetSymbol();
                        exp = ParseDoOpUnary(exp, ScanINT);
                        v   = ParseDoOpUnary(v, ScanINT);
                        SndPlayNote(exp->intVal, v->intVal);
                        if(exp->isTemp) VarDestroyVar(exp);
                        if(v->isTemp) VarDestroyVar(v);
                        break;
                case ScanPRINT:
                        do {
                                s=ScanGetSymbol();
                                if(s==ScanNewStatement || s==ScanEOF || s==ScanE
OL) {
                                        sprintf(lcd_buffer, "\n");
                                        LCDPrintString(lcd_buffer);
                                        return;
                                }
                                exp = ParseExpression();
                                switch(exp->type) {
                                        case String:
                                                sprintf(lcd_buffer, "%s", exp->
str);
                                                LCDPrintString(lcd_buffer);
                                                break;
                                        case Integer:
                                                sprintf(lcd_buffer, "%d", exp->
intVal);
                                                LCDPrintString(lcd_buffer);
                                                break;
                                        case Real:
                                                sprintf(lcd_buffer, "%f", exp->r
ealVal);
                                                LCDPrintString(lcd_buffer);
                                                break;
                                        default:
#ifdef DEBUG_MSG
                                                ScanError("ParseExecuteStatement: Canno
t print this type\n");
#endif
                                                break;
```



```
                                }
                                if(exp->isTemp) VarDestroyVar(exp);
                        } while(s==ScanSemicolon);
#ifndef ATMEL
                        sprintf(lcd_buffer, "\n");
#endif
                        LCDPrintString(lcd_buffer);
#ifdef ATMEL
                        LCDNewLine();
#endif
                        break;
                case ScanINPUT:
                        str = InputGetString();
                        if(*str=='\0') ScanError("ERROR: You must enter input!\n");
                        str = strtok(str, ",");
                        do {
                                s=ScanGetSymbol();
                                switch(s) {
                                        case ScanRealVar: case ScanIntegerVar: c
ase ScanStringVar:
                                                v=ParseGetVar(); break;
                                        default:
#ifdef DEBUG_MSG
                                                ScanError("ParseExecuteStatement: Varia
ble Expected\n");
#endif
                                                break;
                                }
                                switch(v->type) {
                                        case String:    if(v->str!=NULL) xfree(v
->str);
                                                VarNewStr(v,strlen(str));strcpy(
v->str,str);break;
                                        case Integer:   v->intVal=atoi(str);
break;
                                        case Real:              v->realVal=(floa
t)atof(str);    break;
                                        default:
#ifdef DEBUG_MSG
                                                ScanError("ParseExecuteStatement: Canno
t assign this type\n");
#endif
                                                break;
                                }
                                str=strtok(NULL, ",");
                        } while(str!=NULL && s==ScanComma);
                        break;
                case ScanREAD:
                        s=ScanGetSymbol();
                        exp=ParseGetVar();
                        ParseREAD(exp);
                        while(s==ScanComma) {
                                s=ScanGetSymbol();
                                exp=ParseGetVar();
                                ParseREAD(exp);
                        }
                        break;
                case ScanGOTO:
                        s=ScanGetSymbol();
                        exp = ParseExpression();
                        switch(exp->type) {
                                case String:
#ifdef DEBUG_MSG
                                        ScanError("ParseExecuteStatement: GOTO targets can
not be of type String\n");
#endif
                                        break;
                                case Integer:   newPC=exp->intVal; break;
                                case Real:              newPC=(int)exp->realVal;
```

```
break;
                                default:
#ifdef DEBUG_MSG
                                        ScanError("ParseExecuteStatement: Not a valid type f
or GOTO targets\n");
#endif
                                        break;
                        }
                        if(exp->isTemp) VarDestroyVar(exp);
                        break;
                case ScanIF:
                        s=ScanGetSymbol();
                        exp = ParseExpression();
                        switch(exp->type) {
                                case String:
#ifdef DEBUG_MSG
                                        ScanError("ParseExecuteStatement: Strings not valid i
n Conditions\n");
#endif
                                        break;
                                case Integer:   cond=0!=exp->intVal; break;
                                case Real:              cond=0!=exp->realVal;bre
ak;
                                default:
#ifdef DEBUG_MSG
                                        ScanError("ParseExecuteStatement: Not a valid type f
or GOTO targets\n");
#endif
                                        break;
                        }
                        if(s==ScanTHEN) s=ScanGetSymbol(); // THEN is optional i
n many cases
                        if(cond) {
                                ParseExecuteLine(); // if true parse remainder o
f line
                        }
                        else {
                                do{
                                        s=ScanGetSymbol();
                                }while(s!=ScanEOL && s!=ScanEOF);
                        }
                        if(exp->isTemp) VarDestroyVar(exp);
                        break;
                case ScanFOR:
                        s=ScanGetSymbol();
                        forLoops[numForLoops].var=ParseGetVar();
#ifdef DEBUG_MSG
                        if(forLoops[numForLoops].var->type!=Real) ScanError("Loo
p variable must be a real\n");
#endif
                        forLoops[numForLoops].lineNum=curPC+1;
#ifdef DEBUG_MSG
                        if(s!=ScanEqual) ScanError("In FOR statement, initialization (=) expec
ted\n");
#endif
                        s=ScanGetSymbol();
                        exp=ParseExpression(); // assignement of loop variable
                        switch(exp->type) {
                                case Real:              forLoops[numForLoops].va
r->realVal=exp->realVal;break;
                                case Integer:   forLoops[numForLoops].var->realV
al=(float)exp->intVal;break;
                                default:
#ifdef DEBUG_MSG
                                        ScanError("Loop variable inits must be reals or integer
s\n");
#endif
                                        break;
                        }
```



```
                        if(exp->isTemp) VarDestroyVar(exp);
#ifdef DEBUG_MSG
                        if(s!=ScanTO) ScanError("In FOR statement must have a TO.\n");
#endif
                        s=ScanGetSymbol();
                        exp=ParseExpression(); // termination value of loop
                        switch(exp->type) {
                                case Real:              forLoops[numForLoops].fi
nal = exp->realVal;break;
                                case Integer:   forLoops[numForLoops].final = (f
loat)exp->intVal;break;
                                default:
#ifdef DEBUG_MSG
                                        ScanError("Loop termination value must be a real or in
teger\n");
#endif
                                        break;
                        }
                        if(exp->isTemp) VarDestroyVar(exp);
                        if(s==ScanSTEP) {
                                s=ScanGetSymbol();
                                exp=ParseExpression();
                                switch(exp->type) {
                                        case Real:              forLoops[numForL
oops].step = exp->realVal;break;
                                        case Integer:   forLoops[numForLoops].st
ep = (float)exp->intVal;break;
                                        default:
#ifdef DEBUG_MSG
                                                ScanError("STEP must be a real or integer.
\n");
#endif
                                                break;
                                }
                                if(exp->isTemp) VarDestroyVar(exp);
                        }
                        else {
                                forLoops[numForLoops].step = 1;
                        }
                        numForLoops++;
                        break;
                case ScanNEXT:
                        s=ScanGetSymbol();
                        if(s!=ScanEOL && s!=ScanEOF) {
                                exp=ParseExpression();
                                // if loop variable NEXT is early, kill higher s
tacked loops
                                for(;numForLoops>0 && exp!=forLoops[numForLoops-
1].var;numForLoops--);
#ifdef DEBUG_MSG
                                if(numForLoops==0) ScanError("Not a valid loop variable.\
n");
#endif
                        }
#ifdef DEBUG_MSG
                        if(numForLoops<=0) ScanError("NEXT does not have a matching FOR.
\n");
#endif
                        // increment loop variable
                        forLoops[numForLoops-1].var->realVal+=forLoops[numForLoo
ps-1].step;
                        if((forLoops[numForLoops-1].step>=0 && forLoops[numForLo
ops-1].var->realVal<=forLoops[numForLoops-1].final) || (forLoops[numForLoops-1].
step<0 && forLoops[numForLoops-1].var->realVal>=forLoops[numForLoops-1].final))
{
                                newPC=forLoops[numForLoops-1].lineNum;
                                s=ScanEOL;
                        }
                        else {
```

```
                                numForLoops--;

                        }
                        break;
                case ScanIntegerVar:
                        v=ParseGetVar(); // note: consumes next symbol
#ifdef DEBUG_MSG
                        if(s!=ScanEqual) ScanError("ParseExecuteStatement: Is this an assign
ment, I expected an '='\n");
#endif
                        s=ScanGetSymbol();
                        exp = ParseExpression();
                        switch(exp->type) {
                                case String:    v->intVal=atoi(v->str);break;
                                case Integer:   v->intVal=exp->intVal;
break;
                                case Real:              v->intVal=(int)exp->real
Val;break;
                                default:
#ifdef DEBUG_MSG
                                        ScanError("ParseExecuteStatement: Cannot assign this
 type\n");
#endif
                                        break;
                        }
                        if(exp->isTemp) VarDestroyVar(exp);
                        break;

                case ScanRealVar:
                        v=ParseGetVar();
#ifdef DEBUG_MSG
                        if(s!=ScanEqual) ScanError("ParseExecuteStatement: Is this an assign
ment, I expected an '='\n");
#endif
                        s=ScanGetSymbol();
                        exp = ParseExpression();
                        switch(exp->type) {
                                case String:    v->realVal=(float)atof(v->str);b
reak;
                                case Integer:   v->realVal=(float)exp->intVal;
break;
                                case Real:              v->realVal=exp->realVal;
break;
                                default:
#ifdef DEBUG_MSG
                                        ScanError("ParseExecuteStatement: Cannot assign this
 type\n");
#endif
                                        break;
                        }
                        if(exp->isTemp) VarDestroyVar(exp);
                        break;
                case ScanStringVar:
                        v=ParseGetVar();
#ifdef DEBUG_MSG
                        if(s!=ScanEqual) ScanError("ParseExecuteStatement: Is this an assign
ment, I expected an '='\n");
#endif
                        s=ScanGetSymbol();
                        exp = ParseExpression();
                        switch(exp->type) {
                                case String:    if(v->str!=NULL) xfree(v->str);
                                        VarNewStr(v,strlen(exp->str));strcpy(v->
str, exp->str);   break;
                                case Integer:   v->intVal=exp->intVal; break;
                                case Real:              v->realVal=exp->realVal;
break;
                                default:
#ifdef DEBUG_MSG
```



```
                                        ScanError("ParseExecuteStatement: Cannot assign this
 type\n");
#endif
                                        break;
                        }
                        if(exp->isTemp) VarDestroyVar(exp);
                        break;
        }
}

static void ParseExecuteLine() {
        ParseExecuteStatement();
        while(s==ScanNewStatement) {
                s=ScanGetSymbol();
                ParseExecuteStatement();
        }
}

void main() {
        newPC=0;
        curPC=0;
#ifdef ATMEL
        InitLCD();
#endif
#ifdef ATMEL
        initialize_uart();
        SndInit();
#endif
        VarInit();
        sprintf(lcd_buffer, "Enter a BASIC file:");
        LCDPrintString(lcd_buffer);
        ScanInit();
        s=ScanGetSymbol();
        SWAP(s,sBackup);        // parse symbols
        SWAP(c,cBackup);        // scan characters
#ifndef ATMEL
        sndClock = (unsigned)time( NULL );
#endif
        srand(sndClock|(sndClock<<8));
#ifndef ATMEL
        fTemp=fBackup;fBackup= f;f=fTemp;
#endif

        do {
#ifdef ATMEL
        ScanReopenForATMEL();
#else
                ScanInit();
#endif
                if(ScanGetSymbol()!=ScanNum) {
#ifdef DEBUG_MSG
                 ScanError("Line Number Expected\n");
#endif
                }
#ifndef ATMEL
                while(scanIntValue<newPC) ScanSkipLine();
#endif
                curPC=scanIntValue;
                s=ScanGetSymbol();
                newPC=curPC+1; // new line as long as we don't do a GOTO/GOTOSUB

                ParseExecuteLine();
#ifndef ATMEL
                ScanClose();
#endif
        }while(s==ScanEOL);

}
```