```
/* eval3.c   from nut32
     5 Jan 2005
    This is a virtual machine to execute RZ parse-tree
    modify to use with rz3     30 Oct 2010
 */

//  this is 95% correct, every instructions work except "return"
//  16 Nov 2010

#include "compile.h"

#define MAXMEM   20000
#define STKBASE  1000

#define head(e)         car(e)
#define tail(e)         cdr(e)
#define arg1(e)         car(e)
#define arg2(e)         item2(e)
#define arg3(e)         item3(e)

int M[MAXMEM];
int sp = STKBASE;
int fp = STKBASE+1;

int eval(int e){
    int op, v, ty;
    int idx, fs, e1, e2, a;

    if (cnt++ > 1000000) seterror("infinite loop\n");

    if( e == NIL) return NIL;
    if( isatom(e) ) return evalatom(e);

    e1 = tail(e);             // argument list (arg1 arg2 ...)
    e = head(e);              // operator
    if(head(e) != OPER) seterror("expect operator");
    op = tail(e);
    switch(op){
    case tkIF:
        if( eval(arg1(e1)) )
            return eval(arg2(e1));
        return NIL;
    case tkELSE:
        if( eval(arg1(e1)) )
            return eval(arg2(e1));
        else
            return eval(arg3(e1));
    case tkWHILE:
        while( eval(arg1(e1)) )
            v = eval(arg2(e1));
        return v;
    case tkDO:
        while( e1 != NIL ){
            v = eval(head(e1));
            e1 = tail(e1);
        }
        return v;             // return the last eval

    case tkPLUS: return eval(arg1(e1)) + eval(arg2(e1));
    case tkMINUS: return eval(arg1(e1)) - eval(arg2(e1));
    case tkSTAR: return eval(arg1(e1)) * eval(arg2(e1));
```

```
    case tkSLASH: return eval(arg1(e1)) / eval(arg2(e1));
    case tkANDAND: return eval(arg1(e1)) && eval(arg2(e1));
    case tkOROR: return eval(arg1(e1)) || eval(arg2(e1));
    case tkEQEQ: return eval(arg1(e1)) == eval(arg2(e1)) ? 1 : 0;
    case tkNE: return eval(arg1(e1)) != eval(arg2(e1)) ? 1 : 0;
    case tkLT: return eval(arg1(e1)) < eval(arg2(e1)) ? 1 : 0;
    case tkLE: return eval(arg1(e1)) <= eval(arg2(e1)) ? 1 : 0;
    case tkGT: return eval(arg1(e1)) > eval(arg2(e1)) ? 1 : 0;
    case tkGE: return eval(arg1(e1)) >= eval(arg2(e1)) ? 1 : 0;
    case tkUNOT: return eval(arg1(e1)) ? 0 : 1;
    case tkUMINUS: return -eval(arg1(e1));
    case tkCALL:
        push(fp);                   // save fp'
        idx = tail(head(e1)); // type is FUNCTION
        e1 = tail(e1);
        while( e1 != NIL ){         // eval all arg
            v = eval(head(e1));
            push(v);
            e1 = tail(e1);
        }
        return eval(getRef(idx));   // eval body of fun
    case tkRETURN:
        if( e1 != NIL ) v = eval(arg1(e1));
        else v = NIL;
//          sp = fp-1;
//          fp = M[fp];             // delete stack frame
//          contflag = 0;
//          printf("ret %d\n",v);
        return v;

    case tkFUN:
//       lo           stack frame
//
//     fp'  <- fp
//     lv1   .. pv1
//     lv2
//     ...
//     lvn  <- sp
//
//     hi

        idx = tail(arg1(e1)); // function
        a = getArg(idx);            // arity
        fs = getFs(idx);            // frame size
        fp = sp-a;
        sp = fp+fs;
        v = eval(arg2(e1));         // eval body of fun
        sp = fp-1;                        // delete frame
        fp = M[fp];
        return v;

    case tkDEREF:             // RHS *
        a = effAds(head(e1));
        return M[M[a]];
    case tkADS:                       // RHS &
        return effAds(head(e1));
    case tkVEC:
        a = effAds(arg1(e1)) + eval(arg2(e1));
        return M[a];
    case tkEQ:                        // assignment
        a = effAds(arg1(e1));       // LHS effective address
```

```
        v = eval(arg2(e1));         // do RHS
        if( isderef(arg1(e1)))
            M[M[a]] = v;
        else
            M[a] = v;
        return v;

    case tkPRINT:
        while( e1 != NIL ){
            e2 = head(e1);
            if(isatom(e2)){
                ty = head(e2);
                v = tail(e2);
                switch(ty){
                case NUM: printf("%d",v); break;
                case STRING: printString(&strbuf[v]); break;
                case GNAME: printf("%d", M[getRef(v)]); break;
                case LNAME: printf("%d", M[fp+v]); break;
                }
            }else
                printf("%d",eval(e2));
            e1 = tail(e1);
        }
        return NIL;
    default:
        seterror("unknown op\n");
    }
    return NIL;
}


void push(int a){
    sp++;
    if(sp >= MAXMEM ) seterror("stack overflow");
    M[sp] = a;
}

int cnt = 0;

int evalatom(int a){
    int ty, val;
    ty = head(a);
    val = tail(a);
    switch(ty){
    case NUM: return val;
    case STRING: return val;
    case GNAME: return M[getRef(val)];          // RHS
    case LNAME: return M[fp+val];               // RHS
    }
    return NIL;
}

// e is [*] exp
int isderef(int e){
    return (head(head(e)) == OPER) && (tail(head(e)) == tkDEREF);
}
// e is [&] exp
int isads(int e){
    return (head(head(e)) == OPER) && (tail(head(e)) == tkAND);
}
// e is {vec name idx, name}
```

```
int isvec(int e){
    return (head(head(e)) == OPER) && (tail(head(e)) == tkVEC);
}

// effective address  e is [*] {vec name idx, name}
int effAds(int e){
    int nm, ref, a;
    if( isderef(e) )
        e = arg2(e);                            // strip "*"
    if( isvec(e) ){
        nm = arg2(e);
        a = eval(arg3(e));                      // idx
    }else{
        nm = e;
        a = 0;
    }
    ref = tail(nm);
    a += head(nm) == GNAME ? getRef(ref) : fp+ref;
    return a;
}

// handle "\n"
void printString(char *s){
    char c;
    while( (c = *s) != 0 ){
        if( c == 92 && *(s+1) == 110){    // "\n"
            printf("\n");
            s++;
        }else
            printf("%c",c);
        s++;
    }
}

// end
```