Micro Compiler Project

Micro language PL/Micro is defined as follows:

Program

S

E

T

F

This language allows a sequence of assignment statements, each followed by ‘;’, and the whole program is followed by ‘!’.

An example program

\[ a=2; b=3; c=4; a=(b+c)*a; b=a*(c+b); ! \]

Documentation: Assign 2 to a. Assign 3 to b. Assign 4 to c. Store the value of (b+c)*a to a. Store the value of a*(c+b) to b.
This program is just an example, and has no special meaning.
Parser for PL/M

char current; int i; char x[100]; char c, left;

error()
{ printf("error at %d
", i); }
init()
{ 
i=1; do{scanf("%c", &c); x[i]=c; i++;}while(x[i-1]!="");
}
getsym(){ i++; current=x[i]; }

S()
{ 
  if ((current>=97)&&(current<=122)) { 
    left=current; getsym(); 
  } else error();
  if(current==='=') getsym(); else error();
  E();
  if(current!=;') error();
}

E()
{ 
  T(); while(current==='+'){ getsym(); T();}
  if((current!=')')&&((current!=;')&&!((current==='+')))) error();
}

T()
{ 
  F(); while(current==='*'){getsym(); F();}
  if((current!=')')&&((current!=;')&&!((current==='+')))) error();
}

F()
{ 
  if(current==='('){getsym(); E(); if(current===')') getsym(); else error();
  } else if((current>=97)&&((current<=122)) { 
    getsym();
  } else error( );
}

main(){
  init(); i=0; getsym();
  do{
    S(); getsym();
  }while(current!=;');
  printf("no errors\n");
}
Memory Management

Storage for variables, \( m \)

Snapshot at (\#)  
\[ \begin{array}{c|c} 
| & \\ 
- & \\
| \hline 
a & 14 \\
| b & 3 \\
| c & 4 \\
| \end{array} \]

Storage for object code, \( \text{code} \)  

\[
\begin{align*}
\text{lit} & \quad 2 \\
sto & \quad a \\
\text{lit} & \quad 3 \\
sto & \quad b \\
\text{lit} & \quad 4 \\
sto & \quad c \\
lod & \quad b \\
lod & \quad c \\
add & \quad b+c \\
lod & \quad a \\
\text{mul} & \quad (b+c)*a \\
sto & \quad a=(b+c)*a; \\
lod & \quad a \\
lod & \quad c \\
\end{align*}
\]

\[ i \longrightarrow \]

Storage for run-time stack, \( s \)  

Execution result at \( \text{sto} \)

Snapshot at (\#)  
\[ \begin{array}{c|c|c} 
| & & \\ 
- & & \\
| \hline 
14 & 2 \\
4 & 3 \\
4 & 14 \\
\end{array} \]

K \[ \longrightarrow \]

3 & 98

Machine instructions of PL/M machine

- **LOD a**: Increase the stack pointer. Load the value of a into the stack top.
- **STO a**: Store the stack top into a. Decrease the stack pointer.
- **LIT n**: Increase the stack pointer. Load the number n into the stack top.
- **ADD**: Add the two stack top elements. Store the result into the next stack top.
- **MUL**: Multiply the two stack top elements. Store the result into the next stack top.

For **ADD** and **MUL**, we decrease the stack pointer.

Those instructions are executed by the interpreter in the source code.
Documentation of Compiler/Interpreter

Compiler generates the object code for a source program, and interpreter executes the object code based on the memory management described in the previous page.

Variables
array x: container of user's program
m: storage for user defined variables
s: runtime stack
code: container of object code
adr: used for operand. If operand is variable, address is given. If number, its value.

Functions
Error( ): error handler
init( ) initializes one line of user's program into array x
id( ) gives address of variable
getsym( ) reads the next symbol into current.
S( ) handles statement. After processing the left-hand side, if the next symbol is “=”, read the next symbol, and process an expression.
E( ) handles expression. After processing a term, while the next symbol is “+”, read the next symbol, and process a term.
T( ) handles term. After processing a factor, while the next symbol is “*”, read the next symbol, and process a factor.
F( ) handles a factor. If the current symbol is “(”, read the next symbol, and process an expression. After that, if the current symbol is “)”, read the next symbol, and exit. If the current is a variable at the above point (#), read the next symbol, and exit. If none of those at (#), error( ).
Interpret( ) executes user's object code. Instruction counter is given by variable “i”.
This executes code[i] for i=1, ..., j-1.

Source Program of Compiler/Interpreter

```c
char current; int i,j,k; int code[100], adr[100], s[100], m[30];
char x[100]; char c, left;
int add=1, mul=2, lod=3, sto=4, lit=5;

error()
{ printf("error at %d\n", i); }

loc(char ch){
    if(ch>=97)return ch-96; /* a ==> location 1, b ==> location 2, etc */
    if((ch>=48)&&(ch<=57))return ch-48; /* 1 ==> value 1, 2 ==> value 2, etc */
}

init()
{i=1; do{ scanf("%c", &c); x[i]=c; i++;}while(x[i-1]!='$');}

gsym() { i++; current=x[i]; }
```

S() {
    if((current>=97)&&(current<=122)) {
        left=current; getsym();
    } else error();
    if(current=='\n') getsym(); else error();
    if(current!=='\n') error();
    code[j]=sto; adr[j]=loc(left);
    printf("sto %c %d\n", left, adr[j]);
    j++;
}

E() {
    T();
    while(current=='\n') (getsym(); T(); printf("add\n"); code[j]=add; j++;
    if((current!=='\n')&&!((current!=='\n'))) error();
}

T() {
    F();
    while(current=='\n') (getsym(); F(); printf("mul\n"); code[j]=2; j++;
    if((current!=='\n')&&!((current!=='\n')&&!((current!=='\n'))) error();
}

F() {
    if(current=='\n') (getsym(); E(); if(current=='\n')) getsym(); else error();
    else if((current>=97)&&(current<=122)) {
        printf("lod %c\n", current);
        code[j]=lod; adr[j]=loc(current); j++; getsym();
    } else if((current>=48)&&(current<=57)) {
        printf("lit %c\n", current);
        code[j]=lit; adr[j]=loc(current); j++; getsym();
    } else error();
}

interpret() {
    k=0;
    for(i=1;i<j;i++) {
        if(code[i]==1) s[k-1]=s[k-1]+s[k]; k--; /* add */
        if(code[i]==2) s[k-1]=s[k-1]*s[k]; k--; /* mul */
        if(code[i]==3) (k++; s[k]=m[adr[i]]); /* lod x for some x=adr[i] */
        if(code[i]==4) (m[adr[i]]=s[k]; printf("%d\n", s[k]); k--; /* sto x */
        if(code[i]==5) (k++; s[k]=adr[i]); /* lit n for some n=0, ..., 9 */
    }
}

main() {
    j=1;
    init(); i=0; getsym();
    do{
        S(); getsym();
    } while(current!=='\n');
    printf("no errors\n");
    for(i=1;i<j;i++) printf("%d %d %d \n", code[i], adr[i]);
    interpret();
}

What more? (1) "-" and "+" can be easily implemented by enhancing E( ) and T( ).
(2) "if B then S" statement can be implemented by enhancing the PL/M machine with
"jump on condition" instruction jpc. After S is processed the jump address will be fixed up.