```
100h:  /*load point for compiler*/

    /*  ***************************************************************  */
    /*  **********          system literals          **********  */
    /*  ***************************************************************  */

declare false          literally '0',
        true           literally '1',
        lit            literally 'literally',
        bdos  lit '5h',      /* entry point to disk operating system */
        startbdos address initial(6h), /*addr of ptr to top of bdos */
        max based startbdos address,
        boot           lit '0',          /* exit to return to operating system */
        pstacksize     lit '48',               /* stack sizes for parser */
        intrecsize     lit  '128',
        dcl            lit 'declare',
        proc           lit         'procedure',
        fileeof        lit  '1',
        rfile          lit  '20',
        identsize      lit  '32',
        addr           lit         'address',
        forever        lit 'while true',
        varcsize       lit '100',
        indexsize      lit 'address',
        statesize      lit 'address',
        maxoncount     lit '23',
        cr             lit '13',
        lf             lit '0ah',
        stringdelim    lit  '22h',
        questionmark   lit  '3fh',
        tab            lit  '09h',
        colin          lit  '3ah',
        comment        lit  '0',
        conbuffsize    lit  '82',
        eolchar        lit '0dh',
        hashtblsize    lit  '64',
        sourcerecsize  lit  '128',
        hashmask       lit  '63',
        contchar       lit  '5ch',
        eoffiller      lit 'lah',
        percent        lit '25h';


declare maxrno    literally '132',/* max read count */
        maxlno    literally '190',/* max look count */
        maxpno    literally '190',/* max push count */
        maxsno    literally '373',/* max state count */
        starts    literally '1',/* start state */
        prodno    literally '183',/* number of productions */
        semic     literally '8',/* semicolon */
        colonc    literally '13',/* colon */
        doc       literally '18',/* do */
        eofc      literally '24',/* eof */
        endc      literally '26',/* end */
        string    literally '49',/* string */
        decimal   literally '52',/* decimal */
        integerc  literally '33',/* integer */
        procc     literally '54',/* procedure */
        identifier lit '55',/*identifier*/
        termno    literally '55';/* terminal count */

declare sbloc              address initial(80h),
        sourcebuff based sbloc(sourcerecsize) byte,
        sourceptr          byte    initial(sourcerecsize),
        buffptr            byte    initial(255),
        errorcount         address initial 0,
        linebuff(conbuffsize) byte,
        lineptr            byte    initial(0),
```

```
        lineno              address,
        pass1               byte    initial(true),
        pass2               byte    initial(false),
        nointfile           byte    initial(false),
        rfcbaddr            address initial(5ch),
        rfcb  based         rfcbaddr(33) byte,
        wfcb(33)            byte initial(0,'        ','ain',0,0,0,0),
        cursourcerecsize    byte initial(sourcerecsize),
        nolook              byte,
        production          byte,
        arr$loc(5)          address,
        arr$num             byte,
        sub$proc$loc        address,
        sub$proc$var$num    byte,
        arr$dim             byte,
        diskoutbuff(intrecsize) byte;

   /* the following global variables are used by the scanner   */

declare token       byte,       /* type of token just scanned */
        hashcode    byte,       /* has value of current token */
        nextchar    byte,       /* current character from getchar */
        accum(identsize)    byte,    /* holds current token */
        cont        byte;       /* indicates accum was full, still more */


    /*****************************************************/
    /*          symbol table global variables            */
    /*****************************************************/

declare base        address, /* base of current entry */
        hashtable(hashtblsize) address,
        sbtbltop    address, /*current top of symbol table*/
        sbtbl       address,
        ptr based base byte, /*first byte of entry */
        aptraddr    address, /*utility variable to access table*/
        addrptr based aptraddr address,
        byteptr based aptraddr byte,
        printname   address, /*set prior to lookup or enter*/
        symhash     byte,
        prev$blk$level(12) byte,
        prev$index  byte initial(255),
        step$flag   byte,
        blk$cnt     byte initial(0),
        blk$level   byte initial(1);

declare read1 data(0,39,12,15,53,55,2,49,52,53,55,5,8,8,19,20,26,27,31
    ,34,35,39,40,42,43,48,53,55,19,20,26,27,31,34,35,39,40,42,43,48,53
    ,55,55,53,55,55,15,53,55,23,2,3,9,20,28,49,52,53,55,55,2,3,9,20,49
    ,52,53,55,49,55,2,3,9,20,49,52,53,55,2,49,55,55,2,49,55,2,2,2,2,49,55
    ,11,14,54,51,55,13,7,4,55,55,2,13,14,2,14,14,2,8,11,7,11,7,11,11,2,8
    ,4,6,10,24,53,55,16,7,11,2,7,11,17,7,11,7,49,55,11,19,20,27,31,34,35
    ,39,40,42,43,48,53,55,32,55,14,8,19,20,27,31,33,34,35,39,40,42,43,44
    ,45,47,48,50,53,54,55,4,11,1,12,15,21,7,11,7,11,4,11,1,7,12,15,32,7
    ,13,36,2,3,9,20,29,30,49,52,53,55,8,8,8,2,2,2,25,39,49,52,53,18,11
    ,55,33,44,45,47,50,54,7,11,55,7,11,8,11,41,55,38,50,51,46,22,37,7,22
    ,19,27,31,34,35,39,40,42,43,48,55,7,3,5,9,0);

declare look1 data(0,12,15,0,15,0,2,0,2,0,11,0,14,0,8,17,32,0,2,14,0,14
    ,0,11,0,14,0,11,0,11,0,2,14,17,0,6,10,0,6,10,0,6,10,0,6,10,0,6,10,0
    ,6,10,0,16,0,16,0,16,0,17,0,11,0,25,0,25,0,11,0,11,0,33,44,45,47,50
    ,54,0,11,0,55,0,46,0,33,44,45,47,50,54,0,32,0,22,0,46,0,3,5,9,0);

declare apply1 data(0,0,1,0,0,0,0,0,131,145,0,0,149,0,0,0,0,44 0,14,15
    ,40,93,0,37,93,143,0,37,143,0,0,0,27,150,0,3,4,32,95,0,0,3,4,6,22,24
    ,35,36,39,86,94,95,112,122,130,134,135,137,0,0,7,8,10,16,17,0,11,18
    ,0,26,0,124,0,3,4,5,14,15,27,32,37,40,93,95,96,101,143,150,0,0,0,0
    ,63,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,42,46,0,61,0,0,0,0,0,0,14,15,37,40
    ,93,101,143,0,0,3,79,0,39,0,31,33,0,0,0,0,0,0,0,0,0,0,0,0,0,125,0,0,0
    ,0,0,0,0,0,0,0,0,0,0,14,15,27,37,40,93,101,143,150,0,0,0,52,0,0,0,0,0
    );

declare read2(254) address initial
```

```
          (0,195,331,332,54,161,4,264,263,262,160,10,353,358,28
     ,29,196,32,36,290,38,195,41,336,42,46,53,159,28,29,371,32,36,290,38
     ,195,41,336,42,46,53,159,344,310,309,368,333,55,70,289,3,7,16,29,33
     ,264,263,262,160,161,3,7,16,29,264,263,262,160,312,313,4,7,16,29,264
     ,263,262,160,292,157,162,343,298,157,162,5,208,300,157,162,20,25,57
     ,52,69,229,218,275,62,367,363,228,285,363,285,285,360,357,279,12,356
     ,13,361,211,355,352,274,11,18,191,288,287,26,207,209,6,311,22,243
     ,291,294,217,158,163,269,28,29,32,36,290,38,195,41,336,42,46,53,159
     ,37,164,281,14,28,29,32,36,268,290,38,195,41,336,42,155,156,214,46
     ,49,53,56,159,280,282,153,154,327,370,295,302,362,364,284,286,153
     ,261,154,327,318,261,24,39,4,7,16,29,34,35,264,263,262,160,193,359
     ,354,293,299,301,31,40,264,263,262,27,21,65,268,155,156,214,49,56
     ,297,304,66,296,303,13,19,341,163,278,50,51,45,30,317,325,30,28,32
     ,36,290,38,195,41,336,42,46,166,260,8,9,17,0);

declare look2(101) address initial
               (0,2,2,328,23,329,43,212,44,215,47,307,48,271,240,265
     ,240,58,59,59,265,60,265,61,305,63,273,64,277,67,206,68,68,265,240
     ,71,71,246,72,72,251,73,73,250,74,74,247,75,75,248,76,76,249,80,252
     ,81,253,82,254,88,257,92,267,120,319,121,320,127,366,128,365,131,131
     ,131,131,131,131,351,139,238,210,142,144,202,145,145,145,145,145,145
     ,349,227,219,147,333,149,290,152,152,152,244);

declare apply2(177) address initial
                    (0,6,77,239,101,194,192,190,115,116,114,189,201,203
     ,124,141,184,216,213,198,373,372,224,197,222,187,223,219,225,226,220
     ,97,143,342,345,221,151,151,338,242,231,95,108,110,314,315,283,316
     ,104,339,326,109,241,105,106,245,107,340,111,103,190,168,170,172,169
     ,171,167,174,175,173,256,255,83,258,176,176,90,87,87,87,87,87,87,87
     ,176,89,87,87,87,257,199,91,177,272,270,185,99,98,276,137,192,266
     ,134,237,78,234,96,235,113,118,119,117,306,308,132,84,85,135,93,93
     ,93,93,93,93,94,130,148,188,146,179,178,323,322,321,324,86,330
     ,233,126,79,232,112,140,125,136,337,334,183,204,150,346,347,348,186
     ,129,350,182,133,239,239,239,239,239,239,239,239,259,122,205,181
     ,180,236,123,369,138);

declare index1 data(0,1,2,50,70,4,70,6,6,11,6,6,12,13,14,28,6,6,6,42,43
     ,45,70,46,70,47,6,238,49,50,50,60,59,60,68,70,70,135,78,70,135,81,82
     ,85,85,86,87,90,91,92,93,94,99,95,96,97,98,99,100,103,105,90,106,91
     ,108,109,111,113,103,114,116,117,117,117,117,117,117,119,120,50,122
     ,122,122,123,125,126,70,128,128,129,131,132,134,135,70,70,59,148,149
     ,150,151,152,171,173,176,177,179,181,183,187,188,189,190,191,201,202
     ,203,204,205,206,207,207,70,208,209,212,212,213,213,214,70,215,221
     ,223,70,224,70,70,226,227,228,229,230,135,233,215,234,234,236,233
     ,238,249,250,1,4,6,8,10,12,14,18,21,23,25,27,29,31,35,38,41,44,47,50
     ,53,55,57,59,61,63,65,67,69,71,78,89,82,84,91,93,95,97,1,2,4,4,5,6,7
     ,7,8,8,8,8,8,8,8,8,11,12,14,14,15,15,16,16,16,16,16,16,17,17,19,19,24,24
     ,24,28,28,28,31,32,32,33,33,33,33,33,33,33,33,33,33,33,33,36,36,41,42
     ,42,60,60,60,60,60,60,61,61,61,67,67,70,70,70,70,70,72,72,72,74,74
     ,90,91,91,92,92,92,92,93,93,95,96,97,97,98,99,99,100,101,102,102,103
     ,103,104,104,105,106,106,106,107,107,107,108,108,108,108,108,108,108
     ,109,109,109,109,112,112,114,115,115,116,116,117,118,126,127,127,130
     ,130,130,132,132,135,136,136,136,136,137,137,137,138,139,140,141,142
     ,143,144,145,146,148,148,149,150,150,151,151,152,152,153,154,154,155
     ,155,156,157,157,158,158,159,169,169,170,170,171,171,173,174,175,176
     ,176);

declare index2 data(0,1,2,9,8,2,8,5,5,1,5,5,1,1,14,14,5,5,5,1,2,1,8,1,8
     ,2,5,11,1,9,9,8,1,8,2,8,8,13,3,8,13,1,3,1,1,1,3,1,1,1,1,1,1,1,1,1,1
     ,1,3,2,1,1,2,1,1,2,2,1,2,2,1,2,2,2,2,2,2,1,2,9,1,1,1,2,1,2,3,1,1,2,1
     ,2,1,13,8,8,1,1,1,1,1,19,2,3,1,2,2,2,4,1,1,1,1,10,1,1,1,1,1,1,1,1,8
     ,1,3,1,1,1,1,1,8,6,2,1,8,2,8,8,1,1,1,1,3,13,1,6,2,1,2,1,11,1,3,3,2,2
     ,2,2,2,4,3,2,2,2,2,2,4,3,3,3,3,3,3,3,2,2,2,2,2,2,2,2,2,7,2,2,2,7,2,2,2
     ,4,1,1,2,0,0,2,0,2,0,0,1,0,1,0,0,1,2,1,2,0,2,0,1,0,0,1,2,2,2,0,0,0,3,1
     ,1,3,1,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,1,0,2,2,3,1,1,0,2,2,0,2,0
     ,0,0,2,2,0,0,0,0,0,1,0,2,1,0,1,0,2,2,1,1,1,2,2,0,2,2,2,1,2,1,1,1,0,2
     ,1,2,2,2,2,2,1,2,1,2,2,2,2,0,1,0,1,1,1,2,1,1,1,2,1,2,2,0,2,0,1,2,0,2
     ,2,0,0,0,0,1,1,1,1,1,0,3,1,1,1,0,1,1,2,1,0,0,0,0,0,0,3,3,2,3,2,2,3,2
     ,2,2,2,1,2,3,2,0,2,1,2,3,0,2);
```

```
    /*************************************************/
    /*              global procedures                */
    /*************************************************/

mon1: procedure(f,a);
      declare     f byte,
      a address;
      go to bdos;
end mon1;

mon2: procedure (f,a) byte;
      declare f byte, a address;
      go to bdos;
      end mon2;

mon3:procedure;
     /*used to return to the system*/
     goto boot;
end mon3;

move: procedure (a,b,l);
      /* moves from a to b for l bytes (l < 255) */
      declare (a,b) address,
      (s based a, d based b,l) byte;
      do while (l:=l - 1) <> 255;
           d=s;  b=b + 1;  a =a + 1;
      end;
      end move;

fill: proc (a,char,n);
       /* move char to a n times */
          declare a addr,(char,n,dest based a) byte;
          do while (n := n -1) <> 255;
                    dest = char;
                    a = a + 1;
          end;
end fill;

read: procedure;
     declare toggle(3) byte;
     toggle = 1;
     call mon1(10,.toggle);
end read;


printchar: procedure(char);
      declare char byte;
      call mon1(2,char);
end printchar;

print: procedure(a);
       declare a address;
       call mon1(9,a);
end print;

diskerr: procedure;
     call print(.'de     $');
     goto boot;
end diskerr;

open$sourcefile: procedure;
     call move(.'alg',rfcbaddr+9,3);
     rfcb(32) = 0;
     if mon2(15,rfcbaddr) = 255 then
          do;
          call print(.'ns $');
          go to boot;
          end;
end open$sourcefile;

close$int$file: procedure ;
      /* closes a file */
     if mon2(16,.wfcb) = 255 then
```

```
        call diskerr;
end close$int$file;

setup$int$file: procedure;
     /*  setup$int$files a new file */
    if nointfile then /*only make file if this toggle is off */
        return;
    call move(.rfcb,.wfcb,9);
    wfcb(32)=0;
    call mon1(19,.wfcb);
    if mon2(22,.wfcb) = 255 then
        call diskerr;
end setup$int$file;

rewind$source$file:proc;
    /*cp/m does not require any
      action prior to reopening*/
     return;
end rewind$source$file;

read$source$file:proc byte;
     declare dcnt byte;
     if(dcnt:=mon2(rfile,rfcbaddr)) > fileeof then
     call diskerr;
     return dcnt;
end read$source$file;

write$int$file: procedure;
    if nointfile then
       return;
    call mon1(26,.diskoutbuff);
    if mon2(21,.wfcb) <> 0 then
        call diskerr;
    call mon1(26,80h); /* reset dma address */
end write$int$file;

crlf: procedure;
     call printchar(cr);
     call printchar(lf);
end crlf;

printdec: procedure(value);
    declare value address, i byte, count byte;
    declare deci(4) address initial(1000,100,10,1);
    declare flag byte;
    flag = false;
    count = 30h;
    do i = 0 to 3;
      do while value >= deci(i);
           value = value - deci(i);
           flag= true;
           count = count + 1;
       end;
       if flag or (i>= 3) then
           call printchar(count);
       else
           call printchar(' ');
     end;
     return;
end printdec;

print$prod:proc;
    call print(.' prod = $');
    call print$dec(production);
    call crlf;
end print$prod;

print$token:proc;
    call print(.' token = $');
    call print$dec(token);
    call crlf;
end print$token;
```

```
emit:proc(objcode);
   declare objcode byte;
   if(buffptr:=buffptr+1) >= intrecsize then /*write to disk*/
     do;
       call write$int$file;
       buffptr=0;
     end;
   diskoutbuff(buffptr)=objcode;
end emit;

clear$line$buff:procedure;
      call fill(.linebuff,' ',conbuffsize);
end clear$line$buff;

listline: procedure(length);
    declare (length,i) byte;
    call print$dec(lineno);
    call print$dec(prev$index+1);
    call print$char(' ');
    do i = 0 to length;
       call printchar(linebuff(i));
    end;
    call crlf;
end listline;

    /*****************************************************************/
    /*   the following variables are used by the parser              */
    /*****************************************************************/

declare listprod     byte initial(false),
        lowertoupper byte initial(true),
        listsource   byte initial(false),
        debugin      byte initial(false),
        listtoken    byte initial(false),
        errset       byte initial(false),
        compiling    byte,
        codesize     address,  /* used to count size of code area */
        prtct        address initial(0fffeh),  /* used to count size of prt */

         /* variables used during for loop code generation */

        forcount     byte initial(0),
        randomfile   byte,
        fileio       byte initial(false);


   /*****************************************************************/
   /*            scanner procedures                                 */
   /*****************************************************************/

getchar: procedure byte;
     declare addeof data ('eof', eolchar,lf); /* add to end if left off */

     next$source$char: procedure byte;
          return sourcebuff(sourceptr);
     end next$source$char;

     checkfile: procedure byte;
      do forever;
        if (sourceptr:=sourceptr+1)>=cursourcerecsize then
          do;
             sourceptr=0;
             if read$source$file=fileeof then
                  return true;
          end;
        if(nextchar:=next$source$char)<>lf then
          return false;
      end;
     end checkfile;

    if checkfile or (nextchar = eoffiller) then
              do; /* eof reached */
              call move(.addeof,sbloc,3);
```

```
                sourceptr = 0;
                nextchar=next$source$char;
                end;
        linebuff(lineptr:=lineptr + 1)=nextchar; /*output line*/
        if nextchar  = eolchar  then
                do;
                  lineno = lineno + 1;
                  if listsource  then
                       call listline(lineptr-1);
                  lineptr = 0;
                  call clearlinebuff;
                end;
        if nextchar = tab then
            nextchar = ' ';
       return nextchar;
end getchar;

getnoblank: procedure;
          do while((getchar = ' ') or (nextchar = eoffiller));
                end;
end getnoblank;


title:procedure;
       call print(.'algol-m vers 1.0$');
       call crlf;
end title;

print$error:proc;
   call printdec(errorcount);
   call printchar(' ');
   call print(.'error(s) detected$');
   call crlf;
end print$error;

error: procedure(errcode);
        declare errcode address,
                i       byte;
        errorcount=errorcount+1;
        call print(.'***$');
        call print$dec(lineno);
        call print(.'  error #');
        call printchar(' ');
        call printchar(high(errcode));
        call printchar(low(errcode));
        call crlf;
        call print$prod;
        if token=eofc then
          do;
           call print$error;
           call mon3;
          end;
end error;


initialize$scanner: procedure;
    declare count byte;
    call open$sourcefile;
    lineno,lineptr = 0;
    call clear$line$buff;
    sourceptr = 128;
    call getnoblank;
      do while nextchar = '$';
         call get$no$blank;
         if(count := (nextchar and 5fh) - 'a') <= 4 then
                do case count;
                      if pass1 then listsource = true;
                      listprod = true;
                      nointfile = true;
                      listtoken = true;
                      debugin = true;
                end; /* of case */
      call getnoblank;
```

```
      end;
end initialize$scanner;

    /*****************************************************/
    /*                  scanner                          */
    /*****************************************************/

scanner: procedure;

     putinaccum: procedure;
          if not cont then
               do;
               accum(accum := accum + 1) = nextchar;
               hashcode = (hashcode + nextchar) and hashmask;
               if accum = 31 then cont = true;
               end;
     end putinaccum;

     putandget: procedure;
          call putinaccum;
          call getnoblank;
     end putandget;

     putandchar: procedure;
          call putinaccum;
          nextchar = getchar;
     end putandchar;

     numeric: procedure byte;
          return(nextchar - '0') <= 9;
     end numeric;

     lowercase: procedure byte;
          return (nextchar >= 61h) and (nextchar <= 7ah);
     end lower$case;

     decimalpt:proc byte;
          return nextchar='.';
     end decimalpt;

     conv$to$upper:proc;
        if lowercase and lowertoupper then
           nextchar=nextchar and 5fh;
     end conv$to$upper;

     letter: procedure byte;
        call conv$to$upper;
        return ((nextchar - 'a') <= 25) or lowercase;
     end letter;

     alphanum: procedure byte;
          return numeric or letter or decimalpt;
     end alphanum;

     spoolnumeric: procedure;
               do while numeric;
                  call putandchar;
                end;
     end spoolnumeric;


     setup$next$call: procedure;
          if nextchar = ' ' then
               call getnoblank;
          cont = false;
     end setup$next$call;


lookup: procedure byte;

declare maxrwing lit  '9';
```

```
declare vocab data(0,'<','(','+',5dh,7ch,'*',')',';','-','/',',','>'
    ,':',5bh,'=','**',':=','do','go','if','of','or','to','eof','and'
    ,'end','for','not','pic','tab','case','else','file','goto','read'
    ,'step','then','array','begin','close','until','while','write'
    ,'string','decimal','initial','integer','writeon'
    ,'comment','external','function','procedure');
declare vloc data(0,1,16,32,53,81,111,117,152,168,177);
declare vnum data(0,1,16,24,31,38,44,45,50,54);
declare count data(0,14,7,6,6,5,0,4,1,0);
declare ptr address, (field based ptr) (9) byte;
declare i byte;

        compare: procedure byte;
             declare i byte;
             i = 0;
             do while (field(i) = accum(i := i + 1)) and i <= accum;
                  end;
             return i > accum;
        end compare;

        if accum > maxrwlng then
             return false;
        ptr=vloc(accum)+.vocab;
        do i=vnum(accum) to (vnum(accum)+count(accum));
            if compare then
                do;
                  if i=50 then
                    token=comment;
                  else
                    token=i;
                  return true;
                end;
            ptr=ptr+accum;
        end;
        return false;
    end lookup;


/*****************************************************/
/*             scanner main code                     */
/*****************************************************/


    do forever;
      accum, hashcode, token = 0;
      do while nextchar=eolchar;
          call getnoblank;
          end;
      if(nextchar = stringdelim) or cont then
         do; /* found string */
              token = string;
              cont = false;
              do forever;
                   do while getchar <> stringdelim;
                        call putinaccum;
                        if cont then return;
                        end;
                   call getnoblank;
                   if nextchar <> stringdelim then
                        return;
                   call put$in$accum;
              end; /* of do forever */
         end; /* of recognizing a string */


  else if numeric or decimalpt then
         do; /* have digit */
              token = integerc;
              do while nextchar='0'; /*elim leading zeros*/
                 nextchar=getchar;
              end;
              call spoolnumeric;
              if decimalpt then
```

```
                do;
                token=decimal;
                call putandchar;
                call spoolnumeric;
                end;
            if accum=0 then
                hashcode, accum(accum := 1) = '0';
            call setup$next$call;
            return;
        end; /* of recoguizing numeric constant */


    else if letter then
        do; /* have a letter */
            do while alphanum;
                call putandchar;
                end;
            if not lookup then
            do;
                token = identifier;
                call setup$next$call;
                return;
            end;
            else /* is a rw but if comment skip */
                if token = comment then
                   do;
                     do while nextchar <> ';';
                          nextchar = getchar;
                          end;
                     call get$no$blank;
                   end;
                else
                    do;
                      call set$up$next$call;
                      return;
                    end;
        end; /* of recognizing rw or ident */


    else
     do; /* special character */
       if nextchar = 25h then
          do;
              nextchar=getchar;
              do while nextchar <> 25h;
                 nextchar = getchar;
              end;
              call get$no$blank;
          end;
       else
         do;
           if nextchar = ':' then
             do;
               call putandchar;
               if nextchar = '=' then
               call putandget;
             end;
          else
            if nextchar = '*' then
             do;
               call putandchar;
               if nextchar = '*' then
               call putandget;
             end;
            else  call putandget;
            if not lookup then
              call error('ic');
            call setup$next$call;
            return;
         end;
     end; /* of recognizing special char */
    end;  /* of do forever */
end scanner; /* end of scanner */
```

```
/*****************************************************/
/*      procedures for synthesizer                   */
/*****************************************************/

initialize$symtbl:proc;
    if pass1 then
      do:
   /* fill hashtable with 0's */
      call fill(.hashtable,0,shl(hashtblsize,2));
      sbtbl = .memory;
      end;
   /*initialize pointer to top of symbol table*/
    sbtbltop = max - 2;
end initialize$symtbl;

setaddrptr:proc(offset); /*set ptr for addr reference*/
    declare offset byte;
    aptraddr = base + ptr + offset;/*position for addr reference*/
end setaddrptr;

set$blk$level:proc(level);
   declare level byte;
   call setaddrptr(6);
   byteptr = level;
end set$blk$level;

gethash:proc byte;
   declare hash byte,
           i    byte;
    hash = 0;
    aptraddr = base + 2;
    do i = 1 to ptr;
       hash = (hash + byteptr(i)) and hashmask;
    end;
    return hash;
end gethash;

nextentry:proc;
   base = base + ptr + 8;
end nextentry;

setlink:proc;
   aptraddr = base + 1;
end setlink;

hashtbl$of$symhash:proc address;
   return hashtable(symhash);
end hashtbl$of$symhash;

limits:proc(count);
     /*check to see if additional sbtbl will overflow limits of
      memory. if so then punt else return */
      declare count byte;  /*size being added is count */
      if sbtbltop <= (sbtbl + count) then
        do:
          call error('to');
          call mon3;
        end;
end limits;

setaddr:proc(loc);
     /*set the address field and resolved bit*/
     declare loc address;
     call setaddrptr (4);
     addrptr = loc;
end setaddr;

lookup$current$blk:proc(chk$blk) byte;
   declare chk$blk byte,
        len byte,
         n based printname byte;
```

```
   base=hashtbl$of$symhash;
   do while base <> 0;
      call setaddrptr(6);
      if byteptr < chk$blk then
           return false;
      if byteptr = chk$blk then
         do;
           if (len:=ptr) = n then
             do while (ptr(len + 2) = n(len));
               if (len := len - 1) = 0 then
                  return true;
             end;
      end;
      call setlink;
      base = addrptr;
   end;
   return false;
end lookup$current$blk;

lookup:proc byte;
 declare test$blk byte,
         test$index byte;
 test$index = prev$index+1;
 test$blk = blk$level;
 do while (test$index := test$index - 1) <> 255;
    if lookup$current$blk(test$blk) then
         return true;
    test$blk = prev$blk$level(test$index);
 end;
 return false;
end lookup;

enter:proc;
   /*enter token reference by printname and symhash
     into next available location in the symbol table.
     set base to beginning of this entry and increment
     sbtbl.  also check for symbol table full.  */
   declare i byte,
           n based printname byte;
   call limits(i:=n+8);
   base = sbtbl; /*base for new entry */
   call move(printname + 1,sbtbl + 3,(ptr := n));
   call setaddrptr(3);/*set resolve bit to 0*/
   byteptr = 0;
   call setlink;
   addrptr = hashtbl$of$symhash;
   hashtable(symhash) = base;
   call set$blk$level(blk$level);
   sbtbl = sbtbl + i;
end enter;

getlen:proc byte; /*return length of the p/n */
   return ptr;
end getlen;

getype:proc byte; /*returns type of variable*/
   call setaddrptr (3);
   return byteptr;
end getype;

setsubtype:proc(stype);/*enter the subtype in sbtbl*/
     declare stype byte;
     call setaddrptr(7);
     byteptr=stype;
end setsubtype;

get$parm: proc byte;
     call setaddrptr(10);
     return byteptr;
end getparm;

getsubtype:proc byte;/*return the subtype*/
     call setaddrptr(7);
```

```
     return byteptr;
end getsubtype;
setype:proc (type); /*set typefield = type*/
   declare type byte;
    call setaddrptr (3);
    byteptr =  type;
end setype;

getaddr:proc address;
    call setaddrptr(4);
    return addrptr;
end getaddr;


do;   /* block for parser */

      /* pneumonics for ALGOL-M machine  */

declare nop lit  '0',  str lit  '1',  int lit  '2',  xch lit  '3',
        lod lit  '4',  dcb lit  '5',  dmp lit  '6',  xit lit  '7',
        ald lit  '8',  als lit  '9',  aid lit '10',  ais lit '11',
        adl lit '12',  add lit '13',  sbl lit '14',  sbd lit '15',
        mpi lit '16',  mpd lit '17',  dvi lit '18',  dvd lit '19',
       dneg lit '21',  neg lit '22',  cil lit '23',  ci2 lit '24',
       deci lit '25',  pop lit '26',  imi lit '27',  im2 lit '28',
                                      cat lit '31',  bli lit '32',
        brs lit '34',  bsc lit '35',  lss lit '36', dlss lit '37',
       slss lit '38',  gtr lit '39', dgtr lit '40', sgtr lit '41',
        eql lit '42', deql lit '43', seql lit '44',  neq lit '45',
       dneq lit '46', sneq lit '47',  geq lit '48', dgeq lit '49',
       sgeq lit '50',  leq lit '51', dleq lit '52', sleq lit '53',
       inot lit '54', dnot lit '55', snot lit '56', iand lit '57',
       dand lit '58', sand lit '59',  ior lit '60',  dor lit '61',
        sor lit '62',  wic lit '63',  wdc lit '64',  wsc lit '65',
        wid lit '66',  wdd lit '67',  wsd lit '68',  sbr lit '69',
        bra lit '70',  row lit '71',  sub lit '72',  rcl lit '73',
        rcd lit '74',  rcs lit '75',  rdi lit '76',  rdd lit '77',
        rds lit '78',  rcn lit '79',  ecr lit '80',  sil lit '81',
        sdi lit '82',  ssi lit '83',  sld lit '84',  sdd lit '85',
        ssd lit '86',  opn lit '87',  cls lit '88',  rdb lit '89',
        rdf lit '90',  edr lit '91',  edw lit '92',  pro lit '93',
        sav lit '94',  sv2 lit '95',  uns lit '96',  rtn lit '97';

declare state                    statesize,
        statestack(pstacksize)   statesize,
        hash(pstacksize)           byte,
        symloc(pstacksize)         address,
        srloc(pstacksize)          address,
        var(pstacksize)            byte,
        type(pstacksize)           byte,
        stype(pstacksize)          byte,
        varc(varcsize)             byte,
        varindex                 byte,
        (sp,mp,mpp1,nolook)        byte,
        onstack(maxoncount)        byte,
        clabing                  byte initial(2),
        clab2                    byte initial(23),
        clable                   byte,
        str$size                 byte initial(10),
        dec$size                 byte initial(9),
        n                        byte,
        pvnum                    byte initial(0),
        saveparm                 address,
        parme based saveparm     byte,
        fpcount                  byte,
        parmbase                 address,
        proc$type                byte,
       (ptest,i)                 byte,
        pcount                   byte initial(0),
        typetemp                 byte,
        fflag                    byte,
        lpcount                  byte,
        sident                   address;
```

```
initialize$synthesize:procedure;
    codesize,onstack,clable = 0;
    prev$index = 255;
    blk$cnt = 0;
    blk$level=0;
end initialize$synthesize;


synthesize: proc;


  /* **** synthesize local declarations  **** */


declare simvar          lit '0bh',
        subvar          lit '99',
        pro             lit  '93',
        ext$proc        lit '03',
        blt$in$func     lit '05',
        const           lit '06',
        lab             lit '07',
        integer         lit '08',
        str             lit '1',
        filel           lit '0ch',
        func            lit '0dh',
        parm            lit '10h';

declare (typesp,typemp,typempl)         byte,
        (b,temp)                        byte,
        (stypesp,stypemp,stypempl)      byte,
        (hashsp,hashmp,hashmpl)         byte,
        (symlocsp,symlocmp, symlocmpl) address,
        (srlocsp,srlocmp)               address;

    /*   ***************************************************   */
    /*   **********     code generation proc's     **********   */
    /*   ***************************************************   */

     copy:procedure;
          typesp = type(sp);
          typempl = type(mppl);
          typemp = type(mp);
          stypesp = stype(sp);
          stypempl = stype(mppl);
          stypemp = stype(mp);
          symlocsp = symloc(sp);
          symlocmpl = symloc(mppl);
          symlocmp = symloc(mp);
          hashmp = hash(mp);
          hashmpl = hash(mppl);
          hashsp = hash(sp);
          srlocsp = srloc(sp);
          srlocmp = srloc(mp);
     end copy;


    setsymlocsp: procedure(a);
         declare a address;
         symloc(sp) = a;
    end setsymlocsp;

     setsymlocmp: procedure(a);
       declare a address;
       symloc(mp) = a;
     end setsymlocmp;

    settypesp: procedure(b);
         declare b byte;
         type(sp) = b;
    end settypesp;
```

```
    setstypesp: procedure(b);
         declare b byte;
         stype(sp) = b;
    end setstypesp;

    setstypemp: procedure(b);
         declare b byte;
         stype(mp) = b;
    end setstypemp;

    settypemp: procedure(b);
         declare b byte;
         type(mp) = b;
    end settypemp;


    sethashmp: procedure(b);
         declare b byte;
         hash(mp) = b;
    end sethashmp;

    sethashsp: procedure(b);
         declare b byte;
         hash(sp) = b;
    end sethashsp;

    setsrlocsp: procedure(a);
         declare a address;
         srloc(sp) = a;
    end setsrlocsp;
   setsrlocmp: proc(a);
       declare a byte;
       srloc(mp)=a;
   end setsrlocmp;

   getsrloc: proc byte;
       call setaddrptr(8);
       return addrptr;
   end  getsrloc;

     generate:proc(objcode);
       /*writes generated code and counts size
       of code area.  */
       declare objcode byte;
       codesize = codesize + 1;
       if not pass1 then
         call emit(objcode);
     end generate;

     gen$int$v: proc(a);
         declare a byte;
         call generate(imi);
         call generate(a);
     end gen$int$v;

incr$blk$level: proc;
     prev$blk$level(prev$index := prev$index+1) = blk$level;
     blk$level = blk$cnt + 1;
     blk$cnt = blk$cnt+1;
     call generate(bll);
end incr$blk$level;

decr$blk$level: proc;
     blk$level = prev$blk$level(prev$index);
     prev$index = prev$index-1;
     call generate(bld);
end decr$blk$level;

calc$varc: procedure(b) address;
  declare b byte;
  return var(b) + .varc;
end calc$varc;
```

```
setlookup: procedure(a);
     declare a byte;
     printname = calc$varc(a);
     symhash = hash(a);
end setlookup;

lookup$only: procedure(a) byte;
     declare a byte;
     call setlookup(a);
     if lookup$current$blk(blk$level) then
          return true;
     base$flag = true;
     return false;
end lookup$only;

full$lookup:proc(a) byte;
     declare a byte;
     call setlookup(a);
     if lookup then
       return true;
     return false;
end full$lookup;

normal$lookup: procedure(a) byte;
     declare a byte;
     if lookup$only(a) then
          return true;
     call enter;
     return false;
end normal$lookup;

countprt:proc address;
/*counts the size of the prt */
   return (prtct := prtct + 2);
end countprt;

gentwo:proc(a);
/* writes two bytes of object code on disk for literals */
  declare a address;
  call generate(high(a));
  call generate(low(a));
end gentwo;


literal:proc(a);
   declare a address;
   call gentwo(a or 8000h);
end literal;

setcname:proc;
     printname=.clablng;
     symhash=clable and hashmask;
end setcname;

enter$compiler$label:proc(b);
     declare b byte;
     if pass1 then
       do;
         call setcname;
         call enter;
         call setaddr(codesize + b);
       end;
end enter$compiler$label;

set$compiler$label:proc;
     declare x byte;
     clable = clable + 1;
     call setcname;
     if pass2 then
       x=lookup;
end set$compiler$label;

compiler$label:proc;
```