```
(* author : NASSER ABBASI
   final project : compiler design cse565 Oakland University
   Compiler for Subset Of Pascal For 8088
   april 1988

   description:
    this is the BACKEND compiler for pascal -> 8088 assempler.

    the FRONT END was written in C using lex and yacc.

    Input Files: this program Reads the parse table Generated by
    Yacc running on the prime (primix OS) and the table re build
    and scanned a number of times to build needed tables ( symbol
    table , hashed into linked list, and block description table,
    etc.
    Also the Lex genrated identifires tables are downloaded. these
    include the ident,integer,real, and string tables.

    Output files: the assemply code .

    language used : VAX pascal.

FINAL :
  at the bottom of this file ( after backend.pas ) is the output
  for the 3 problems assigned.
  for each problem there is the assemply generated code and the symbol
  table and the bst table. ( encloded also an example of sucssesful
assemply
  of the generated code on the ibm pc using MASM )

%a 5000
letter   [A-Za-z]
digit    [0-9]+
id       [a-zA-Z][a-zA-Z0-9_]*
%%
" "                                     ;
\n                                      ;
\{                                      comment=1;
\}                                      comment=0;
\+                                      if(!(comment)) return('+');
\-                                      if(!(comment)) return('-');
\*                                      if(!(comment)) return('*');
\/                                      if(!(comment)) return('/');
\=                                      if(!(comment)) return('=');
\<                                      if(!(comment)) return('<');
\>                                      if(!(comment)) return('>');
\(                                      if(!(comment)) return('(');
\)                                      if(!(comment)) return(')');
\[                                      if(!(comment)) return('[');
\]                                      if(!(comment)) return(']');
\,                                      if(!(comment)) return(',');
\:                                      if(!(comment)) return(':');
\;                                      if(!(comment)) return(';');
\'                                      if(!(comment)) return('\'');
```

```
\#                                  if(!(comment)) return('#');
\$                                  if(!(comment)) return('$');
\:\=                                if(!(comment)) return(tkasg());
\<\>                                if(!(comment)) return(tkne());
\<\=                                if(!(comment)) return(tkle());
\>\=                                if(!(comment)) return(TKGE);
\.\.                                if(!(comment)) return(TKDTDT);
\.                                  if(!(comment)) return('.');
[Aa][Bb][Ss][Oo][Ll][Uu][Tt][Ee]    if(!(comment)) return(TKABSOLUTE);
[Aa][Nn][Dd]                        if(!(comment)) return(TKAND);
[Aa][Rr][Rr][Aa][Yy]                if(!(comment)) return(TKARRAY);
[Bb][Ee][Gg][Ii][Nn]                if(!(comment)) return(tkbegin());
[Cc][Aa][Ss][Ee]                    if(!(comment)) return(TKCASE);
[Cc][Oo][Nn][Ss][Tt]                if(!(comment)) return(TKCONST);
[Cc][Hh][Aa][Rr]                    if(!(comment)) return(TKCHAR);
[Bb][Yy][Tt][Ee]                    if(!(comment)) return(TKBYTE);
[Dd][Ii][Vv]                        if(!(comment)) return(TKDIV);
[Dd][Oo]                            if(!(comment)) return(TKDO);
[Dd][Oo][Ww][Nn][Tt][Oo]            if(!(comment)) return(TKDOWNTO);
[Ee][Ll][Ss][Ee]                    if(!(comment)) return(TKELSE);
[Ee][Nn][Dd]                        if(!(comment)) return(TKEND);
[Ee][Xx][Tt][Ee][Rr][Nn][Aa][Ll]    if(!(comment)) return(TKEXTERNAL);
[Ff][Ii][Ll][Ee]                    if(!(comment)) return(TKFILE);
[Ff][Oo][Rr][Ww][Aa][Rr][Dd]        if(!(comment)) return(TKFORWARD);
[Ff][Oo][Rr]                        if(!(comment)) return(TKFOR);
[Ff][Uu][Nn][Cc][Tt][Ii][Oo][Nn]    if(!(comment)) return(TKFUNCTION);
[Gg][Oo][Tt][Oo]                    if(!(comment)) return(TKGOTO);
[Ii][Nn][Ll][Ii][Nn][Ee]            if(!(comment)) return(TKINLINE);
[Ii][Ff]                            if(!(comment)) return(TKIF);
[Ii][Nn]                            if(!(comment)) return(TKIN);
[Ll][Aa][Bb][Ee][Ll]                if(!(comment)) return(TKLABEL);
[Mm][Oo][Dd]                        if(!(comment)) return(TKMOD);
[Nn][Ii][Ll]                        if(!(comment)) return(TKNIL);
[Nn][Oo][Tt]                        if(!(comment)) return(TKNOT);
[Oo][Vv][Ee][Rr][Ll][Aa][Yy]        if(!(comment)) return(TKOVERLAY);
[Oo][Ff]                            if(!(comment)) return(TKOF);
[Oo][Rr]                            if(!(comment)) return(TKOR);
[Pp][Aa][Cc][Kk][Ee][Dd]            if(!(comment)) return(TKPACKED);
[Pp][Rr][Oo][Cc][Ee][Dd][Uu][Rr][Ee]  if(!(comment))
return(tkprocedure());
[Pp][Rr][Oo][Gg][Rr][Aa][Mm]        if(!(comment)) return(tkprogram());
[Rr][Ee][Cc][Oo][Rr][Dd]            if(!(comment)) return(TKRECORD);
[Rr][Ee][Pp][Ee][Aa][Tt]            if(!(comment)) return(TKREPEAT);
[Ss][Ee][Tt]                        if(!(comment)) return(TKSET);
[Ss][Hh][Ll]                        if(!(comment)) return(TKSHL);
[Ss][Hh][Rr]                        if(!(comment)) return(TKSHR);
[Ss][Tt][Rr][Ii][Nn][Gg]            if(!(comment)) return(TKSTRING);
[Tt][Hh][Ee][Nn]                    if(!(comment)) return(TKTHEN);
[Tt][Yy][Pp][Ee]                    if(!(comment)) return(TKTYPE);
[Tt][Oo]                            if(!(comment))  return(TKTO);
[Tt][Ee][Xx][Tt]                    if(!(comment)) return(TKTEXT);
[Uu][Nn][Tt][Ii][Ll]                if(!(comment)) return(TKUNTIL);
[Vv][Aa][Rr]                        if(!(comment)) return(tkvar());
[Ww][Hh][Ii][Ll][Ee]                if(!(comment)) return(TKWHILE);
```

```
[Ww][Ii][Tt][Hh]                        if(!(comment)) return(TKWITH);
[Xx][Oo][Rr]                            if(!(comment)) return(TKXOR);
[Rr][Ee][Aa][Ll]                        if(!(comment)) return(TKREAL);
[Bb][Oo][Oo][Ll][Ee][Aa][Nn]            if(!(comment)) return(TKBOOLEAN);
[Ii][Nn][Tt][Ee][Gg][Ee][Rr]            if(!(comment)) return(TKINTEGER);
[Rr][Ee][Aa][Dd][Ll][Nn]                if(!(comment)) return(TKREADLN);
[Ww][Rr][Ii][Tt][Ee][Ll][Nn]            if(!(comment)) return(TKWRITELN);
[Rr][Ee][Aa][Dd]                        if(!(comment)) return(TKREAD);
[Ww][Rr][Ii][Tt][Ee]                    if(!(comment)) return(TKWRITE);
[Tt][Rr][Uu][Ee]                        if(!(comment)) return(TKTRUE);
[Ff][Aa][Ll][Ss][Ee]                    if(!(comment)) return(TKFALSE);
{id}                                    if(!(comment)) return(stelookup());
{digit}                                 if(!(comment)) return(stilookup());
\$[0-9A-Fa-f]+                          if(!(comment)) return(stilookup());
[0-9]+\.[0-9]*([Ee][-+]?[0-9]+)?        if(!(comment)) return(strlookup());
\.[0-9]+([Ee][-+]?[0-9]+)?              if(!(comment)) return(strlookup());
\#[0-9]+                                if(!(comment))
return(stringlookup());
\^[A-Za-z]                              if(!(comment))
return(stringlookup());
\'.*\'                                  if(!(comment))
return(stringlookup());
%%
/*************************************************/

int tkprocedure()
{
 int debug= 0;

   if (debug)
      printf ("\n saw procedure token \n");
   else
   ;

     return(TKPROCEDURE);

}
/*********************/
int tkasg()

{
 int debug= 0;

   if (debug)
      printf ("\n saw :=  token \n");
   else
   ;

     return(TKASG);

}
/*******************/

int tkne()
```

```
{
 int debug= 0;

   if (debug)
      printf ("\n saw ne token \n");
   else
   ;

     return(TKNE);

}
/********************/

int tkle()
{
 int debug= 0;

   if (debug)
      printf ("\n saw le token \n");
   else
   ;

     return(TKLE);

}
/******************/

int tkbegin()

{
 int debug= 0;

   if (debug)
      printf ("\n saw begin token \n");
   else
   ;

     return(TKBEGIN);

}
/****************/

int tkvar()
{
 int debug= 0;

   if (debug)
      printf ("\n saw var token \n");
   else
   ;

     return(TKVAR);

}
```

```
/*****************************************************/
int tkprogram()
{
int debug=0;

 if (debug)
    printf (" saw program token \n");
 else
 ;

   return(TKPROGRAM);
}
/*****************************************/
int stelookup()
{
 int debug=0;
 int len=0;
 int yes=1;
 int no= 0;
 int found;
 int keep_searching;

  if (debug)
     printf ("\n in stelookup yytext= %s
ident_index=%d",yytext,ident_index);
  else
  ;

  ident_index =0;
  keep_searching = yes;
  found = no;

  while (keep_searching)
        {
         if (ident_index > symtable_last)
            keep_searching = no;
         else
            if (symtable[ident_index] != NULL )
               if (strcmp(symtable[ident_index],yytext) != 0)
                  ident_index++;
               else
                  {
                  found=yes;
                  keep_searching = no;
                  }
            else
                keep_searching = no;
        }

     if (!(found))
       {
          symtable[symtable_last] =(char *) malloc(strlen(yytext)+1);
          strcpy(symtable[symtable_last],yytext);
```

```
        printf ("in symtable after search fail
ident_index=%d",ident_index);
         symtable_last++;
       }
     else
        printf ("in symtable after search found
ident_index=%d",ident_index);


return(ident);
}
/*********************************************************/
int stringlookup()
{
 int debug=1;

 int yes=1;
 int no= 0;
 int found;
 int keep_searching;

  string_index =0;
  keep_searching = yes;
  found = no;

  while (keep_searching)
        {
         if (string_index > string_last)
            keep_searching = no;
         else
            if (stringtable[string_index] != NULL )
               if (strcmp(stringtable[string_index],yytext) != 0)
                  string_index++;
               else
                  {
                  found=yes;
                  keep_searching = no;
                  }
            else
                keep_searching = no;
        }

    if (!(found))
       {
       stringtable[string_last] = (char *) malloc(strlen(yytext)+1);
       strcpy (stringtable[string_last],yytext);
       string_last++;
     }

 return(string);
}
/*************************************************************/
int stilookup()
{
```

```
 int yes=1;
 int no= 0;
 int found;
 int keep_searching;

  int_index =0;
  keep_searching = yes;
  found = no;

  while (keep_searching)
        {
         if (int_index > inttable_last)
            keep_searching = no;
         else
            if (inttable[int_index] != NULL )
               if (strcmp(inttable[int_index],yytext) != 0)
                  int_index++;
               else
                  {
                  found=yes;
                  keep_searching = no;
                  }
            else
                keep_searching = no;
        }

 if (!(found))
   {
       inttable[inttable_last] = (char *) malloc(strlen(yytext)+1);
       strcpy(inttable[inttable_last],yytext);
      inttable_last++;
    }
  return(integer);
}
/*******************************************************/

int strlookup()
{

 int yes=1;
 int no= 0;
 int found;
 int keep_searching;

  real_index =0;
  keep_searching = yes;
  found = no;

  while (keep_searching)
        {
         if (real_index > realtable_last)
            keep_searching = no;
         else
```

```
            if (realtable[real_index] != NULL )
               if (strcmp(realtable[real_index],yytext) != 0)
                  real_index++;
               else
                  {
                  found=yes;
                  keep_searching = no;
                  }
            else
                keep_searching = no;
        }

 if (!(found))
     {
       realtable[realtable_last] = (char *) malloc(strlen(yytext)+1);
       strcpy(realtable[realtable_last],yytext);
       realtable_last++;
    }
return(real);
}


Y A C C ************************************

procedure print_token(a : integer)
begin

    case a of
      TKASG:     write(':=  ')  ;
      TKNE:      write('NE  ')  ;
      TKLE:      write('LE  ')  ;
      TKGE:      write('GE  ')  ;
      TKDTDT:    write('..  ')  ;
      TKABSOLUTE:      write('ABSOLUTE  ')  ;
      TKAND:     write('AND  ')  ;
      TKARRAY:   write('ARRAY  ')  ;
      TKBEGIN:   write('BEGIN  ')  ;
      TK : write('   ')  ;
        TKCONST: write('CONST  ')  ;
        TKDIV:   write('DIV  ')  ;
        TKDO:    write('DO  ')  ;
        TKDOWNTO:      write('DOWNTO  ')  ;
        TKELSE:  write('ELSE  ')  ;
        TKEND:   write('END  ')  ;
        TKEXTERNAL:    write('EXTERNAL  ')  ;
        TKFILE:  write('FILE  ')  ;
        TKFORWARD:     write('FORWARD  ')  ;
        TKFOR:   write('FOR  ')  ;
        TKFUNCTION:    write('FUNCTION  ')  ;
        TKGOTO:  write('GOTO  ')  ;
        TKINLINE:      write('INLINE  ')  ;
        TKIF:    write('IF  ')  ;
        TKIN:    write('IN  ')  ;
        TKLABEL: write('LABEL  ')  ;
```

```
        TKMOD:   write('MOD  ')  ;
        TKNIL:   write('NIL  ')  ;
        TKNOT:   write('NOT  ')  ;
        TKOVERLAY:     write('OVERLAY  ')  ;
        TKOF:    write('OF  ')  ;
        TKOR:    write('OR  ')  ;
        TKPACKED:      write('PACKED  ')  ;
        TKPROCEDURE: write('PROCEDURE  ')  ;
        TKPROGRAM:     write('PROGRAM  ')  ;
        TKRECORD:      write('RECORD  ')  ;
        TKREPEAT:      write('REPEAT  ')  ;
        TKSET:   write('SET  ')  ;
        TKSHL:   write('SHL  ')  ;
        TKSHR:   write('SHR  ')  ;
        TKSTRING:      write('STRING  ')  ;
        TKTHEN:  write('THEN  ')  ;
        TKTYPE:  write('TYPE  ')  ;
        TKTO:    write ('TO  ')  ;
        TKUNTIL: write('UNTIL  ')  ;
        TKVAR:   write('VAR  ')  ;
        TKWHILE: write('WHILE  ')  ;
        TKWITH:  write('WITH  ')  ;
        TKXOR:   write('XOR  ')  ;
        TKREAL:  write('REAL  ')  ;
        TKBOOLEAN:     write('BOOLEAN  ')  ;
        TKINTEGER:     write('INTEGER  ')  ;
        TKREAD:  write('READ  ')  ;
        TKWRITE: write('WRITE  ')  ;
        TKTRUE:  write('TRUE  ')  ;
        TKFALSE: write ('FALSE  ')  ;
        TKWRITELN:  write('WRITELN ')  ;
        TKREADLN: write('READLN ')  ;
        TKBYTE:    write('BYTE ')  ;
        otherwise do
          begin
             write ('error in write token unknown token number') ;
            error;
         end
    end
end;
(*  B A C K E N D  *)

*)

program BackEnd(input,output);
 const
     tkasg=257;      tkne=258;      tkle=259;    tkge=260;    tkdtdt=261;
     tkabsolute=262; tkand=263;     tkarray=264; tkbegin=265;
     tkcase=266;     tkconst=267;   tkdiv=268;   tkdo=269;
tkdownto=270;
     tkelse=271;     tkend=272;     tkexternal=273;
tkfile=274;
     tkforward=275;  tkfor=276;     tkfunction=277;
tkgoto=278;
```

```
     tkinline=279;   tkif=280;      tkin=281;
tklabel=282;
     tkmod=283;
     tknil=284;      tknot=285;     tkoverlay=286;             tkof=287;
     tkor=288; tkpacked=289;
     tkprocedure=290; tkprogram=291; tkrecord=292;
     tkrepeat=293;    tkset=294;     tkshl=295;   tkshr=296;
tkstring=297;
     tkthen=298;      tktype=299;    tkto=300;    tkuntil=301;
     tkvar=302;       tkwhile=303;   tkwith=304;  tkxor=305;
tktext=306;
     tkchar=307;      tkreadln=308;  tkwriteln=309;
tkreal=310;
     tkboolean=311;
     tkinteger=312;   tkread=313;    tkwrite=314;  tktrue=315;
tkfalse=316;
     tkbyte=317;      tkputc=318;    tkgetc=319;
     lex_string=317;  lex_real=318;  lex_ident=319; lex_integer=320;

     subtree=1;
     literal=2;
     ident=3;
     token=4;
     integer_ident=5;
     real_ident=6;
     string_ident=7;
     empty=8;
     ident_size = 50;
     literal_max_size = 50;
     hash_size = 67;
     hlimit = 66;
     maxlen = 30; (* max length of indentifier *)
     stack_max = 20;
 type
     buildin = (chr,ord);  (* build in identifiers *)
     op_type = (sub_,add_,mul_,div_,assign_,le_,gt_);
     maxNest = 0..10;
     treeSize = 0..550;
     symtableSize = 0..150;
     status = 1..100;  (* return status codes *)
     string50 = varying [50] of char;

     (*  P A R S E    T R E E  *)
     treeNodeType = record
                    rhsn :integer;
                    rhstype : array[1..10] of integer;
                    rhsindex : array[1..10] of integer
                    end;
     ParseTreeType = array[treeSize] of treeNodeType;


     (*      B  S  T        T A B L E *)
     BstNodeType = record
                   OuterBlock   : integer;
```

```
                    LexicalLevel : integer;
                    local_size   : integer; (* size of local storage *)
                    parm_size    : integer; (* size of parameters *)
                    block_name   : string50;
                    block_num    : integer;
                    end;

     (* S Y M B O L  T A B L E *)
     sym_type_ = (variable,parm,entry,constant);
     vtype = (byte_,integer_,boolean_,char_,array_,notused);
     symbol = string50;
     symtabp = ^symtabtype;
     symtabtype = record
                  next :symtabp;
                  LEVEL : integer;
                  sym : symbol;
                  saddr  : integer;
                  parm_flag : boolean;
                  vtype_  : vtype; (* data type*)
                  sem_type : sym_type_;
                  blk_num : integer;
                  literal_val : varying [literal_max_size] of char;
                  size : integer;
                  END;

 (*  A S S E M P L Y    L I N E  *)
   assemply_line_type = varying[80] of char;

 (*  C O D E  G E N R A T I O N  S T A C K  *)
     stack_type = record
                  data : array[1..stack_max] of integer;
                  tos  : integer;
                  end;
var
     unique: integer;
     initial_label : symbol;
     debug : boolean;
     LHS : boolean; (* to tell if address or value generating *)
     arr : boolean; (* to tell if lhs is array *)
     assemply_line : assemply_line_type; (* emit string to assemply file
*)
     assemply_line_number : integer; (* to emit number to assemply file
*)
     symf,intf,realf,stringf,treef,assemply : text;
     tables : text;
     symtable : array [symtableSize] of symbol;
     inttable : array [symtableSize] of symbol;
     realtable : array [symtableSize] of symbol;
     stringtable : array [symtableSize] of string50;

     stack : stack_type;
     tree : parseTreeType;
     string_last : integer;
     symtable_last : integer;
```

```
     inttable_last : integer;
     realtable_last : integer;
     tree_last : integer;

     g_cb : integer; (* current block number *)
     g_lb : integer; (* last block number *)
     clevel :integer;

     symtab : ARRAY[0..hlimit] of symtabp;
     g_bsttable : array[maxNest] of BstNodeType;

  function travel_(level:integer):integer;   forward;
  function travel__(level: integer):integer; forward;
  function travel___(level,index : integer):integer; forward;
  function unique_label: integer; forward;
  procedure travel_code_gen(level : integer); forward;
 (*********************************************************)
procedure cleanup;
begin
    close(tables);
end;
  (************************************************)
   procedure error;
    begin
       writeln ('Terminating due to pre-issued error ');
       cleanup;
        HALT
    end;
  (***************************************)
  procedure readSymTable;
    var
    data : string50;

    begin

          while not eof(symf) do
           begin
               readln(symf,data);
               symtable_last := symtable_last +1;
               symtable[symtable_last] := data;
            end;

      close(symf);
  end;
 (*****************************************)
  procedure readStringTable;
  var
    data : string50;

    begin

    while not eof(stringf) do
     begin
        readln(stringf,data);
```

```
        string_last := string_last +1;
        stringtable[string_last] := data;
     end;

    close(stringf);
    end;
(*******************************************)
 procedure readintTable;
   var
    data : string50;
   begin

    while not eof(intf) do
        begin
             readln(intf,data);
             inttable_last := inttable_last +1 ;
             inttable[inttable_last] := data;
        end;

       close(intf);
    end;
 (*******************************************)
  procedure readparsetree;

    var
     j: integer;
     blank :char;
     local_rhsn : integer;
     g1 : integer;
   begin

   while not eof(treef) do
     begin
         read(treef,local_rhsn);
         tree_last := tree_last+1;
         tree[tree_last].rhsn := local_rhsn;
         for j:= 1 to tree[tree_last].rhsn do
             read(treef
                  ,blank
                  ,tree[tree_last].rhstype[j]
                  );
         for j:=1 to tree[tree_last].rhsn do
             read(treef
                  ,blank
                  ,tree[tree_last].rhsindex[j]
                  );
         readln(treef); (* eat eoln mark *)
     end;

    close(treef);

   end;
(*******************************************)
procedure init_global_vars;
```

```
  begin
    g_lb :=0;
    g_cb :=0;
   end;
 (************************************************)
 procedure init;
   var
   counter: integer;
   begin

      string_last :=-1;
      symtable_last :=-1;
      inttable_last :=-1;
      realtable_last :=-1;
      tree_last := -1;

      init_global_vars;
      arr := false;
      unique:= 0;

      clevel :=  0;

      g_bsttable[0].outerblock := -1;
      g_bsttable[0].lexicallevel :=0;
      g_bsttable[0].local_size := 0;
      g_bsttable[0].block_name:= 'outer';

      for counter:=0 to hlimit do
          symtab[counter] := NIL;

      open (treef,file_name :='treef.dat',history:=old); reset(treef);
      open (stringf,file_name:='stringf.dat',history:=old);
reset(stringf);
      open (intf,file_name :='intf.dat',history:=old); reset(intf);
      open (symf,file_name := 'symf.dat',history:=old); reset(symf);
      open (assemply,file_name:='assm.asm',history:=old);
rewrite(assemply);
      open (tables,file_name:='tables.dat',history:=new);
rewrite(tables);
   end;
   (************************************************)
    function resolve_entry_name(level:integer; VAR
which:integer):string50;
   var
     temp : integer;

   begin;
     temp := level;

     if (tree[level].rhstype[1] = subtree ) then (* its proc not pgm *)
       begin
          temp := tree[level].rhsindex[1];
          resolve_entry_name  := symtable[tree[temp].rhsindex[2] ];
          which :=2;
```

```
        end
    else
       if(tree[level].rhstype[1] = token) then
         begin
            resolve_entry_name :=
            symtable [ tree [ tree[level].rhsindex[2] ].rhsindex[1] ];
            which :=1
         end
       else
            begin
                writeln('illegal type in invalide state');
                writeln('error in resolve_entry_name');
                error
            end
   end;
(*************************************************************)
 function hashit (fsym:symbol): integer;
  var n,i:integer;
  begin
     n := 0;
     for i:= 1 to length(fsym) do
          n:= n+ int(fsym[i]);
     n := (128 * n) mod hash_size;
     hashit := n;

(*     hashit:= (128 * n) mod hash_size; *)

  end;
(*************************************************************)
function findsym(fsym:symbol):symtabp;
(* return nil if not found, used to resolve refernces
  IMPORTRANT : object ordered in linked list as deepest lexical level to
               highest so search until lexical level same else return
last
                visited befor that *)
  label 99;
  var sp:symtabp;
      candidate : symtabp;
  begin
      candidate := nil;

      sp:= symtab[hashit(fsym)];

      while sp<> nil do
          begin  (* walk down the hash chain *)
               if sp^.sym=fsym then
                  begin
                       candidate := sp;
                       if sp^.level = g_bsttable[g_cb].lexicallevel then
                          goto 99
                       else
                         sp:= sp^.next
                  end
               else
```

```
                sp := sp^.next
           end; (* while*)
99:
      findsym := candidate;
end;
   (*********************************************)
function  makesym ( fsym: symbol
                    ; syt: vtype
                    ; lev:integer
                    ; id_offset : integer
                    ; id_size : integer (* size of variables in bytes *)
                    ; id_sym :sym_type_
                    ; const_literal : symbol (* for constants *)
                  ): symtabp;
     label 99;
     var sp:symtabp;
         hx: integer;
     begin
         hx := HASHIT(fsym);
         sp:= symtab[hx];
         while sp<> NIL do
         with sp^ do
                   begin
                        if sym=fsym then
                          begin
                            if ( lev = g_bsttable[g_cb].lexicallevel)
                                AND (blk_num = g_cb)  then
                                begin
                                    write('error duplicate declaration
at');
                                    writeln('same lexical level and
block');
                                    error
                                end
                            else
                            ;
                          end
                        else
                         ;
                   sp:=next
                   end;
         new(sp); (* add new entry here *)
         with sp^ do
              begin
                  sem_type := id_sym;
                  sym  := fsym;
                  vtype_ := syt;
                  next := symtab[hx];
                  symtab[hx] := sp;
                  level := lev;
                  if (sem_type = entry) OR (sem_type = constant) then
                     begin
                        id_offset := 0;
                         size := 0;
```

```
                        literal_val := const_literal;
                     end
                  else
                     begin
                        size := id_size;
                        literal_val := '-notused-'
                     end;
                  saddr := id_offset;
                  blk_num := g_cb
               end;
         makesym := sp;
 99:
     end;
(*******************************************************)
procedure clearsym (clevel : integer);
  label 1;
  var hx:integer;
      sp,sptemp:symtabp;

  begin
        (* travel the hash table and get rid of identifirs that
          belong to scope we just left *)

       if clevel <0 then
          clevel:=0
       else
       ;


       for hx:=0 to hlimit do
           begin
               sp:= symtab[hx];
               while sp<> nil do
               with sp^ do
                    begin
                         if level<clevel then
                            goto 1
                         else
                         ;
                         sptemp:=sp;
                         sp:=next;
                         dispose(sptemp);
                    end;
              1:
              symtab[hx] := sp
         end
end;
(*******************************************************)
procedure emit;
  begin
       writeln(assemply,assemply_line);
   end;
(*******************************************************)
procedure subemit;
```

```
  begin
      write(assemply,assemply_line);
  end;
(*************************************************)
procedure subemit_num;
  begin
      write(assemply,assemply_line_number:2);
  end;
(*************************************************)
procedure emit_;
   begin writeln(assemply); end;
(*************************************************)
procedure emit_main_entry(sym : symbol);
begin
   assemply_line := 'st_seq   segment  byte stack ;define stack segment';
emit;
   assemply_line := '         db   20 dup (?)'; emit;
   assemply_line := 'st_seq   ends';            emit;
   assemply_line := ';----------------------------------------'; emit;
   assemply_line := 'code    segment  byte public ; define code seqment';
emit;
   emit_;
   assemply_line := sym + '    proc   far' ; emit;
   assemply_line := '        assume cs:code'; emit;
   assemply_line := 'Start: '; emit;
   assemply_line := '      push ds           ;save old value'; emit;
   assemply_line := '      sub ax,ax         ;put zero in ax '; emit;
   assemply_line := '      push ax           ;save it on stack'; emit;
   assemply_line := ' '; emit; emit_;

end;
(*************************************************)
procedure emit_proc_entry(sym: symbol);
  begin
assemply_line := sym + '  proc   near'; emit;
assemply_line := ' push bp                ;save bp'; emit;
assemply_line := ' mov  bp,sp             ;set up stak frame'; emit;
assemply_line := ' sub  sp,';subemit;
assemply_line_number := g_bsttable[g_cb].local_size ; subemit_num;
assemply_line := '         ;allocate frame'; emit; emit_;

   end;
(*************************************************)
procedure adjust_bst(sym : symbol);
begin

       g_lb := g_lb +1;
       g_bsttable[g_lb].outerblock := g_cb;
       g_cb := g_lb;
       g_bsttable[g_cb].lexicallevel :=
          1+ g_bsttable[g_bsttable[g_cb].outerblock].lexicallevel;
       g_bsttable[g_cb].block_name:= sym;
       g_bsttable[g_cb].block_num := g_cb;
end;
```

```
   (***********************************************)
   procedure  prolog(level: integer);
    var proc_name: symbol;
        symt : vtype;
        literal : symbol;
        sp : symtabp;
        semantic :sym_type_;
        proc_pgm : integer; (* use set later *)
    begin
    proc_pgm :=0;
    literal :='';
    proc_name := resolve_entry_name(level,proc_pgm);

    symt := notused;
    semantic := entry;

    adjust_bst(proc_name);

    sp:= makesym(proc_name
                ,symt              (* symbol type *)
                ,g_bsttable[g_cb].lexicallevel
                ,0
                ,0
                ,semantic
                ,literal);


    end;
   (***********************************************)
   procedure epilog( storage : integer);  (*this is bst epilog pass one
*)
     begin

          g_bsttable[g_cb].local_size := storage;
          g_cb := g_bsttable[g_cb].outerblock;

  end;
(*****************************************************************)
procedure _epilog(level:integer); (* this is the code epilog pass two*)
var proc_pgm :integer;
    proc_name: symbol;
begin
proc_pgm := 0;
proc_name := '-notfound-';

proc_name:= resolve_entry_name(level,proc_pgm);
if proc_pgm = 1 then
   begin
     assemply_line := '     ret     ;go back to OS';  emit;
     assemply_line := proc_name +'      endp'; emit;
     assemply_line := ' ;-----------------------------------'; emit;
emit_;
   end
```

```
else
    begin
      assemply_line := '  mov sp,bp    ;deallocate local variables';
emit;
      assemply_line := '  pop bp       ;restore old value of bp '; emit;
      assemply_line := '  RET '; subemit;
      assemply_line_number := g_bsttable[g_cb].parm_size; subemit_num;
      emit_;
      assemply_line := proc_name +'       endp'; emit;
      assemply_line := ' ;-----------------------------------------';
emit;
      emit_;
    end;

(* clearsym(g_bsttable[g_cb].lexicallevel); (*clean after exit *)
 g_cb := g_bsttable[g_cb].outerblock;

      end;
(*****************************************************)
procedure closing_code;
begin
 assemply_line:=' ;-----------------------------------'; emit;
 assemply_line := 'code ENDS'; emit;
 assemply_line := '     end start' ; emit;
end;
(*****************************************************)
procedure _prolog(level : integer);
 var proc_name:symbol;
     proc_pgm : integer;
 begin
   proc_pgm := 0;
   proc_name := resolve_entry_name(level,proc_pgm);

   if proc_pgm = 1 then
      emit_main_entry(proc_name)
   else
     emit_proc_entry(proc_name)
   ;

   g_lb := g_lb +1;
   g_cb := g_lb;

end;

   (*********************************************)
 procedure receive_parsor_output;
  begin

  readSymTable;
  readIntTable;
  readStringTable;
  readParseTree;

 end;
```

```
(****************************)
function power(wt :integer):integer;
 var i,j: integer;
begin
 j :=1;

  for i:=1 to wt do
      j := j*10;

power :=j;
end;
(******************************************************)
function number( str : string50) :integer;
  var i,j,result,wt,k :integer;
  begin
     result:=0;
     wt :=0;
     i:= 0;
     for j:=1 to length(str) do
         begin
            wt := wt + ((j-1)* 10) ;
            k:= power(j-1);
            result:= result + (int(str[j])-48)*k;
         end;
number:= result;

   end;
(*******************************************************)
procedure gen_ident_ref(level:integer);
var sym : symbol;
    sp : symtabp;
    offset : integer;
    lexical_diff,counter : integer;
(*-----*)
procedure gen1;
    begin
      if LHS= true then
                     begin
                        assemply_line :=' ; resolve lhs reference ';
emit;
                        assemply_line:='     mov ax,BP+'; subemit;
                        assemply_line_number := sp^.saddr; subemit_num;
                        emit_;
    (*                  assemply_line := '     push ax '; emit; *)
                     end
      else
         if lhs= false then
            begin
               assemply_line :='; resolve rhs refernce'; emit;
                assemply_line:='     mov ax,'; subemit;
                assemply_line_number := sp^.saddr; subemit_num;
                assemply_line :='[BP]'; emit;
                assemply_line := '     push ax '; emit;
            end
```

```
        else
        ;
 end;
 (*-----*)
 procedure gen2;
 begin
     if lhs = true then
           if sp^.vtype_ = array_ then
              begin
                   assemply_line:=' ; lhs refernce for array'; emit;
                   assemply_line:='   POP ax  ; get offset '; emit;
                   assemply_line:='   MOV bx,'; subemit;
                   write(assemply,sp^.saddr:2); emit_;
                   assemply_line:='   ADD bx,ax  ; get element offset';
                   emit;
                   assemply_line :='  mov ax,BP'; emit;
                   assemply_line :='  SUB ax,bx'; emit;
              end
           else
              begin
                assemply_line := ' ; resolve lhs refernce '; emit;
                assemply_line := '   mov ax,BP-'; subemit;
                assemply_line_number := sp^.saddr ; subemit_num;
                emit_;
                (* assemply_line := '    push ax'; emit;  *)
             end
    else
       if lhs = false then
             if  sp^.vtype_ = array_ then
              begin
                   assemply_line:=' ; lhs refernce for array'; emit;
                   assemply_line:='   POP ax  ; get offset '; emit;
                   assemply_line:='   MOV bx,'; subemit;
                   write(assemply,sp^.saddr:2); emit_;
                   assemply_line:='   ADD bx,ax  ; get element offset';
                   emit;
                   assemply_line :='  mov ax,-bx[BP]'; emit;
                   assemply_line :='  PUSH ax'; emit;
              end
             else
              begin
                assemply_line:=' ; resolve rhs refernce '; emit;
                assemply_line:='  mov ax,'; subemit;
                assemply_line_number:= -(sp^.saddr); subemit_num;
                assemply_line :='[BP]'; emit;
                assemply_line := '    push ax'; emit;
              end
end;
(*------*)
procedure gen3;
 var sym : symbol;
    begin
      if LHS= true then
         begin
```

```
             writeln (' constant not allowed in LHS');
             error
          end
     else
         begin
           assemply_line :=' ; move number on stack'; emit;
            sym := inttable[(tree[level].rhsindex[1])-1];
            if sym[1] ='$' then
               begin
                   sym[1] := ' ';
                   sym := sym + 'H' ;
               end
            else
            ;
            assemply_line:='     mov ax,'+ sym ; emit;
            assemply_line := '     push ax '; emit;
         end
end;
(*-------*)
procedure gen4; (* call follow up *)
begin
   assemply_line :=' ; generate call argumnet allready on stack ';emit;
   assemply_line :='    CALL ' + sym ; emit;
   emit_;
end;
(*-----*)
procedure gen5; (* outer block ident refernce follow up *)
  begin
    lexical_diff := g_bsttable[g_cb].lexicallevel- sp^.level;
    assemply_line:=' ; refernce variable in outer block'; emit;
    assemply_line := '   mov ax,[BP+4]'; emit;
    for counter := 1 to lexical_diff-1 do
        assemply_line := '  mov ax,[ax+4] ; hup over '; emit;
    if lhs=true then
       begin
           assemply_line:=' ; get the address of outer block variable';
emit;
           assemply_line :='     mov ax,ax-'; subemit;
           assemply_line_number := sp^.saddr; subemit_num;
           emit_;
       end
    else
       if lhs = false then
          begin
             assemply_line :=' ; get the value of outer block variable';
emit;
              assemply_line :='    mov dx,ax  ; save ax'; emit;
              assemply_line :='    mov ax,'; subemit;
              assemply_line_number := -(sp^.saddr); subemit_num;
              assemply_line:='[DX]'; emit;
              assemply_line := '    push ax'; emit;
          end
      else
      ;
```

```
 end;
 (*------*)
begin (* gen ident refr *)
    sym := symtable[tree[level].rhsindex[1]];
    sp := findsym(sym);
    if sp = nil then
         if (sym='chr') or (sym='ord') then
         else
 (*  if (not ((sym[1]='c') and (sym[2]='h') and (sym[3]='r'))) then *)
            begin
              writeln('undeclared ident encountered');
              error
            end
     else
       if sp^.level = g_bsttable[g_cb].lexicallevel then
          case sp^.sem_type of
              parm : gen1 ;
              variable : gen2 ;
              constant : gen3;
              entry : gen4;
              otherwise
                   begin
                   writeln(' unacceptable context for reference ');
                   writeln(' object must be variable or parameter');
                   error
                end
         end
     else
        if sp^.sem_type = entry then
            gen4
        else
            (* it is not in this block  run after it through lex level*)
            gen5 ;
 end;
(*********************************************************)
procedure gen_rhs_real(level : integer);
begin
end;
(*********************************************************)
procedure gen_rhs_int(level: integer);
var sym : symbol;
begin

    sym := inttable[tree[level].rhsindex[1]];
    if sym[1] = '$' then
       begin
         sym[1] := ' ';
         sym := sym + 'H'
       end
    else
    ;
    assemply_line :=' ; push the value of variable on stack'; emit;
    assemply_line :='   mov   ax,' + sym ; emit;
    assemply_line :='   push  ax'; emit;
```

```
    end;
(*********************************************************)
procedure gen_rhs_string(level: integer);
var sym : symbol;
begin
   assemply_line:=' ; push char on stack '; emit;
   assemply_line :='   mov ax,'; subemit;
   sym := stringtable[tree[level].rhsindex[1]];
   assemply_line_number:= int(sym[2]); subemit_num;
   emit_;
   assemply_line :='   PUSH ax'; emit;
   end;
(*********************************************************)
procedure normalize(op :op_type);
var lab1,lab2:integer;
begin
    case op of
     add_ : begin
            assemply_line :=' ; perform addition'; emit;
            assemply_line := '   POP ax'; emit;
            assemply_line := '   POP bx'; emit;
            assemply_line := '   ADD ax,bx'; emit;
            assemply_line := '   push ax'; emit;
            end;
     mul_ : begin
            assemply_line:=' ; perform multiplication'; emit;
            assemply_line := '   POP ax'; emit;
            assemply_line := '   POP bx'; emit;
            assemply_line := '   MUL bx'; emit;
            assemply_line := '   push ax'; emit;
            end;
    assign_: begin    (* note lhs allready in ax *)
            assemply_line :=' ; perform assignment '; emit;
            assemply_line :='    POP bx'; emit;
            assemply_line :='    MOV ax,bx'; emit;
            end;
     le_   : begin

              assemply_line:=' ;---- resolve le '; emit;
              assemply_line:='; leave ax=1 on true, ax=0 on false'; emit;
              assemply_line:='   POP  ax'; emit;
              assemply_line:='   POP  bx'; emit;
              assemply_line:='   CMP  ax,bx'; emit;
              lab1:=unique_label;
              assemply_line:='   JG lab'; subemit;
              write(assemply,lab1:1);
              assemply_line :='   ; jump on greater than '; emit; emit_;

              assemply_line:='   mov  ax,1  ; test passed'; emit;
              lab2 :=unique_label;

              assemply_line :='   JMP lab';subemit;
              write(assemply,lab2:1); emit_;
```

```
            assemply_line :='lab'; subemit;
            write(assemply,lab1:1);
            assemply_line :=':'; emit;
            emit_;

            assemply_line:='    mov ax,0  ;test failed' ; emit;
            assemply_line :='lab'; subemit;
            write(assemply,lab2:1);
            assemply_line :=':'; emit; emit_;

            end;
     sub_: begin
            assemply_line :='  POP ax'; emit;
            assemply_line :='  POP dx'; emit;
            assemply_line :='  SUB ax,bx'; emit;
            assemply_line :='  PUSH ax'; emit;
          end;
     div_ : begin
            assemply_line := ' ; handle division unsigned'; emit;
            assemply_line := '    POP ax'; emit;
            assemply_line := '    POP bx'; emit;
            assemply_line := '    DIV bx  ; ax/bx '; emit;
            assemply_line :='     PUSH ax'; emit;
            end;
      gt_ : begin
            lab1 := unique_label;
            lab2 := unique_label;
            assemply_line :=' ; generate code for gt '; emit;
            assemply_line :='   POP ax'; emit;
            assemply_line :='   POP bx'; emit;
            assemply_line :='   CMP ax,bx'; emit;
            assemply_line :='   JLE lab'; subemit;
            write(assemply,lab1:1); emit_;

            assemply_line :='   mov ax,1  ; test passed'; emit;
            assemply_line := '   JMP lab'; subemit;
            write(assemply,lab2:1); emit_;

            assemply_line :='lab'; subemit;
            write(assemply,lab1:1); assemply_line:=':';emit;

            assemply_line :='    mov ax,0 ; test failed'; emit;

            assemply_line := 'lab'; subemit;
            write(assemply,lab2:1);
            assemply_line:= ':'; emit;
         end;
  end;
end;
(*****************************************************)
function unique_label;  (* :integer *)
begin
   unique := unique +1;
   unique_label := unique;
```

```
 end;
(*******************************************************)
procedure gentest(level: integer);
var op :op_type;
 begin
     case tree[level].rhsn of
     3: if tree[level].rhsindex[1]= int('(')+128 then
           gentest(tree[level].rhsindex[2])
        else
           if tree[level].rhstype[3] = subtree then
              begin
                  travel_code_gen(tree[level].rhsindex[3]);
                  if tree[level].rhstype[1] = subtree then
                     begin
                       travel_code_gen(tree[level].rhsindex[1]);
                       case tree[tree[level].rhsindex[2]].rhsindex[1] of
                       tkle : begin
                              op:= le_;
                              normalize(op);
                              end;
                        end;
                     end
                  else
                  ;
              end
           else
           ;
     end;
 end;
 (*******************************************************)
 procedure gen_body(level:integer);
 begin
     travel_code_gen(level);
 end;
(*******************************************************)
procedure process_others(level:integer);
var
sym : symbol;
lab2,lab1 : integer;
begin

  case tree[level].rhsn of
    4: case tree[level].rhstype[1] of
       token: case tree[level].rhsindex[1] of
              tkwhile: begin
                       lab1 := unique_label;
                       lab2 := unique_label;
                       assemply_line :=' ; code for while stmt'; emit;
                       write(assemply,'lab'); write(assemply,lab1:1);
                       assemply_line :=':'; emit; emit_;

                       assemply_line:=' ;Test for While Loop'; emit;
                       gentest(tree[level].rhsindex[2]);
```

```
                        assemply_line:='   mov dx,1'; emit;
                        assemply_line:='   cmp ax,dx'; emit;
                        assemply_line:='   JL  lab';subemit;
                        write(assemply,lab2:1); emit_;

                        assemply_line:=' ; Body of While Loop'; emit;
                        gen_body(tree[level].rhsindex[4]);

                        assemply_line:='   JMP lab'; subemit;
                        write(assemply,lab1:1);
                        emit_;

                        assemply_line :='lab'; subemit;
                        write(assemply,lab2:1);
                        assemply_line :=':'; emit;
                        emit_;

                        end;
                end;
        end;
  end;
end;
(*************************************************************)
procedure process_if(level :integer);
var lab1,lab2,lab3: integer;
begin
    lab1 := unique_label;
    lab2 := unique_label;
    lab3 := unique_label;

   assemply_line :=' ; if then else stmt '; emit;
   gentest(tree[level].rhsindex[2]);
   assemply_line :='  MOV dx,1'; emit;
   assemply_line :='  cmp ax,dx'; emit;
   assemply_line :='  JL'; subemit;
   write(assemply,lab1:1); emit_;
   gen_body(tree[level].rhsindex[4]);
   assemply_line :='   JMP lab'; subemit;
   write(assemply,lab3:1); emit_;
   assemply_line :='lab:'; subemit;
   write(assemply,lab2:1); emit_;
   gen_body(tree[level].rhsindex[6]);
   assemply_line:='lab:';
   subemit;
   write(assemply,lab3:1); emit_;

   end;

(*************************************************************)
procedure gen_IO;
begin
    assemply_line :='   POP ax  ; get char from the stack '; emit;
    assemply_line :='   mov al,ax'; emit;
    assemply_line := '  mov ah,02'; emit;
```

```
    assemply_line := '  INT doscall ;  output it '; emit;
end;
(*******************************************************)
procedure travel_code_gen; (* (level:integer); *)
 (* this is recursive *)
 var op :op_type;
 begin
       if tree[level].rhstype[1] <> empty then
          begin
            case tree[level].rhsn of
            1: case (tree[level].rhstype[1]) of
                ident: gen_ident_ref(level);
                integer_ident:
                      if (lhs=true) and (arr=false)  then
                          begin
                              writeln('cant have integer in lhs');
                              error
                          end
                     else
                        gen_rhs_int(level);
                real_ident:
                      if lhs=true then
                         begin
                           writeln('cant have real in lhs');
                           error
                         end
                      else
                        gen_rhs_real(level);
                 string_ident:
                       if lhs=true then
                           begin
                            writeln('cant have string in lhs');
                            error
                           end
                       else
                         gen_rhs_string(level);
                 otherwise
                         begin
                          writeln('unexpected type in lhs');
                          error
                         end
                end; (* case 1 *)
              2: if tree[level].rhstype[2] = subtree then
                    begin
                       travel_code_gen(tree[level].rhsindex[2]);
                       if (tree[level].rhstype[1] = subtree) then
                           travel_code_gen(tree[level].rhsindex[1])
                       else
                          if (tree[level].rhsindex[1] = TKWRITE ) then
                              gen_IO
                          else
                          if tree[level].rhsindex[1] = int('-')+128 then
                             begin
                               travel_code_gen(tree[level].rhsindex[2]);
```

```
                                assemply_line:=' ; make negative number';
emit;
                                assemply_line:='  POP ax'; emit;
                                assemply_line :=' mov bx,-1'; emit;
                                assemply_line :=' MUL bx'; emit;
                                assemply_line :=' PUSH ax'; emit;
                              end
                           else
                           begin
                              writeln('unexpcetd type of node in
context');
                              error
                           end
                     end
                   else
                      begin
                          writeln('unacceptable type of node in context');
                          error
                      end;
               3: case tree[level].rhsindex[2] of
                  int(';')+128: begin
                                   travel_code_gen(tree[level].rhsindex[1]);
                                   travel_code_gen(tree[level].rhsindex[3])
                                end;
                  tkasg :  begin
                              op := assign_;
                              travel_code_gen(tree[level].rhsindex[3]);
                              lhs := true;
                              travel_code_gen(tree[level].rhsindex[1]);
                              lhs := false;
                              normalize(op)
                           end;
                  int('*')+128 : begin
                                    op := mul_;
                                    travel_code_gen(tree[level].rhsindex[3]);
                                    travel_code_gen(tree[level].rhsindex[1]);
                                    normalize(op)
                                  end;
                  int('+')+128 : begin
                                    op:= add_;
                                    travel_code_gen(tree[level].rhsindex[3]);
                                    travel_code_gen(tree[level].rhsindex[1]);
                                    normalize(op)
                                 end;
                  int('-')+128: begin
                                    op:= sub_;

travel_code_gen(tree[level].rhsindex[3]);

travel_code_gen(tree[level].rhsindex[1]);
                                     normalize(op);
                                  end;
                otherwise
                    begin
```

```
                      if tree[level].rhsindex[1] = tkbegin then
                           travel_code_gen(tree[level].rhsindex[2])
                      else (* check for argumnet *)
                          if tree[level].rhsindex[1] = int ('(')+128 then
                            travel_code_gen(tree[level].rhsindex[2])
                          else
                             if tree[level].rhsindex[1] = int('[')+128
then
                                 begin
                                   arr := true;

travel_code_gen(tree[level].rhsindex[2]);
                                   arr := false
                                 end
                               else
                                  if tree[level].rhstype[2] = subtree then
                                      begin

travel_code_gen(tree[level].rhsindex[1]);

travel_code_gen(tree[level].rhsindex[3]);
                                      case

tree[tree[level].rhsindex[2]].rhsindex[1]
                                      of int('>')+128: begin
                                                         op:= gt_;
                                                         normalize(op);
                                                       end;
                                      end
                                  end
                               else
                               ;
                    end
                end; (*case 3 *)
            4: process_others(level);
            6: process_if(level);
        end; (* main case *)
        end
    else
    ;

  end;
(*****************************************************)
function ret_array(level :integer) : integer;
 var temp : integer;
   begin
      if tree[level].rhstype[3] = SUBTREE then
          begin (* it in form [0..x] *)
                temp:=tree[tree[level].rhsindex[3]].rhsindex[3];
                ret_array := number(inttable[tree[temp].rhsindex[1]]);
          end
      else
        ret_array :=
             number(inttable[tree[level].rhsindex[3] ])
```

```
   end;
(*********************************************************)
procedure travel_dcl_ (level : integer
                      ;type_ : vtype
                      ;var local_size,size : integer
                      ;semantic :sym_type_
                      );
   var t1,t2 : integer;
             literal : symbol;
             sp : symtabp;
 begin

literal :='';
if tree[level].rhsindex[1] <> 0 then
 BEGIN
   if tree[level].rhsindex[2] = int(';')+128 then
     begin
         travel_dcl_(tree[level].rhsindex[1]
                    ,type_
                    ,local_size
                    ,size
                    ,semantic); (* jump left *)
         travel_dcl_(tree[level].rhsindex[3]
                    ,type_
                    ,local_size
                    ,size
                    ,semantic); (* jump right *)
     end
  else
     begin
         if tree[level].rhsindex[2] = int(':')+128 then
            begin
                t1 := tree[level].rhsindex[3] ;
                if tree[ t1 ].rhstype[1] <> token then
                   begin
                       writeln('error need token type here ');
                       writeln (' error in function collect_');
                       error
                    end
                else
                    if tree[t1].rhstype[1] <> token then
                      begin
                          writeln('type must be token in this context');
                          error
                      end
                    else
                    ;
                    case tree[t1].rhsindex[1] of
                         TKARRAY : begin;
                                     t2 := t1-1;
                                     case tree[t2].rhstype[1] of
                                      TKBYTE: type_:=byte_;
                                      TKINTEGER: type_ :=integer_;
                                      TKBOOLEAN:  type_ :=boolean_;
```

```
                                        TKCHAR : type_ :=char_;
                                       end;
                                       size  := 2* ret_array(t1);
                                      end;
                           TKINTEGER : begin
                                         type_ := integer_;
                                         size := 2;
                                        end;
                           TKBYTE :  begin
                                         type_ := byte_;
                                         size := 2;
                                     end;
                           TKBOOLEAN : begin
                                         type_ := boolean_;
                                         size := 2;
                                       end;
                           TKCHAR : begin
                                         type_ := char_;
                                         size := 2;
                                         end
                     end;
            travel_dcl_(tree[level].rhsindex[1]
                        ,type_
                        ,local_size
                        ,size
                        ,semantic
                        ); (* jump left*)
            end
          else
            begin
                 if tree[level].rhstype[1] = subtree then
                    travel_dcl_(tree[level].rhsindex[1]
                               ,type_
                               ,local_size
                               ,size
                                ,semantic)
                 else
                    if tree[level].rhstype[1] = ident then
                       begin
                          local_size := local_size + size;
                          sp := makesym(symtable[
tree[level].rhsindex[1]]
                                        ,type_
                                        ,g_bsttable[g_cb].lexicallevel
                                        ,local_size
                                        ,size
                                        ,semantic
                                        ,literal )
                       end
                    else
                       begin
                         writeln('error expect an identifire found
another ');
                          writeln('error in collect_');
```

```
                        error
                       end;

               if tree[level].rhsindex[2] = int(',')+128 then
                  if tree[level].rhstype[3] = subtree then
                     travel_dcl_(tree[level].rhsindex[3]
                                , type_
                                , local_size
                                , size
                                ,semantic)
                  else
                     if tree[level].rhstype[3] = ident then
                        begin
                            local_size := local_size + size;
                            makesym(symtable[tree[level].rhsindex[3]]
                                   ,type_
                                   ,g_bsttable[g_cb].lexicallevel
                                   ,local_size
                                   ,size
                                   ,semantic
                                   ,literal)
                        end
                     else
                         begin
                           writeln(' have to be ident or subtree only');
                           error
                          end
               else
               ;
       end;
   end;
 END;
end;
 (*********************************************************)
 procedure travel_dcl(level : integer; var local_size: integer);
     var type_ : vtype ;
         size : integer;
         semantic : sym_type_;
     begin
          type_ := notused;
          semantic := variable;
          size  := 0;
          travel_dcl_ (tree[level].rhsindex[2]
                      ,type_
                      ,local_size
                      ,size
                      ,semantic );
          travel_dcl_ (tree[level].rhsindex[4]
                       ,type_
                       ,local_size
                       ,size
                       ,semantic);
     end;
 (************************************)
```

```
  procedure travel( level : integer);
   (* this is recursive proc *)
   var local_size : integer;
   begin

       local_size :=0;

       if (tree[level].rhstype[1]) <> empty then
          if (tree[tree[level].rhsindex[1]].rhsn = 6 ) then
             begin
                local_size :=travel___(level,1); (* look for dcls *)
                travel  (* goto lower procedures dcls *)

(tree[tree[tree[level].rhsindex[1]].rhsindex[5]].rhsindex[5]);
                epilog(local_size)
             end
          else
             if (tree[tree[level].rhsindex[1]].rhsn=8) then
                begin
                local_size := travel___(level,1);
                travel

(tree[tree[tree[level].rhsindex[1]].rhsindex[7]].rhsindex[5]);
                epilog(local_size)
                end
            else
               if(tree[tree[level].rhsindex[1]].rhsn = 2) then
                  begin
                     travel(tree[level].rhsindex[1]);
                     travel(tree[level].rhsindex[2])
                  end
               else
               ;

      if (tree[level].rhstype[2]) <> empty then
          if (tree[tree[level].rhsindex[2]].rhsn = 6) then
             begin
                local_size :=travel___(level,2);
                travel

(tree[tree[tree[level].rhsindex[2]].rhsindex[5]].rhsindex[5]);
               epilog(local_size);
             end
          else
               if (tree[tree[level].rhsindex[2]].rhsn=8) then
                 begin
                   local_size:= travel___(level,2);
                   travel

(tree[tree[tree[level].rhsindex[2]].rhsindex[7]].rhsindex[5]);
                   epilog(local_size);
                 end
             else
                 if(tree[tree[level].rhsindex[2]].rhsn = 2) then
```

```
                    begin
                       travel(tree[level].rhsindex[1]);
                       travel(tree[level].rhsindex[2])
                    end
                 else
                 ;


end;
(*************************************************************)
function travel___ ;(* (level,index: integer); *)
var local_size : integer;
begin
       local_size := travel_(tree[level].rhsindex[index]);
       travel___ := local_size;
end;
(*************************************************************)
function travel_args(level:integer):integer;
var semantic: sym_type_;
    type_ : vtype;
    size,arg_storage : integer;
 begin
  arg_storage :=0;
  size :=0;
  semantic := parm;
  type_ := notused;

  if tree[level].rhsn <> 1 then
     begin
         if tree[level].rhsindex[1] = int('(')+128 then
            travel_dcl_(tree[level].rhsindex[2]
                        ,type_
                        ,arg_storage
                        ,size
                        ,semantic)
         else
            begin
                 writeln('invalide token in this conext');
                 writeln(' in travel_agrs'); error
            end;
    end
else
;
    travel_args := arg_storage;

end;
  (*************************************************************)
  function travel_ ; (* (level:integer) *)
  begin
      if tree[level].rhstype[1] <> empty then
         if tree[tree[level].rhsindex[1]].rhstype[1] <> empty then
           begin
             prolog(level);
```

```
            g_bsttable[g_cb].parm_size :=
travel_args(tree[level].rhsindex[2]);
            travel_ :=  travel__(tree[level].rhsindex[5])
          end
        else
     else
    ;

end;
(*************************************************************)
procedure travel_const (level: integer);
  var type_ : vtype;
      literal : symbol;
      semantic : sym_type_;
  begin
      type_ := notused;
      semantic := constant;

      if tree[level].rhsindex[2] = int(';')+128 then
         begin
            travel_const(tree[level].rhsindex[1]);
            travel_const(tree[level].rhsindex[3])
         end
      else
         makesym(symtable[tree[level].rhsindex[1]]
               ,type_
               ,g_bsttable[g_cb].lexicallevel
               ,0
               ,number(inttable[tree[level].rhsindex[3]])
               ,semantic
               ,inttable[tree[level].rhsindex[3]]
             );
end;
(*************************************************************)
function travel__ ;  (*  (level:integer)  *)
 var
     local_size,lower_level : integer;
     proc_name : symbol;
     temp :integer;


   begin

     local_size :=0;
     (* see if there are constants will become EQU *)
     temp:= tree[level].rhsindex[2];
     if (tree[temp].rhsindex[1] = TKCONST) then
       travel_const(tree[temp].rhsindex[2])
     else
     ;

     (* see if there are varibles in this proc *)
     temp := tree[level].rhsindex[4] ; (* point to VAR node *)
```

```
     if ( tree[temp].rhsindex[1] = TKVAR) then
         travel_dcl(temp,local_size)
     else
     ;

     travel__ := local_size;

  end;
(***********************************************)
procedure make_outer_level_node;
 begin
    (* build imaginitive outer block for main program so that
       recursion work right *)
    tree[tree_last+1].rhstype[1] := subtree;
    tree[tree_last+1].rhstype[2] := subtree;
    tree[tree_last+1].rhsn :=2;
    (* now point this to the main *)
    tree[tree_last+1].rhsindex[1] := tree_last;
    tree[tree_last+1].rhsindex[2] := tree_last+2;
    tree[tree_last+2].rhstype[1] := EMPTY;
    tree[tree_last+2].rhsindex[1] := 9999;
    tree[tree_last+2].rhsn :=1;

 end;
 (*************************************)
  procedure build_global_symbol_table;
   var proc_name : symbol;
       symt : vtype;
   begin

    make_outer_level_node;
      (* now start travering the tree *)
    travel(tree_last+1);

   end;
(*****************************************************)
procedure print_token(a : integer);
begin
    case a of
      tkchar:   write(tables,'CHAR  ');
      TKASG:     write(tables,':=  ')  ;
      TKNE:      write(tables,'NE  ')  ;
      TKLE:      write(tables,'LE  ')  ;
      TKGE:      write(tables,'GE  ')  ;
      TKDTDT:    write(tables,'..  ')  ;
      TKABSOLUTE:      write(tables,'ABSOLUTE  ')  ;
      TKAND:     write(tables,'AND  ')  ;
      TKARRAY:   write(tables,'ARRAY  ')  ;
      TKBEGIN:   write(tables,'BEGIN  ')  ;
      TKCASE :   write(tables,'   ')  ;
        TKCONST: write(tables,'CONST  ')  ;
        TKDIV:   write(tables,'DIV  ')  ;
        TKDO:    write(tables,'DO  ')  ;
        TKDOWNTO:      write(tables,'DOWNTO  ')  ;
```

```
        TKELSE:  write(tables,'ELSE  ')  ;
        TKEND:   write(tables,'END  ')  ;
        TKEXTERNAL:    write(tables,'EXTERNAL  ')  ;
        TKFILE:  write(tables,'FILE  ')  ;
        TKFORWARD:     write(tables,'FORWARD  ')  ;
        TKFOR:   write(tables,'FOR  ')  ;
        TKFUNCTION:    write(tables,'FUNCTION  ')  ;
        TKGOTO:  write(tables,'GOTO  ')  ;
        TKINLINE:      write(tables,'INLINE  ')  ;
        TKIF:    write(tables,'IF  ')  ;
        TKIN:    write(tables,'IN  ')  ;
        TKLABEL: write(tables,'LABEL  ')  ;
        TKMOD:   write(tables,'MOD  ')  ;
        TKNIL:   write(tables,'NIL  ')  ;
        TKNOT:   write(tables,'NOT  ')  ;
        TKOVERLAY:     write(tables,'OVERLAY  ')  ;
        TKOF:    write(tables,'OF  ')  ;
        TKOR:    write(tables,'OR  ')  ;
        TKPACKED:      write(tables,'PACKED  ')  ;
        TKPROCEDURE: write(tables,'PROCEDURE  ')  ;
        TKPROGRAM:     write(tables,'PROGRAM  ')  ;
        TKRECORD:      write(tables,'RECORD  ')  ;
        TKREPEAT:      write(tables,'REPEAT  ')  ;
        TKSET:   write(tables,'SET  ')  ;
        TKSHL:   write(tables,'SHL  ')  ;
        TKSHR:   write(tables,'SHR  ')  ;
        TKSTRING:      write(tables,'STRING  ')  ;
        TKTHEN:  write(tables,'THEN  ')  ;
        TKTYPE:  write(tables,'TYPE  ')  ;
        TKTO:    write ('TO  ')  ;
        TKUNTIL: write(tables,'UNTIL  ')  ;
        TKVAR:   write(tables,'VAR  ')  ;
        TKWHILE: write(tables,'WHILE  ')  ;
        TKWITH:  write(tables,'WITH  ')  ;
        TKXOR:   write(tables,'XOR  ')  ;
        TKREAL:  write(tables,'REAL  ')  ;
        TKBOOLEAN:     write(tables,'BOOLEAN  ')  ;
        TKINTEGER:     write(tables,'INTEGER  ')  ;
        TKREAD:  write(tables,'READ  ')  ;
        TKWRITE: write(tables,'WRITE  ')  ;
        TKTRUE:  write(tables,'TRUE  ')  ;
        TKFALSE: write ('FALSE  ')  ;
        TKWRITELN:  write(tables,'WRITELN ')  ;
        TKREADLN: write(tables,'READLN ')  ;
        TKBYTE:    write(tables,'BYTE ')  ;
        otherwise
          begin
             write ('error in write token unknown token number') ;
            error;
         end
    end
end;
(*****************************************************)
procedure dump_indx(i,j : integer);
```

```
begin
               case tree[i].rhstype[j] of
                   SUBTREE:
                        write (tables,'(',tree[i].rhsindex[j]:1,')');
                   LITERAL:
                       write (tables,(tree[i].rhsindex[j]-128):1);
                   IDENT:
                        write (tables,symtable[tree[i].rhsindex[j]]:1);
                   INTEGER_IDENT:
                        write (tables,inttable[tree[i].rhsindEx[j]]:1);
                   TOKEN:
                        print_token(tree[i].rhsindex[j]);
                   STRING_IDENT:
                        write
(tables,stringtable[tree[i].rhsindex[j]]:1);
                   EMPTY:
                        write(tables,'*empty*');
                   otherwise
                        begin
                        writeln(' error cannot recorgize type in tree');
                        error;
                        end;
                 end;
end;
 (*********************************************************)
procedure dump_parse_tree;
 var i,j : integer;

 begin
writeln(tables);
writeln(tables,'       P   A    R   S   E          T   A   B  L  E ');

    for i:=0 to tree_last do
       begin
           WRITE(tables,i,'. ');
           for j:=1 to tree[i].rhsn do
               case tree[i].rhstype[j] of

               SUBTREE :      write(tables,' subtree ');
               LITERAL :      write(tables,' literal ');
               IDENT   :      write(tables,' ident ');
               INTEGER_IDENT: write(tables,' int_idnt ');
               REAL_IDENT:    write(tables,' real_idnt ');
               TOKEN :        write(tables,' token ');
               STRING_IDENT:  write(tables,' string ');
               EMPTY :        write(tables,' EMPTY ');
               otherwise
                    begin
                        writeln;
                        writeln('dont understand this type in
pase_tree');
                        writeln('error in dump tree');
                        error;
                    end;
```

```
              end; (*case*)
           writeln(tables);
          write(tables,'             ');
          for j:=1 to tree[i].rhsn do
              begin
                dump_indx(i,j);
                write(tables,'    ');
             end;
       writeln(tables);
      end;

writeln(tables,'** end of parse table **');
end;
(*********************************************************************)
procedure dump_bst_table;
  var i,j : integer;
  begin

 writeln(tables,'                B  S  T       T A B L E   ');
 writeln(tables,' Index  Entry_name   lex_level  Outer  localsize
parmsize');
     for i:=1 to g_lb do
          begin
               write(tables
                    ,i:5
                    ,g_bsttable[i].block_name:10
                    ,g_bsttable[i].lexicallevel:12
                    ,g_bsttable[i].outerblock:8
                    ,g_bsttable[i].local_size:10
                    ,g_bsttable[i].parm_size:10
                    );
               writeln(tables)
          end;
  writeln(tables,'** end of bst tables **');
end;
(*******************************************************)
procedure dump_symbol_table;

 var sp: symtabp;
     i: integer;

(*-----*)
   procedure dump(sp : symtabp);
     begin
          with sp^ do
              begin
                  write(tables,sym:13,Level:5,saddr:9,size:10);
                  case vtype_ of
                     byte_ : write(tables,'   BYTE');
                     integer_: write(tables,'   INTEGER');
                     boolean_ :write(tables,'   BOOL');
                     char_ :write(tables,'   CHAR');
                     array_ :write(tables,'   ARRAY');
```

```
                    notused: write(tables,'     n/a');
                    otherwise
                       begin
                         writeln('dont undertand ident type in dump
symtable');
                         error
                       end;
                    end; (* case *)
                 case sem_type of
                    entry: write(tables,'    ENTRY');
                    parm     :write(tables,'     PARM');
                    constant :write(tables,'     CONST');
                    variable : write(tables,'    VAR');
                 end;
                 write(tables,literal_val:10);
                 write(tables,blk_num:8);
                 writeln(tables); writeln(tables);
              end;
 end;
(*----*)

 begin
 writeln(tables,'              S Y M B O L      T A B L E');
 writeln(tables,'symbol    Level  Offset  size(equ)  data   Type  literal
Blk_num');

 for i:=0 to hlimit do
     begin
          sp:=symtab[i];
          while sp<> NIL do
               Begin
                    dump(sp);
                    sp:=sp^.next
               end;
     end;
 end;
(*********************************************************)
procedure generate_code___ (level,index:integer);
  var temp:integer;
 begin
      temp:= tree[level].rhsindex[index];
      if tree[temp].rhstype[1] <> empty then
         if tree[tree[temp].rhsindex[1]].rhstype[1] <> empty then
             begin
               _prolog(temp);
               temp:=

tree[tree[tree[level].rhsindex[index]].rhsindex[5]].rhsindex[7];
               LHS := false;
               travel_code_gen (temp);
            end

end;
```

```
(*************************************************************)
procedure generate_code (level:integer);
 (* this is recursive *)
 var temp: integer;
 begin

   if (tree[level].rhstype[1]) <> empty then
      begin
           if tree[level].rhsindex[1] <> 0 then
               begin
                   if (tree[tree[level].rhsindex[1]].rhsn = 6 ) then
                      begin
                          generate_code___ (level,1);
                          _epilog(tree[level].rhsindex[1]);
                          temp:= tree[level].rhsindex[1];
                          temp:= tree[temp].rhsindex[5];
                          temp:= tree[temp].rhsindex[5];
                          generate_code(temp)
                      end
                   else
                      begin
                        if(tree[tree[level].rhsindex[1]].rhsn = 2 ) then
                          begin
                              generate_code(tree[level].rhsindex[1]);
                              generate_code(tree[level].rhsindex[2])
                          end
                        else
                        ;
                      end
               end
         else
         ;

        if (tree[level].rhstype[2]) <> empty then
          if tree[level].rhsindex[2] <> 0 then
             if (tree[tree[level].rhsindex[2]].rhsn =6) then
                begin
                    generate_code___(level,2);
                    _epilog(tree[level].rhsindex[2]);
                    temp:= tree[level].rhsindex[2];
                    temp := tree[temp].rhsindex[5];
                    temp := tree[temp].rhsindex[5];
                    generate_code(temp)
                end
            else
               if (tree[tree[level].rhsindex[2]].rhsn = 2) then
                  begin
                      generate_code(tree[level].rhsindex[1]);
                      generate_code(tree[level].rhsindex[2])
                  end
               else
                ;
      end
  else
```

```
  ;
end;

(*************************************************************)
procedure global_generate_code; (* i come here after initial tree
traversal
                                where symtable and bst have been
constructed *)
 begin
     init_global_vars;
     make_outer_level_node;
     generate_code(tree_last+1);
   end;
(*************************************************************)
procedure generate_EQU;
 var sp : symtabp;
     sym1 : symbol;
     hx : integer;

 begin
    assemply_line :=' ; OUTPUT of pascal compiler by Naser Abbasi'; emit;
    assemply_line :=' ; CSE565 Oakland University  April 1988'; emit;
    for hx:=0 to hlimit do
     begin
        sp:=symtab[hx];
          while sp<>nil do
           begin
                if sp^.sem_type=constant then
                   begin
                     sym1 := sp^.literal_val;
                     (* handel hex values *)
                     if sym1[1]='$' then
                        begin
                           sym1[1] :=' ';
                           sym1 := sym1 + 'H'
                        end
                     else
                      ;
                     assemply_line:=' '+ sp^.sym+' EQU  '  + sym1;
                     emit;
                   end
                else
                ;

               sp:= sp^.next
          end;
     end;
 assemply_line :='          mov   al,ax'; emit;
 assemply_line :='          mov   ah,02'; emit;
 assemply_line :=' doscall  EQU   21h  ; dos interupt routine'; emit;
 emit_;

end;
 (*************************************************************)
```

```
 (*  M A I N    L I N E    S T A R T S   H E R E          *)
begin

  debug := true;

  init;

  receive_parsor_output;
  if tree_last=-1  then begin
     writeln(' parse tree was empty ');
     error
     end
  else
  ;

  build_global_symbol_table;

  if debug then
     BEGIN
         dump_symbol_table;
         dump_bst_table;
         dump_parse_tree
     END
   else
   ;

   generate_EQU;
   global_generate_code;
   closing_code;
   cleanup;

end.



(*  The following is final result for TEST1 including
    - parse tree
    - symbol table
    - BST table
    - ASSEMPLY output
    - listing and map from succsesful assemply on IBM pc

(*************************************************************)
 ; OUTPUT of pascal compiler by Naser Abbasi
 ; CSE565 Oakland University  April 1988
 lf EQU   0aH
 cr EQU   0dH
 doscall  EQU   21h  ; dos interupt routine

st_seq   segment  byte stack ;define stack segment
         db   20 dup (?)
st_seq   ends
;-----------------------------------------
code    segment  byte public ; define code seqment
```

```
test1    proc   far
        assume cs:code
Start:
      push ds           ;save old value
      sub ax,ax         ;put zero in ax
      push ax            ;save it on stack


 ; move number on stack
     mov ax, 0dH
     push ax
 ; resolve lhs refernce
   mov ax,BP- 4
 ; perform assignment
    POP bx
    MOV ax,bx
 ; resolve rhs refernce
  mov ax,-4[BP]
    push ax
 ; generate call argumnet allready on stack
    CALL putc

 ; move number on stack
     mov ax, 0aH
     push ax
 ; resolve lhs refernce
   mov ax,BP- 4
 ; perform assignment
    POP bx
    MOV ax,bx
 ; resolve rhs refernce
  mov ax,-4[BP]
    push ax
 ; generate call argumnet allready on stack
    CALL putc

 ; push char on stack
   mov ax,72
   PUSH ax
 ; generate call argumnet allready on stack
    CALL putc

 ; push char on stack
   mov ax,105
   PUSH ax
 ; generate call argumnet allready on stack
    CALL putc

 ; push char on stack
   mov ax,33
   PUSH ax
 ; generate call argumnet allready on stack
    CALL putc
```

```
 ; move number on stack
     mov ax, 0dH
     push ax
 ; resolve lhs refernce
   mov ax,BP- 4
 ; perform assignment
    POP bx
    MOV ax,bx
 ; resolve rhs refernce
  mov ax,-4[BP]
    push ax
 ; generate call argumnet allready on stack
    CALL putc

 ; move number on stack
     mov ax, 0aH
     push ax
 ; resolve lhs refernce
   mov ax,BP- 4
 ; perform assignment
    POP bx
    MOV ax,bx
 ; resolve rhs refernce
  mov ax,-4[BP]
    push ax
 ; generate call argumnet allready on stack
    CALL putc

 ; push the value of variable on stack
   mov   ax,0
   push  ax
 ; resolve lhs refernce
   mov ax,BP- 2
 ; perform assignment
    POP bx
    MOV ax,bx
 ; code for while stmt
lab1:

 ;Test for While Loop
 ; push the value of variable on stack
   mov   ax,9
   push  ax
 ; resolve rhs refernce
  mov ax,-2[BP]
    push ax
 ;---- resolve le
; leave ax=1 on true, ax=0 on false
   POP  ax
   POP  bx
   CMP  ax,bx
   JG lab3   ; jump on greater than
```

```
   mov  ax,1  ; test passed
   JMP lab4
lab3:

    mov ax,0  ;test failed
lab4:

   mov dx,1
   cmp ax,dx
   JL  lab2
 ; Body of While Loop
 ; push the value of variable on stack
   mov   ax, 30H
   push  ax
 ; resolve rhs refernce
  mov ax,-2[BP]
    push ax
 ; perform addition
   POP ax
   POP bx
   ADD ax,bx
   push ax
 ; resolve lhs refernce
   mov ax,BP- 4
 ; perform assignment
    POP bx
    MOV ax,bx
 ; resolve rhs refernce
  mov ax,-4[BP]
    push ax
 ; generate call argumnet allready on stack
    CALL putc

 ; push the value of variable on stack
   mov   ax,1
   push  ax
 ; resolve rhs refernce
  mov ax,-2[BP]
    push ax
 ; perform addition
   POP ax
   POP bx
   ADD ax,bx
   push ax
 ; resolve lhs refernce
   mov ax,BP- 2
 ; perform assignment
    POP bx
    MOV ax,bx
   JMP lab1
lab2:

 ; move number on stack
     mov ax, 0dH
```

```
     push ax
 ; resolve lhs refernce
   mov ax,BP- 4
 ; perform assignment
    POP bx
    MOV ax,bx
 ; resolve rhs refernce
  mov ax,-4[BP]
    push ax
 ; generate call argumnet allready on stack
    CALL putc

 ; move number on stack
     mov ax, 0aH
     push ax
 ; resolve lhs refernce
   mov ax,BP- 4
 ; perform assignment
    POP bx
    MOV ax,bx
 ; resolve rhs refernce
  mov ax,-4[BP]
    push ax
 ; generate call argumnet allready on stack
    CALL putc

     ret     ;go back to OS
test1      endp
 ;-------------------------------------

putc  proc   near
 push bp                ;save bp
 mov  bp,sp             ;set up stak frame
 sub  sp, 0         ;allocate frame

; resolve rhs refernce
     mov ax, 2[BP]
     push ax
   POP ax  ; get char from the stack
  mov al,ax
  mov ah,02
  INT doscall ;  output it
  mov sp,bp    ;deallocate local variables
  pop bp       ;restore old value of bp
  RET  2
putc       endp
 ;-----------------------------------------

 ;-------------------------------------
code ENDS
     end start

(*************************************************************)
```

```
                S Y M B O L      T A B L E
symbol Level  Offset  size(equ)  data   Type  literal     Blk_num
  c    2        2         2     CHAR    PARM  -notused-       2

  y    1        4         2     CHAR    VAR   -notused-       1

 lf    1        0         0     n/a     CONST       $0a       1

put    2        0         0     n/a     ENTRY                 2

  x    1        2         2    BYTE     VAR   -notused-       1

test   1        0         0     n/a     ENTRY                  1

 cr    1        0         0     n/a     CONST       $0d       1


                B  S  T       T A B L E
 Index  Entry_name   lex_level  Outer  localsize  parmsize
    1     test1           1       0         4         0
    2      putc           2       1         0         2

** end of bst tables **

(*****************************************************************)
IBM Personal Computer MACRO Assembler   Version 2.00     Page  1-1
                                              11-02-92


1                              ; OUTPUT of pascal compiler by Naser A
                       bbasi
2                              ; CSE565 Oakland University  April 198
                       8
3      = 000A                        lf EQU   0aH
4      = 000D                        cr EQU   0dH
5      = 0021                        doscall  EQU   21h  ; dos interupt rou
                       tine
6
7      0000                    st_seq   segment  byte stack ;define st
                       ack segment
8      0000     14 [                          db   20 dup (?)
9                      ??
10                        ]
11
12     0014                    st_seq   ends
13                             ;-------------------------------------
                       -----
14     0000                    code    segment  byte public ; define c
                       ode seqment
15
16     0000                    test1    proc   far
17                                     assume cs:code
18     0000                    Start:
19     0000  1E                      push ds           ;save old value
```

```
20     0001  2B C0                    sub ax,ax         ;put zero in ax

21     0003  50                      push ax            ;save it on st
                     ack
22
23
24                                 ; move number on stack
25     0004  B8 000D                       mov ax, 0dH
26     0007  50                     push ax
27                                 ; resolve lhs refernce
28     0008  8B C1                        mov ax,BP- 4
29                                 ; perform assignment
30     000A  5B                     POP bx
31     000B  8B C3                       MOV ax,bx
32                                 ; resolve rhs refernce
33     000D  8B 46 FC                   mov ax,-4[BP]
34     0010  50                     push ax
35                                 ; generate call argumnet allready on s
                     tack
36     0011  E8 00CE R              CALL putc
37
38                                 ; move number on stack
39     0014  B8 000A                       mov ax, 0aH
40     0017  50                     push ax
41                                 ; resolve lhs refernce
42     0018  8B C1                        mov ax,BP- 4
43                                 ; perform assignment
44     001A  5B                     POP bx
45     001B  8B C3                       MOV ax,bx
46                                 ; resolve rhs refernce
47     001D  8B 46 FC                   mov ax,-4[BP]
48     0020  50                     push ax
49                                 ; generate call argumnet allready on s
                     tack
50     0021  E8 00CE R              CALL putc
51
52                                 ; push char on stack
53     0024  B8 0048                      mov ax,72
54     0027  50                     PUSH ax
55                                 ; generate call argumnet allready on s
                     tack
56     0028  E8 00CE R              CALL putc
57
58                                 ; push char on stack
59     002B  B8 0069                      mov ax,105
60     002E  50                     PUSH ax
61                                 ; generate call argumnet allready on s
                     tack
62     002F  E8 00CE R              CALL putc
63
64                                 ; push char on stack
65     0032  B8 0021                      mov ax,33
66     0035  50                     PUSH ax
67                                 ; generate call argumnet allready on s
```

```
                        tack
68     0036  E8 00CE R          CALL putc
69
70                              ; move number on stack
71     0039  B8 000D                    mov ax, 0dH
72     003C  50                  push ax
73                              ; resolve lhs refernce
74     003D  8B C1                     mov ax,BP- 4
75                              ; perform assignment
76     003F  5B                  POP bx
77     0040  8B C3                     MOV ax,bx
78                              ; resolve rhs refernce
79     0042  8B 46 FC                mov ax,-4[BP]
80     0045  50                  push ax
81                              ; generate call argumnet allready on s
                        tack
82     0046  E8 00CE R          CALL putc
83
84                              ; move number on stack
85     0049  B8 000A                    mov ax, 0aH
86     004C  50                  push ax
87                              ; resolve lhs refernce
88     004D  8B C1                     mov ax,BP- 4
89                              ; perform assignment
90     004F  5B                  POP bx
91     0050  8B C3                     MOV ax,bx
92                              ; resolve rhs refernce
93     0052  8B 46 FC                mov ax,-4[BP]
94     0055  50                  push ax
95                              ; generate call argumnet allready on s
                        tack
96     0056  E8 00CE R          CALL putc
97
98                              ; push the value of variable on stack
99     0059  B8 0000                   mov   ax,0
100    005C  50                  push  ax
101                             ; resolve lhs refernce
102    005D  8B C3                     mov ax,BP- 2
103                             ; perform assignment
104    005F  5B                  POP bx
105    0060  8B C3                     MOV ax,bx
106                             ; code for while stmt
107    0062                    lab1:
108
109                             ;Test for While Loop
110                             ; push the value of variable on stack
111    0062  B8 0009                   mov   ax,9
112    0065  50                  push  ax
113                             ; resolve rhs refernce
114    0066  8B 46 FE                mov ax,-2[BP]
115    0069  50                  push ax
116                             ;---- resolve le
117                             ; leave ax=1 on true, ax=0 on false
118    006A  58                  POP  ax
```

```
119    006B  5B                  POP  bx
120    006C  3B C3                    CMP  ax,bx
121    006E  7F 06                    JG lab3   ; jump on greater than
122
123    0070  B8 0001                  mov  ax,1  ; test passed
124    0073  EB 04 90                 JMP lab4
125    0076                  lab3:
126
127    0076  B8 0000                   mov ax,0  ;test failed
128    0079                  lab4:
129
130    0079  BA 0001                  mov dx,1
131    007C  3B C2                    cmp ax,dx
132    007E  7C 2D                    JL  lab2
133                          ; Body of While Loop
134                          ; push the value of variable on stack
135    0080  B8 0030                  mov   ax, 30H
136    0083  50                    push  ax
137                          ; resolve rhs refernce
138    0084  8B 46 FE                mov ax,-2[BP]
139    0087  50                     push ax
140                          ; perform addition
141    0088  58                    POP ax
142    0089  5B                    POP bx
143    008A  03 C3                    ADD ax,bx
144    008C  50                    push ax
145                          ; resolve lhs refernce
146    008D  8B C1                    mov ax,BP- 4
147                          ; perform assignment
148    008F  5B                     POP bx
149    0090  8B C3                     MOV ax,bx
150                          ; resolve rhs refernce
151    0092  8B 46 FC                mov ax,-4[BP]
152    0095  50                     push ax
153                          ; generate call argumnet allready on s
                       tack
154    0096  E8 00CE R               CALL putc
155
156                          ; push the value of variable on stack
157    0099  B8 0001                  mov   ax,1
158    009C  50                    push  ax
159                          ; resolve rhs refernce
160    009D  8B 46 FE                mov ax,-2[BP]
161    00A0  50                     push ax
162                          ; perform addition
163    00A1  58                    POP ax
164    00A2  5B                    POP bx
165    00A3  03 C3                    ADD ax,bx
166    00A5  50                    push ax
167                          ; resolve lhs refernce
168    00A6  8B C3                    mov ax,BP- 2
169                          ; perform assignment
170    00A8  5B                     POP bx
171    00A9  8B C3                     MOV ax,bx
```

```
172    00AB  EB B5                    JMP lab1
173    00AD                  lab2:
174
175                           ; move number on stack
176    00AD  B8 000D                    mov ax, 0dH
177    00B0  50                      push ax
178                           ; resolve lhs refernce
179    00B1  8B C1                    mov ax,BP- 4
180                           ; perform assignment
181    00B3  5B                     POP bx
182    00B4  8B C3                     MOV ax,bx
183                           ; resolve rhs refernce
184    00B6  8B 46 FC                mov ax,-4[BP]
185    00B9  50                     push ax
186                           ; generate call argumnet allready on s
                      tack
187    00BA  E8 00CE R              CALL putc
188
189                           ; move number on stack
190    00BD  B8 000A                    mov ax, 0aH
191    00C0  50                      push ax
192                           ; resolve lhs refernce
193    00C1  8B C1                    mov ax,BP- 4
194                           ; perform assignment
195    00C3  5B                     POP bx
196    00C4  8B C3                     MOV ax,bx
197                           ; resolve rhs refernce
198    00C6  8B 46 FC                mov ax,-4[BP]
199    00C9  50                     push ax
200                           ; generate call argumnet allready on s
                      tack
201    00CA  E8 00CE R              CALL putc
202
203    00CD  CB                      ret     ;go back to OS
204    00CE                  test1      endp
205                           ;-------------------------------------
206
207    00CE                  putc  proc   near
208    00CE  55                     push bp                ;save bp
209    00CF  8B EC                    mov  bp,sp              ;set up stak f
                      rame
210    00D1  83 EC 00                 sub  sp, 0         ;allocate frame
211
212                           ; resolve rhs refernce
213    00D4  8B 46 02                   mov ax, 2[BP]
214    00D7  50                      push ax
215    00D8  58                    POP ax  ; get char from the stack
                                          int doscall ;  output it
217    00DB  8B E5                   mov sp,bp    ;deallocate local variab
                      les
218    00DD  5D                     pop bp       ;restore old value of bp

219    00DE  C2 0002                  RET  2
220    00E1                  putc       endp
```

```
221                          ;-----------------------------------
                    -------
222
223                          ;-----------------------------------
224    00E1                 code ENDS
225                              end start


Segments and Groups:

           N a m e               Size  Align Combine    Class

CODE . . . . . . . . . . . . .  00E1  BYTE       PUBLIC
ST_SEQ . . . . . . . . . . . .  0014  BYTE       STACK

Symbols:

           N a m e               Type  Value Attr

CR . . . . . . . . . . . . . .  Number     000D
DOSCALL. . . . . . . . . . . .  Number     0021
LAB1 . . . . . . . . . . . . .  L NEAR     0062  CODE
LAB2 . . . . . . . . . . . . .  L NEAR     00AD  CODE
LAB3 . . . . . . . . . . . . .  L NEAR     0076  CODE
LAB4 . . . . . . . . . . . . .  L NEAR     0079  CODE
LF . . . . . . . . . . . . . .  Number     000A
PUTC . . . . . . . . . . . . .  N PROC     00CE  CODE  Length =0013
START. . . . . . . . . . . . .  L NEAR     0000  CODE
TEST1. . . . . . . . . . . . .  F PROC     0000  CODE  Length =00CE

50096 Bytes free

Warning Severe
Errors     Errors
0     0

(*********************************************************************)


 Start  Stop   Length  Name                   Class
 00000H 000E0H 00E1H  CODE
 000F0H 00103H 0014H  ST_SEQ

 Origin   Group

Program entry point at 0000:0000

(*********************************************************************)



(*************  T E S T 2    problem  *********************)
```

```
 ; OUTPUT of pascal compiler by Naser Abbasi
 ; CSE565 Oakland University  April 1988
 lf EQU   0aH
 cr EQU   0dH
 doscall  EQU   21h  ; dos interupt routine

st_seq   segment  byte stack ;define stack segment
         db   20 dup (?)
st_seq   ends
;-----------------------------------------
code    segment  byte public ; define code seqment

test2    proc   far
        assume cs:code
Start:
      push ds           ;save old value
      sub ax,ax         ;put zero in ax
      push ax            ;save it on stack


 ; push the value of variable on stack
   mov   ax,5
   push  ax
 ; push the value of variable on stack
   mov   ax,0
   push  ax
 ; resolve lhs refernce
   mov ax,BP- 6
 ; perform assignment
    POP bx
    MOV ax,bx
 ; push the value of variable on stack
   mov   ax,2
   push  ax
 ; push the value of variable on stack
   mov   ax,1
   push  ax
 ; resolve lhs refernce
   mov ax,BP- 6
 ; perform assignment
    POP bx
    MOV ax,bx
 ; push the value of variable on stack
   mov   ax,1
   push  ax
 ; push the value of variable on stack
   mov   ax,1
   push  ax
 ; make negative number
  POP ax
 mov bx,-1
 MUL bx
 PUSH ax
 ; push the value of variable on stack
```

```
   mov   ax,2
   push  ax
 ; resolve lhs refernce
   mov ax,BP- 6
 ; perform assignment
    POP bx
    MOV ax,bx
 ; push the value of variable on stack
   mov   ax,1
   push  ax
 ; push the value of variable on stack
   mov   ax,3
   push  ax
 ; resolve lhs refernce
   mov ax,BP- 6
 ; perform assignment
    POP bx
    MOV ax,bx
 ; generate call argumnet allready on stack
    CALL newline

 ; push the value of variable on stack
   mov   ax,0
   push  ax
 ; resolve lhs refernce
   mov ax,BP- 4
 ; perform assignment
    POP bx
    MOV ax,bx
 ; code for while stmt
lab1:

 ;Test for While Loop
 ; push the value of variable on stack
   mov   ax,20
   push  ax
 ; resolve rhs refernce
  mov ax,-4[BP]
    push ax
 ;---- resolve le
; leave ax=1 on true, ax=0 on false
   POP  ax
   POP  bx
   CMP  ax,bx
   JG lab3   ; jump on greater than

   mov  ax,1  ; test passed
   JMP lab4
lab3:

    mov ax,0  ;test failed
lab4:

   mov dx,1
```

```
   cmp ax,dx
   JL  lab2
 ; Body of While Loop
 ; resolve rhs refernce
  mov ax,-4[BP]
    push ax
 ; generate call argumnet allready on stack
    CALL prnum

 ; push the value of variable on stack
   mov   ax,0
   push  ax
 ; resolve lhs refernce
   mov ax,BP- 6
 ; perform assignment
    POP bx
    MOV ax,bx
 ; push the value of variable on stack
   mov   ax,3
   push  ax
 ; resolve lhs refernce
   mov ax,BP- 2
 ; perform assignment
    POP bx
    MOV ax,bx
 ; code for while stmt
lab5:

 ;Test for While Loop
 ; push the value of variable on stack
   mov   ax,0
   push  ax
 ; resolve rhs refernce
  mov ax,-2[BP]
    push ax
   mov dx,1
   cmp ax,dx
   JL  lab6
 ; Body of While Loop
 ; resolve rhs refernce
  mov ax,-2[BP]
    push ax
 ; resolve rhs refernce
  mov ax,-6[BP]
    push ax
 ; resolve rhs refernce
  mov ax,-4[BP]
    push ax
 ; resolve rhs refernce
  mov ax,-6[BP]
    push ax
 ; perform multiplication
   POP ax
   POP bx
```

```
   MUL bx
   push ax
 ; perform addition
   POP ax
   POP bx
   ADD ax,bx
   push ax
 ; resolve lhs refernce
   mov ax,BP- 6
 ; perform assignment
    POP bx
    MOV ax,bx
 ; resolve lhs refernce
   mov ax,BP- 2
 ; perform assignment
    POP bx
    MOV ax,bx
   JMP lab5
lab6:

 ; resolve rhs refernce
  mov ax,-6[BP]
    push ax
 ; generate call argumnet allready on stack
    CALL prnum

 ; generate call argumnet allready on stack
    CALL newline

 ; push the value of variable on stack
   mov   ax,1
   push  ax
 ; resolve rhs refernce
  mov ax,-4[BP]
    push ax
 ; perform addition
   POP ax
   POP bx
   ADD ax,bx
   push ax
 ; resolve lhs refernce
   mov ax,BP- 4
 ; perform assignment
    POP bx
    MOV ax,bx
   JMP lab1
lab2:

 ; generate call argumnet allready on stack
    CALL newline

     ret     ;go back to OS
test2      endp
 ;---------------------------------------
```

```
putc  proc   near
 push bp                ;save bp
 mov  bp,sp             ;set up stak frame
 sub  sp, 0         ;allocate frame

 ; resolve rhs refernce
  mov ax,-8[BP]
    push ax
   POP ax  ; get char from the stack
  INT doscall ;  output it
  mov sp,bp    ;deallocate local variables
  pop bp       ;restore old value of bp
  RET  2
putc       endp
 ;-----------------------------------------

newline  proc   near
 push bp                ;save bp
 mov  bp,sp             ;set up stak frame
 sub  sp, 6         ;allocate frame

 ; refernce variable in outer block
   mov ax,[BP+4]
   mov ax,[BP+4]
 ; get the value of outer block variable
    mov dx,ax  ; save ax
    mov ax, 0[DX]
    push ax
 ; generate call argumnet allready on stack
    CALL putc

 ; refernce variable in outer block
   mov ax,[BP+4]
   mov ax,[BP+4]
 ; get the value of outer block variable
    mov dx,ax  ; save ax
    mov ax, 0[DX]
    push ax
 ; generate call argumnet allready on stack
    CALL putc

  mov sp,bp    ;deallocate local variables
  pop bp       ;restore old value of bp
  RET  0
newline       endp
 ;-----------------------------------------

prstring  proc   near
 push bp                ;save bp
 mov  bp,sp             ;set up stak frame
 sub  sp, 6         ;allocate frame

 ; push the value of variable on stack
```

```
   mov   ax,1
   push  ax
 ; resolve lhs refernce
   mov ax,BP- 2
 ; perform assignment
    POP bx
    MOV ax,bx
 ; push the value of variable on stack
   mov   ax,0
   push  ax
 ; refernce variable in outer block
   mov ax,[BP+4]
   mov ax,[BP+4]
 ; get the value of outer block variable
    mov dx,ax  ; save ax
    mov ax,-6[DX]
    push ax
 ; resolve lhs refernce
   mov ax,BP- 4
 ; perform assignment
    POP bx
    MOV ax,bx
 ; code for while stmt
lab7:

 ;Test for While Loop
 ; resolve rhs refernce
  mov ax,-4[BP]
    push ax
 ; resolve rhs refernce
  mov ax,-2[BP]
    push ax
 ;---- resolve le
; leave ax=1 on true, ax=0 on false
   POP  ax
   POP  bx
   CMP  ax,bx
   JG lab9   ; jump on greater than

   mov  ax,1  ; test passed
   JMP lab10
lab9:

    mov ax,0  ;test failed
lab10:

   mov dx,1
   cmp ax,dx
   JL  lab8
 ; Body of While Loop
 ; resolve rhs refernce
  mov ax,-2[BP]
    push ax
 ; refernce variable in outer block
```

```
   mov ax,[BP+4]
   mov ax,[BP+4]
 ; get the value of outer block variable
    mov dx,ax  ; save ax
    mov ax,-6[DX]
    push ax
 ; resolve lhs refernce
   mov ax,BP- 6
 ; perform assignment
    POP bx
    MOV ax,bx
 ; resolve rhs refernce
  mov ax,-6[BP]
    push ax
 ; generate call argumnet allready on stack
    CALL putc

 ; push the value of variable on stack
   mov   ax,1
   push  ax
 ; resolve rhs refernce
  mov ax,-2[BP]
    push ax
 ; perform addition
   POP ax
   POP bx
   ADD ax,bx
   push ax
 ; resolve lhs refernce
   mov ax,BP- 2
 ; perform assignment
    POP bx
    MOV ax,bx
   JMP lab7
lab8:

  mov sp,bp    ;deallocate local variables
  pop bp       ;restore old value of bp
  RET  0
prstring       endp
 ;-------------------------------------------

prnum  proc   near
 push bp                ;save bp
 mov  bp,sp             ;set up stak frame
 sub  sp, 6         ;allocate frame

 ; push the value of variable on stack
   mov   ax,10
   push  ax
 ; resolve lhs refernce
   mov ax,BP- 2
 ; perform assignment
    POP bx
```

```
    MOV ax,bx
 ; code for while stmt
lab11:

 ;Test for While Loop
 ; push the value of variable on stack
   mov   ax,3
   push  ax
 ; resolve rhs refernce
  mov ax,-2[BP]
    push ax
   mov dx,1
   cmp ax,dx
   JL  lab12
 ; Body of While Loop
 ; resolve lhs refernce
   mov ax,BP- 4
 ; perform assignment
    POP bx
    MOV ax,bx
 ; push the value of variable on stack
   mov   ax, 30H
   push  ax
 ; perform addition
   POP ax
   POP bx
   ADD ax,bx
   push ax
 ; resolve lhs refernce
   mov ax,BP- 6
 ; perform assignment
    POP bx
    MOV ax,bx
 ; resolve rhs refernce
  mov ax,-4[BP]
    push ax
 ; resolve lhs reference
     mov ax,BP+ 2
 ; perform assignment
    POP bx
    MOV ax,bx
 ; resolve rhs refernce
  mov ax,-6[BP]
    push ax
 ; resolve lhs refernce
   mov ax,BP- 8
 ; perform assignment
    POP bx
    MOV ax,bx
 ; resolve rhs refernce
  mov ax,-8[BP]
    push ax
 ; resolve lhs refernce
   mov ax,BP- 2
```

```
 ; refernce variable in outer block
   mov ax,[BP+4]
   mov ax,[BP+4]
 ; get the address of outer block variable
     mov ax,ax- 6
 ; perform assignment
    POP bx
    MOV ax,bx
 ; resolve lhs refernce
   mov ax,BP- 2
 ; perform assignment
    POP bx
    MOV ax,bx
   JMP lab11
lab12:

 ; push the value of variable on stack
   mov   ax,10
   push  ax
 ; resolve lhs refernce
   mov ax,BP- 6
 ; perform assignment
    POP bx
    MOV ax,bx
 ; resolve rhs refernce
  mov ax,-6[BP]
    push ax
 ; push the value of variable on stack
   mov   ax,0
   push  ax
 ; refernce variable in outer block
   mov ax,[BP+4]
   mov ax,[BP+4]
 ; get the address of outer block variable
     mov ax,ax- 6
 ; perform assignment
    POP bx
    MOV ax,bx
 ; push char on stack
   mov ax,32
   PUSH ax
 ; push the value of variable on stack
   mov   ax,1
   push  ax
 ; refernce variable in outer block
   mov ax,[BP+4]
   mov ax,[BP+4]
 ; get the address of outer block variable
     mov ax,ax- 6
 ; perform assignment
    POP bx
    MOV ax,bx
 ; push char on stack
   mov ax,32
```

```
   PUSH ax
 ; push the value of variable on stack
   mov   ax,2
   push  ax
 ; refernce variable in outer block
   mov ax,[BP+4]
   mov ax,[BP+4]
 ; get the address of outer block variable
     mov ax,ax- 6
 ; perform assignment
    POP bx
    MOV ax,bx
 ; push char on stack
   mov ax,32
   PUSH ax
 ; push the value of variable on stack
   mov   ax,3
   push  ax
 ; refernce variable in outer block
   mov ax,[BP+4]
   mov ax,[BP+4]
 ; get the address of outer block variable
     mov ax,ax- 6
 ; perform assignment
    POP bx
    MOV ax,bx
 ; push the value of variable on stack
   mov   ax,4
   push  ax
 ; resolve lhs refernce
   mov ax,BP- 2
 ; perform assignment
    POP bx
    MOV ax,bx
 ; code for while stmt
lab13:

 ;Test for While Loop
   mov dx,1
   cmp ax,dx
   JL  lab14
 ; Body of While Loop
 ; push char on stack
   mov ax,32
   PUSH ax
 ; resolve lhs refernce
   mov ax,BP- 2
 ; refernce variable in outer block
   mov ax,[BP+4]
   mov ax,[BP+4]
 ; get the address of outer block variable
     mov ax,ax- 6
 ; perform assignment
    POP bx
```

```
    MOV ax,bx
 ; push the value of variable on stack
   mov   ax,1
   push  ax
 ; resolve rhs refernce
  mov ax,-2[BP]
    push ax
 ; perform addition
   POP ax
   POP bx
   ADD ax,bx
   push ax
 ; resolve lhs refernce
   mov ax,BP- 2
 ; perform assignment
    POP bx
    MOV ax,bx
   JMP lab13
lab14:

 ; generate call argumnet allready on stack
    CALL prstring

  mov sp,bp    ;deallocate local variables
  pop bp       ;restore old value of bp
  RET  2
prnum       endp
 ;-------------------------------------------

code ENDS
     end start

(**********************)

              S Y M B O L      T A B L E
symbol          Level  Offset  size(equ)  data    Type  literal    Blk_num
            d    2        6         2   BYTE     VAR  -notused-       5

            z    2        2         2   INTEGER  PARM -notused-       5

            c    2        8         2   CHAR     VAR -notused-       5

            c    2        2         2   CHAR     PARM -notused-       2

            n    2        4         2   BYTE     VAR -notused-       4

            y    1        6         2   INTEGER  VAR -notused-       1

           lf    1        0         0     n/a    CONST       $0a       1

         putc    2        0         0     n/a    ENTRY                 2

            x    2        4         2   INTEGER   VAR -notused-       5
```

```
           x    1        4         2   INTEGER   VAR -notused-       1
     prstring    2        0         0     n/a    ENTRY                4
        test2    1        0         0     n/a    ENTRY                1
      newline    2        0         0     n/a    ENTRY                3
            i    2        2         2   INTEGER  VAR -notused-       5
            i    2        2         2   BYTE     VAR -notused-       4
            i    1        2         2   INTEGER   VAR -notused-       1
        prnum    2        0         0     n/a    ENTRY                5
            s    1        6         0     n/a    VAR -notused-       1
           ch    2        6         2   CHAR     VAR -notused-       4
           cr    1        0         0     n/a     CONST       $0d       1
            p    1        6         0     n/a     VAR -notused-       1


              B  S  T       T A B L E
 Index  Entry_name   lex_level  Outer  localsize  parmsize
    1     test2           1       0         6         0
    2      putc           2       1         0         2
    3   newline           2       1         0         0
    4  prstring           2       1         6         0
    5     prnum           2       1         8         2
** end of bst tables **



(**  T E S T    3    ouput   *********************************)

 ; OUTPUT of pascal compiler by Naser Abbasi
 ; CSE565 Oakland University  April 1988
 lf EQU  10
 cr EQU  13
 doscall  EQU   21h  ; dos interupt routine

st_seq   segment  byte stack ;define stack segment
         db   20 dup (?)
st_seq   ends
;-----------------------------------------
code     segment  byte public ; define code seqment

test3     proc   far
         assume cs:code
Start:
```

```
      push ds           ;save old value
      sub ax,ax         ;put zero in ax
      push ax            ;save it on stack


 ; generate call argumnet allready on stack
    CALL newline

 ; push the value of variable on stack
   mov   ax,0
   push  ax
 ; resolve lhs refernce
   mov ax,BP- 4
 ; perform assignment
    POP bx
    MOV ax,bx
 ; code for while stmt
lab1:

 ;Test for While Loop
 ; push the value of variable on stack
   mov   ax,7
   push  ax
 ; resolve rhs refernce
  mov ax,-4[BP]
    push ax
 ;---- resolve le
; leave ax=1 on true, ax=0 on false
   POP  ax
   POP  bx
   CMP  ax,bx
   JG lab3   ; jump on greater than

   mov  ax,1  ; test passed
   JMP lab4
lab3:

    mov ax,0  ;test failed
lab4:

   mov dx,1
   cmp ax,dx
   JL  lab2
 ; Body of While Loop
 ; resolve rhs refernce
  mov ax,-4[BP]
    push ax
 ; generate call argumnet allready on stack
    CALL prnum

 ; resolve rhs refernce
  mov ax,-4[BP]
    push ax
 ; generate call argumnet allready on stack
```

```
    CALL factorial

 ; resolve lhs refernce
   mov ax,BP- 6
 ; perform assignment
    POP bx
    MOV ax,bx
 ; resolve rhs refernce
  mov ax,-6[BP]
    push ax
 ; generate call argumnet allready on stack
    CALL prnum

 ; generate call argumnet allready on stack
    CALL newline

 ; push the value of variable on stack
   mov   ax,1
   push  ax
 ; resolve rhs refernce
  mov ax,-4[BP]
    push ax
 ; perform addition
   POP ax
   POP bx
   ADD ax,bx
   push ax
 ; resolve lhs refernce
   mov ax,BP- 4
 ; perform assignment
    POP bx
    MOV ax,bx
   JMP lab1
lab2:

 ; generate call argumnet allready on stack
    CALL newline

     ret     ;go back to OS
test3      endp
 ;-------------------------------------

putc  proc   near
 push bp                ;save bp
 mov  bp,sp             ;set up stak frame
 sub  sp,22         ;allocate frame

 ; resolve rhs refernce
  mov ax,-8[BP]
    push ax
   POP ax  ; get char from the stack
  INT doscall ;  output it
  mov sp,bp    ;deallocate local variables
  pop bp       ;restore old value of bp
```

```
  RET  2
putc       endp
 ;-------------------------------------------

newline  proc   near
 push bp                 ;save bp
 mov  bp,sp              ;set up stak frame
 sub  sp, 2         ;allocate frame

 ; refernce variable in outer block
   mov ax,[BP+4]
   mov ax,[BP+4]
 ; get the value of outer block variable
    mov dx,ax  ; save ax
    mov ax, 0[DX]
    push ax
 ; generate call argumnet allready on stack
    CALL putc

 ; refernce variable in outer block
   mov ax,[BP+4]
   mov ax,[BP+4]
 ; get the value of outer block variable
    mov dx,ax  ; save ax
    mov ax, 0[DX]
    push ax
 ; generate call argumnet allready on stack
    CALL putc

  mov sp,bp    ;deallocate local variables
  pop bp       ;restore old value of bp
  RET  0
newline       endp
 ;-------------------------------------------

prstring  proc   near
 push bp                 ;save bp
 mov  bp,sp              ;set up stak frame
 sub  sp, 2         ;allocate frame

 ; push the value of variable on stack
   mov   ax,1
   push  ax
 ; resolve lhs refernce
   mov ax,BP- 2
 ; perform assignment
    POP bx
    MOV ax,bx
 ; push the value of variable on stack
   mov   ax,0
   push  ax
 ; refernce variable in outer block
   mov ax,[BP+4]
   mov ax,[BP+4]
```

```
 ; get the value of outer block variable
    mov dx,ax  ; save ax
    mov ax,-22[DX]
    push ax
 ; resolve lhs refernce
   mov ax,BP- 4
 ; perform assignment
    POP bx
    MOV ax,bx
 ; code for while stmt
lab5:

 ;Test for While Loop
 ; resolve rhs refernce
  mov ax,-4[BP]
    push ax
 ; resolve rhs refernce
  mov ax,-2[BP]
    push ax
 ;---- resolve le
; leave ax=1 on true, ax=0 on false
   POP  ax
   POP  bx
   CMP  ax,bx
   JG lab7   ; jump on greater than

   mov  ax,1  ; test passed
   JMP lab8
lab7:

    mov ax,0  ;test failed
lab8:

   mov dx,1
   cmp ax,dx
   JL  lab6
 ; Body of While Loop
 ; resolve rhs refernce
  mov ax,-2[BP]
    push ax
 ; refernce variable in outer block
   mov ax,[BP+4]
   mov ax,[BP+4]
 ; get the value of outer block variable
    mov dx,ax  ; save ax
    mov ax,-22[DX]
    push ax
 ; resolve lhs refernce
   mov ax,BP- 6
 ; perform assignment
    POP bx
    MOV ax,bx
 ; resolve rhs refernce
  mov ax,-6[BP]
```

```
    push ax
 ; generate call argumnet allready on stack
    CALL putc

 ; push the value of variable on stack
   mov   ax,1
   push  ax
 ; resolve rhs refernce
  mov ax,-2[BP]
    push ax
 ; perform addition
   POP ax
   POP bx
   ADD ax,bx
   push ax
 ; resolve lhs refernce
   mov ax,BP- 2
 ; perform assignment
    POP bx
    MOV ax,bx
   JMP lab5
lab6:

  mov sp,bp    ;deallocate local variables
  pop bp       ;restore old value of bp
  RET  0
prstring       endp
 ;-----------------------------------------

prnum  proc   near
 push bp                ;save bp
 mov  bp,sp             ;set up stak frame
 sub  sp, 2         ;allocate frame

 ; push the value of variable on stack
   mov   ax,10
   push  ax
 ; resolve lhs refernce
   mov ax,BP- 2
 ; perform assignment
    POP bx
    MOV ax,bx
 ; code for while stmt
lab9:

 ;Test for While Loop
 ; push the value of variable on stack
   mov   ax,3
   push  ax
 ; resolve rhs refernce
  mov ax,-2[BP]
    push ax
   mov dx,1
   cmp ax,dx
```

```
  JL  lab10
 ; Body of While Loop
 ; resolve lhs refernce
  mov ax,BP- 4
 ; perform assignment
   POP bx
   MOV ax,bx
 ; push the value of variable on stack
  mov   ax,48
  push  ax
 ; push the value of variable on stack
  mov   ax,10
  push  ax
 ; resolve rhs refernce
 mov ax,-4[BP]
   push ax
 ; perform multiplication
  POP ax
  POP bx
  MUL bx
  push ax
; resolve rhs refernce
    mov ax, 2[BP]
    push ax
 POP ax
 POP dx
 SUB ax,bx
 PUSH ax
 ; perform addition
  POP ax
  POP bx
  ADD ax,bx
  push ax
 ; resolve lhs refernce
  mov ax,BP- 6
 ; perform assignment
   POP bx
   MOV ax,bx
 ; resolve rhs refernce
 mov ax,-4[BP]
   push ax
 ; resolve lhs reference
    mov ax,BP+ 2
 ; perform assignment
   POP bx
   MOV ax,bx
 ; resolve rhs refernce
 mov ax,-6[BP]
   push ax
 ; resolve lhs refernce
  mov ax,BP- 8
 ; perform assignment
   POP bx
   MOV ax,bx
```

```
 ; resolve rhs refernce
  mov ax,-8[BP]
    push ax
 ; resolve lhs refernce
   mov ax,BP- 2
 ; refernce variable in outer block
   mov ax,[BP+4]
   mov ax,[BP+4]
 ; get the address of outer block variable
     mov ax,ax-22
 ; perform assignment
    POP bx
    MOV ax,bx
 ; push the value of variable on stack
   mov   ax,1
   push  ax
 ; resolve rhs refernce
  mov ax,-2[BP]
    push ax
  POP ax
  POP dx
  SUB ax,bx
  PUSH ax
 ; resolve lhs refernce
   mov ax,BP- 2
 ; perform assignment
    POP bx
    MOV ax,bx
   JMP lab9
lab10:

 ; push the value of variable on stack
   mov   ax,10
   push  ax
 ; resolve lhs refernce
   mov ax,BP- 6
 ; perform assignment
    POP bx
    MOV ax,bx
 ; resolve rhs refernce
  mov ax,-6[BP]
    push ax
 ; push the value of variable on stack
   mov   ax,0
   push  ax
 ; refernce variable in outer block
   mov ax,[BP+4]
   mov ax,[BP+4]
 ; get the address of outer block variable
     mov ax,ax-22
 ; perform assignment
    POP bx
    MOV ax,bx
 ; push char on stack
```

```
  mov ax,32
  PUSH ax
 ; push the value of variable on stack
  mov   ax,1
  push  ax
 ; refernce variable in outer block
  mov ax,[BP+4]
  mov ax,[BP+4]
 ; get the address of outer block variable
    mov ax,ax-22
 ; perform assignment
   POP bx
   MOV ax,bx
 ; push char on stack
  mov ax,32
  PUSH ax
 ; push the value of variable on stack
  mov   ax,2
  push  ax
 ; refernce variable in outer block
  mov ax,[BP+4]
  mov ax,[BP+4]
 ; get the address of outer block variable
    mov ax,ax-22
 ; perform assignment
   POP bx
   MOV ax,bx
 ; push char on stack
  mov ax,32
  PUSH ax
 ; push the value of variable on stack
  mov   ax,3
  push  ax
 ; refernce variable in outer block
  mov ax,[BP+4]
  mov ax,[BP+4]
 ; get the address of outer block variable
    mov ax,ax-22
 ; perform assignment
   POP bx
   MOV ax,bx
 ; push the value of variable on stack
  mov   ax,4
  push  ax
 ; resolve lhs refernce
  mov ax,BP- 2
 ; perform assignment
   POP bx
   MOV ax,bx
 ; code for while stmt
lab11:

 ;Test for While Loop
 ; resolve rhs refernce
```

```
 mov ax,-2[BP]
   push ax
; push the value of variable on stack
  mov   ax,10
  push  ax
; resolve rhs refernce
 mov ax,-2[BP]
   push ax
; refernce variable in outer block
  mov ax,[BP+4]
  mov ax,[BP+4]
; get the value of outer block variable
   mov dx,ax  ; save ax
   mov ax,-22[DX]
   push ax
; push char on stack
  mov ax,48
  PUSH ax
  mov dx,1
  cmp ax,dx
  JL  lab12
; Body of While Loop
; push char on stack
  mov ax,32
  PUSH ax
; resolve lhs refernce
  mov ax,BP- 2
; refernce variable in outer block
  mov ax,[BP+4]
  mov ax,[BP+4]
; get the address of outer block variable
    mov ax,ax-22
; perform assignment
   POP bx
   MOV ax,bx
; push the value of variable on stack
  mov   ax,1
  push  ax
; resolve rhs refernce
 mov ax,-2[BP]
   push ax
; perform addition
  POP ax
  POP bx
  ADD ax,bx
  push ax
; resolve lhs refernce
  mov ax,BP- 2
; perform assignment
   POP bx
   MOV ax,bx
  JMP lab11
lab12:
```

```
 ; generate call argumnet allready on stack
    CALL prstring

  mov sp,bp    ;deallocate local variables
  pop bp       ;restore old value of bp
  RET  2
prnum       endp
 ;-----------------------------------------

     ret     ;go back to OS
test3      endp

 ;-----------------------------------
code ENDS
     end start
```

S Y M B O L  T A B L E

| symbol | Level | Offset | size(equ) | data | Type | literal | Blk_num |
|---|---|---|---|---|---|---|---|
| factorial | 1 | 0 | 0 | n/a | ENTRY | | 6 |
| d | 2 | 6 | 2 | BYTE | VAR | -notused- | 5 |
| z | 1 | 2 | 2 | INTEGER | PARM | -notused- | 6 |
| z | 2 | 2 | 2 | INTEGER | PARM | -notused- | 5 |
| c | 2 | 8 | 2 | CHAR | VAR | -notused- | 5 |
| c | 2 | 2 | 2 | CHAR | PARM | -notused- | 2 |
| n | 2 | 4 | 2 | BYTE | VAR | -notused- | 4 |
| y | 1 | 6 | 2 | INTEGER | VAR | -notused- | 1 |
| lf | 1 | 0 | 0 | n/a | CONST | 10 | 1 |
| putc | 2 | 0 | 0 | n/a | ENTRY | | 2 |
| x | 2 | 4 | 2 | INTEGER | VAR | -notused- | 5 |
| x | 1 | 4 | 2 | INTEGER | VAR | -notused- | 1 |
| test3 | 1 | 0 | 0 | n/a | ENTRY | | 1 |
| prstring | 2 | 0 | 0 | n/a | ENTRY | | 4 |
| newline | 2 | 0 | 0 | n/a | ENTRY | | 3 |
| i | 2 | 2 | 2 | INTEGER | VAR | -notused- | 5 |

```
            i    2        2         2   BYTE      VAR -notused-       4

            i    1        2         2   INTEGER   VAR -notused-       1

         prnum    2        0         0     n/a    ENTRY                5

            s    1       22        16     n/a     VAR -notused-       1

           ch    2        6         2   CHAR      VAR -notused-       4

           cr    1        0         0     n/a     CONST        13      1


                B  S  T       T A B L E
 Index  Entry_name   lex_level  Outer  localsize  parmsize
    1     test3           1       0         2         0
    2      putc           2       1         0         2
    3   newline           2       1         0         0
    4  prstring           2       1         6         0
    5     prnum           2       1         8         2
    6 factorial           1       0         0         2

** end of bst tables **
```