(\* 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 \*\*