SP Assignment – 1

Pavan S. Patil
Roll no. 51

Page No.
Date

Name :- Pavan S. Patil
Roll no :- 51

ASSIGNMENT-1

* Problem statement :-
write a program to implement Pass-I of Pass-II assembler for symbols and literal processing consider following cases
i) forward reference
ii) DS and DC statement
iii) START, EQU, LTORG, END
iv) Error Handling symbol used but not defined

* objective :-
1. To study basic translation process of assembly language to machine language
2. To study two pass Assembly process.

* Theory -

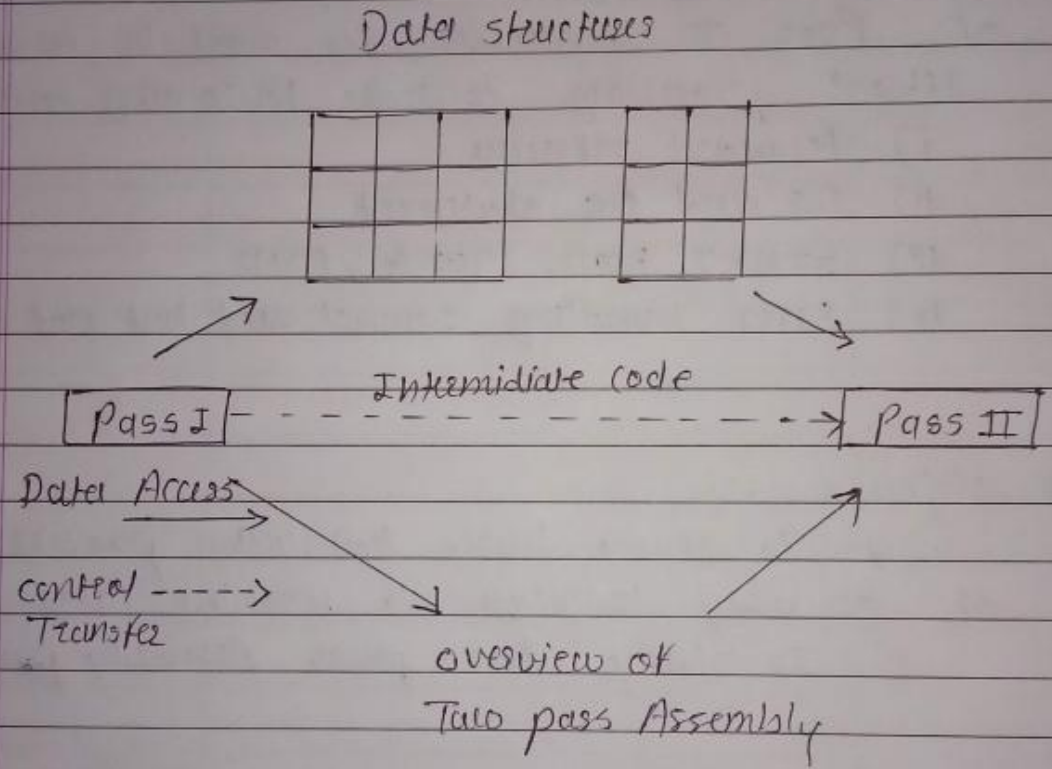
A language Translator bridges an execution gap to machine language of computer sys. An assembler is a language translator whose source language is Assembly language.

Language processing activity consist of two part/phases. Analysis phase and synthesis phase

Sheikrupp

Date

Analysis of source program consist of three components, lexical rules, syntax rules and symantic rules lexical rules govern the formation of valid statements in source language



overview of Two pass Assembly

Analysis of source program statements may not be immediately followed by synthesis of equivalent target statements This is due to forward reference issues concerning memory requirements and organization of language processor (LP.)

Page No.
Date / /

Language Processor Pass :-

it is the processing of every statement in source program or its equivalent representation to perform language processing function.

Assemby language statements :-

(1) Imperative statement - Indicates an action to be performed during the execution of Assembled program each imperative statement usually translates into one machine instn.

(2) Declarative statement - e.g. DS saves area of memory and associates names with them.

(3) DC constructs memory word containing constants Assembler directives instructs the Assembler to perform certain actions during assembly of a program.

Function of Analysis And Synthesis

Analysis Phase -

(1) Isolate the label operation code and operand fields of statements.

Date

(2) Enter the symbol found in label field (if any) and address of next available machine word into symbol table.

(3) Validate the mnemonic operation code by looking it up in the mnemonic table.

(4) Determine the machine storage requirements of the statement by considering the mnemonic operation code and operand fields of statement.

(5) Calculate address of the first machine word following the target code generated for this statement

Synthesis phase.

1. Obtain the machine operation code corresponding to the mnemonic operation code by searching the mnemonic table.
2. Obtain the address of the operand from the symbol table.
3. Synthesize the machine instruction or the machine form of the constant as the case may be.

Page No.
Date

Design of a Two Pass Assemblee: -

Tasks peefoemed by the passes of two-pass assemblee aee as follows:

Pass I: -

Sepaeate the symbol, mnemonic opcode and opeeand fields.

Deteemine the stoeage-eequieed foe eveey assembly language statement and update the location countee.

Build the symbol table and the liteeal table.

Consteuct the inteemediate code foe eveey assembly language statement.

Pass II: -

Synthesize the taeget code by peocessing the inteemediate code geneeated dueing

Data steuctuees eequieed foe pass I:

1. Souece file containing assembly peogeam
2. MOT: A table of mnemonic op-codes and eelated infoemation. It has the following fields

Mnemonic : Such as ADD, END, DC

TYPE : IS foe impeeative, DL foe declaeative and AD foe Assemblee dieective OP- code : Opeeation code indicating the opeeation to be peefoemed.

Length : Length of insteuction eequieed foe Location Countee Peocessing

Page No
Date

Index	Mnemonic	Type	O-code	len.
0	ADD	IS	01	01
1	BC	IS	07	01
2	COMP	IS	06	01
3	DIV	IS	08	01
4	EQU	AD	03	-
5	DC	DL	01	-
6	DS	DL	02	-
7	END	AD	05	-

3. SYMTB : The Symbol Table

Fields are symbol name, Address (LC val) Initialize all values in the address fields to 2 and then symbol gets added when its appears in label field replace address value with current LC.

Symbol	Address
loop	204
Next	214

4. LITTAB and POOLTAB : Literal table stores the literals used in the program and POOLTABLE stores the pointers to the literals in current literal pool.

Page No.

Date / /

Data Steuctuee used by Pass II:

1. OPTAB : A table of mnemonic opcodes and eelated infoemation.
2. SYMTAB : The symbol table
3. LITTAB : A table of liteeals used in the peogeam
4. Inteemediate code geneeated by Pass I
5. Output file containing Taeget code / eeeoe listing

Conclusion : Thus we have designed and implemented pass 1of a 2 pass assemblee in C.

FAQs :

Q. 1) what is meant by pass of an Assembler

→ it generates instructions by evaluating the mnemonics (symbol) in operation field and find the value of symbol and literal to produce machine code Assembler reads the Assembly language source code twice before it outputs object code. each read of source code is called pass.

Q. 2) Explain need of two pass Assembler.

→ most assemblers use a 2-pass system is to address the problem of forwarding references - references to variable or subroutine that have not yet been encountered when passing the source code.

Q. 3) Explain terms such as forward reference and backward reference.

→ forward reference are not resolved, That is the use of symbol are at one point in the next which is not defined until some later point gives the wrong result. Backward references, on the other hand handled correctly.

Shrikrupa

ASSIGNMENT 1 CODE :

```
//Implimentation of Pass-1 of assembler

#include<stdio.h>
#include<string.h>
#include<stdlib.h>

static int newadd=0; //default addrees start from '0'
static int symsr=1;
static int stentry=0;
static int ltrsr=1;
static int ptsr=1;
int address;

typedef struct srccode
{
      char label[30];
      char mnemonic[30];
      char operand1[30];
      char operand2[30];
}srccode; // for reading instr from src code


typedef struct motcontent
{
      int sr;
      char name[30];
      int opcode;
      char class[30];
}motcontent; // for reading instr from mot file


typedef struct opcodeclass
{
      char a1[10];
      char a2[10];
}opcodeclass;
```

```
typedef struct operand1reg
{
      char a1[10];
      char a2[10];
}operand1reg;


typedef struct operand2sym
{
      char a1[10];
      char a2[10];
}operand2sym;



void writedatatost(char label[],int add)
{
      FILE *fpsym;

      fpsym=fopen("symtable.txt","a");
      fprintf(fpsym,"%d\t%s\t%d\n",symsr,label,add);
    symsr++;
      fclose(fpsym);
      stentry=1;
}//writing a data to symbol table


int issymbolalreadyexist(char label[])
{
      FILE *fpsym;
      int sr,symadd;
      char symname[30];

            fpsym=fopen("symtable.txt","r");
            fscanf(fpsym,"%d%s%d",&sr,symname,&symadd);
            while(!feof(fpsym))
            {
                  if(strcmp(symname,label)==0)
```

```
                return 1;  //already present
                fscanf(fpsym,"%d%s%d",&sr,symname,&symadd);
           }//while
           fclose(fpsym);
return 0;  //not present
}//if the symbol already exist in symbol table



void updateaddressinst(char label[],int add)
{
     FILE *fpsym,*fptemp;
     int sr,symadd;
     char symname[30];

           fpsym=fopen("symtable.txt","r");
           fptemp=fopen("temp.txt","w");
           fscanf(fpsym,"%d%s%d",&sr,symname,&symadd);
           while(!feof(fpsym))
           {
                if(strcmp(symname,label)==0)

     fprintf(fptemp,"%d\t%s\t%d\n",sr,label,add);
                else

     fprintf(fptemp,"%d\t%s\t%d\n",sr,symname,symadd);
                fscanf(fpsym,"%d%s%d",&sr,symname,&symadd);
           }//while
           fclose(fpsym);
           fclose(fptemp);

           remove("symtable.txt");
           rename("temp.txt","symtable.txt");
           remove("temp.txt");
}//updating the address in the symbol table


int loc_of_sym(char operand[])
{
     FILE *fpsym;
```

```
      int sr,symadd;
      char symname[30];

            fpsym=fopen("symtable.txt","r");
            fscanf(fpsym,"%d%s%d",&sr,symname,&symadd);
            while(!feof(fpsym))
            {
                  if(strcmp(symname,operand)==0)
                        return sr;
                  fscanf(fpsym,"%d%s%d",&sr,symname,&symadd);
            }//while
      return -1;
}//return the sr no of the symbol from the symbol table


int add_of_sym(char operand[])
{
      FILE *fpsym;
      int sr,symadd;
      char symname[30];

            fpsym=fopen("symtable.txt","r");
            fscanf(fpsym,"%d%s%d",&sr,symname,&symadd);
            while(!feof(fpsym))
            {
                  if(strcmp(symname,operand)==0)
                        return symadd;
                  fscanf(fpsym,"%d%s%d",&sr,symname,&symadd);
            }//while
      return -1;
}//return the sr no of the symbol from the symbol table



void writedatatolt(char label[],int add)
{
      FILE *fpltr;

      fpltr=fopen("ltrtable.txt","a");
      fprintf(fpltr,"%d\t%s\t%d\n",ltrsr,label,add);
```

```
    ltrsr++;
     fclose(fpltr);
}//writing a data to symbol table


int findtotalPT()
{

     int cnt=0;
     FILE *fpltr;
     int sr,ltradd;
     char ltrname[30];
          fpltr=fopen("ltrtable.txt","r");
          fscanf(fpltr,"%d%s%d",&sr,ltrname,&ltradd);
          while(!feof(fpltr))
          {
               cnt++;
               fscanf(fpltr,"%d%s%d",&sr,ltrname,&ltradd);
          }//while
          fclose(fpltr);

return cnt;


}


int isliteralalreadyexist(char label[30])
{
     FILE *fpltr;
     int sr,ltradd;
     char ltrname[30];
          fpltr=fopen("ltrtable.txt","r");
          fscanf(fpltr,"%d%s%d",&sr,ltrname,&ltradd);
          //printf("\nhii\n");
          while(!feof(fpltr))
          {
               //printf("%d %s %d",&sr,ltrname,&ltradd);

     //printf("Comaprison:%d",strcmp(ltrname,label));
```

```
                if(strcmp(ltrname,label)==0 && ltradd==-1)
                return 1;  //already present
                fscanf(fpltr,"%d%s%d",&sr,ltrname,&ltradd);
          }//while
          fclose(fpltr);

return 0;  //not present
}//if the symbol already exist in symbol table



int loc_of_ltr(char operand[])
{
     FILE *fpltr;
     int sr,ltradd;
     char ltrname[30];

          fpltr=fopen("ltrtable.txt","r");
          fscanf(fpltr,"%d%s%d",&sr,ltrname,&ltradd);
          while(!feof(fpltr))
          {
                if(strcmp(ltrname,operand)==0)
                     return sr;
                fscanf(fpltr,"%d%s%d",&sr,ltrname,&ltradd);
          }//while
     return -1;
}//return the sr no of the symbol from the symbol table



int writedataintoic(opcodeclass a,operand1reg b,operand2sym c)
{
     FILE *fpic;

          fpic=fopen("ictable.txt","a");
          if(strcmp(a.a1,"AD")==0)

     fprintf(fpic,"\t*\t(%s,%s)\t(%s,%s)\t(%s,%s)\n",a.a1,a.a2,
b.a1,b.a2,c.a1,c.a2);
          else
```

```
            {

      fprintf(fpic,"\t%d\t(%s,%s)\t(%s,%s)\t(%s,%s)\n",newadd,a.
a1,a.a2,b.a1,b.a2,c.a1,c.a2);
                  //printf("\nNew address Found %d",newadd);
                  newadd++;
            }
            fclose(fpic);
}// write data in IC



int returnsymboladdress(char symbol[])
{
      FILE *fpsym;
      int sr,symadd;
      char symname[30];
            fpsym=fopen("symtable.txt","r");
            fscanf(fpsym,"%d%s%d",&sr,symname,&symadd);

            while(!feof(fpsym))
            {
                  if(strcmp(symname,symbol)==0)
                  {
                  return symadd;
                  }
                  fscanf(fpsym,"%d%s%d",&sr,symname,&symadd);
            }
            fclose(fpsym);

return 1;
}//return symbol address




void updateicbyltrtable()
{
      FILE *fpltr,*fpic,*fppt,*fptemp;
      opcodeclass a;
```

```
      int sr,ltradd;
      char ltrname[30];

      fpltr=fopen("ltrtable.txt","r");
      fpic=fopen("ictable.txt","a");
      fppt=fopen("pooltable.txt","a");
      fptemp=fopen("temp.txt","w");

      fscanf(fpltr,"%d%s%d",&sr,ltrname,&ltradd);
      while(!feof(fpltr))
      {
            if(ltradd==-1)
            {
                  fprintf(fpic,"\t%d\t%s\t(-,-)\t(-,-
)\n",newadd,ltrname);

      fprintf(fptemp,"%d\t%s\t%d\n",sr,ltrname,newadd);
                  newadd++;
            }
            else
            {

      fprintf(fptemp,"%d\t%s\t%d\n",sr,ltrname,ltradd);
            }
           fscanf(fpltr,"%d%s%d",&sr,ltrname,&ltradd);
      }//while

      fclose(fpltr);
      fclose(fpic);
      fclose(fppt);
      fclose(fptemp);

      remove("ltrtable.txt");
      rename("temp.txt","ltrtable.txt");
      remove("temp.txt");
}//writing data to ic using literal table



void writeaddofltr()
```

```
{
      int sr,ltradd;
      char ltrname[30];
      FILE *fpltr,*fpic,*fptemp;
      fpltr=fopen("ltrtable.txt","r");
      fpic=fopen("ictable.txt","a");
      fptemp=fopen("temp.txt","w");

      fscanf(fpltr,"%d%s%d",&sr,ltrname,&ltradd);
      while(!feof(fpltr))
      {
            if(ltradd==-1)
            {
                  fprintf(fpic,"%d\t%s\t(-,-)\t(-,-
)\n",newadd,ltrname);

      fprintf(fptemp,"\t%d\t%s\t%d\n",sr,ltrname,newadd);
                  newadd++;
            }
            else
            {

      fprintf(fptemp,"\t%d\t%s\t%d\n",sr,ltrname,ltradd);
            }
            fscanf(fpltr,"%d%s%d",&sr,ltrname,&ltradd);
      }//while


fclose(fpltr);
fclose(fpic);
fclose(fptemp);
remove("ltrtable.txt");
rename("temp.txt","ltrtable.txt");
remove("temp.txt");
}//after end statement is



int issymbolisvalid()
{
```

```
      FILE *fpsym;
      int sr,symadd;
      char symname[30];
      int flag=1;

            fpsym=fopen("symtable.txt","r");
            fscanf(fpsym,"%d%s%d",&sr,symname,&symadd);

            while(!feof(fpsym))
            {
                  if(symadd==-1)
                  {
                        //printf("\nSymbol '%s' has no
address",symname);
                        flag=0;
                  }
                  fscanf(fpsym,"%d%s%d",&sr,symname,&symadd);
      }
            fclose(fpsym);
      if(flag==0)
      return 0;
return 1;
}//check validity of a symbol




int main()
{
      remove("symtable.txt");
      remove("ictable.txt");
      remove("pooltable.txt");
      remove("ltrtable.txt");

      char srcfile[]="code.txt"; //source file
      char motfile[]="mot.txt"; //mot file
      FILE *fpcode,*fpmot,*fpltr,*fppt;

      srccode sc;      //creating object of structure
```

```
      motcontent mot;  //creating object of structure

      opcodeclass a;
      operand1reg b;
      operand2sym c;

      int motflag=0,updateaddvalue=0,ltorgflag=0,startflag=0;
      char tempstr[30],*token;

      fpltr=fopen("ltrtable.txt","w");
      fclose(fpltr);

      fppt=fopen("pooltable.txt","w");
      fclose(fppt);

      fppt=fopen("pooltable.txt","a");
      fprintf(fppt,"%d\t#1\n",ptsr,findtotalPT());
      ptsr++;

      fpcode=fopen(srcfile,"r");
      fscanf(fpcode,"%s%s%s%s",sc.label,sc.mnemonic,sc.operand1,
sc.operand2);
    if(strcmp(sc.operand1,"*")!=0)
      {
            newadd=atoi(sc.operand1);
            address=newadd;
      }//if starting address is present
    while(!(feof(fpcode)))
    {


       //for label field
      if(strcmp(sc.label,"*")!=0)
      {
            if(stentry==0)
            {
                  writedatatost(sc.label,newadd);
            }//if no other symbol is present in symbol table
            else
            {
```

```
                if(issymbolalreadyexist(sc.label)==0)
                          writedatatost(sc.label,newadd);
                     else
                          updateaddressinst(sc.label,newadd);

                }//if one of the symbol is already present in
symbol table
     }//for label other than * symbol



     //for mnemonic field
     if(strcmp(sc.mnemonic,"*")!=0)
     {
          motflag==0;
          fpmot=fopen(motfile,"r");


     fscanf(fpmot,"%d%s%d%s",&mot.sr,mot.name,&mot.opcode,mot.c
lass);
                while(!(feof(fpmot)))
                {
                     //printf("%s\n",mot.name);
                     if(strcmp(sc.mnemonic,mot.name)==0)
                     {
                          //printf("hii...");
                          //printf("%s\n",mot.name);

                          strcpy(a.a1,mot.class);
                          sprintf(a.a2,"%d",mot.opcode);
                          motflag=1;
                          break;
                          //printf("%s\t%s\n",a.a1,a.a2);
                     }//mnemonic found in mot


     fscanf(fpmot,"%d%s%d%s",&mot.sr,mot.name,&mot.opcode,mot.c
lass);
                }//while
```

```
                if(motflag==0)
                {
                      printf("\nInvalid Mnemonic
%s\n",sc.mnemonic);
                      exit(0);
                }//mnomonic not fond in mot
               fclose(fpmot);
      }//for mnemonic checking mot entries



      //for operand1 field
      if(strcmp(sc.operand1,"*")!=0)
      {
            if(strcmp(sc.operand1,"AREG")==0 ||
strcmp(sc.operand1,"A")==0)
            {
                  if(strcmp(sc.operand2,"*")!=0)
                  {
                        strcpy(b.a1,"R");
                        sprintf(b.a2,"%d",1);
                        //printf("%s\t%s\n",b.a1,b.a2);
                  }//means both the operands are present
                  else
                  {
                        if(stentry==0)
                        {
                              writedatatost(sc.operand1,-1);
                        }//if no other symbol is present in symbol
table
                        else
                        {

      if(issymbolalreadyexist(sc.label)==0)

      writedatatost(sc.operand1,-1);
                        }//if one of the symbol is already present
in symbol table

                              strcpy(b.a1,"S");
```

```
      sprintf(b.a2,"%d",loc_of_sym(sc.operand1));
                              //printf("%s\t%s\n",b.a1,b.a2);
                  }//means only one operand is present
            }//operand1 is A register


                  else if(strcmp(sc.operand1,"BREG")==0 ||
strcmp(sc.operand1,"B")==0)
            {
                  if(strcmp(sc.operand2,"*")!=0)
                  {
                        strcpy(b.a1,"R");
                        sprintf(b.a2,"%d",2);
                        //printf("%s\t%s\n",b.a1,b.a2);
                  }//means both the operands are present
                  else
                  {
                        if(stentry==0)
                        {
                              writedatatost(sc.operand1,-1);
                        }//if no other symbol is present in symbol
table
                        else
                        {

      if(issymbolalreadyexist(sc.label)==0)

      writedatatost(sc.operand1,-1);
                              }//if one of the symbol is already
present in symbol table

                              strcpy(b.a1,"S");

      sprintf(b.a2,"%d",loc_of_sym(sc.operand1));
                              //printf("%s\t%s\n",b.a1,b.a2);
                  }//means only one operand is present
            }//operand1 is B register
```

```
           else if(strcmp(sc.operand1,"CREG")==0 ||
strcmp(sc.operand1,"C")==0)
           {
                 if(strcmp(sc.operand2,"*")!=0)
                 {
                       strcpy(b.a1,"R");
                       sprintf(b.a2,"%d",3);
                       //printf("%s\t%s\n",b.a1,b.a2);
                 }//means both the operands are present
                 else
                 {
                       if(stentry==0)
                       {
                             writedatatost(sc.operand1,-1);
                       }//if no other symbol is present in symbol
table
                       else
                       {

      if(issymbolalreadyexist(sc.label)==0)

      writedatatost(sc.operand1,-1);
                             }//if one of the symbol is already
present in symbol table

                             strcpy(b.a1,"S");

      sprintf(b.a2,"%d",loc_of_sym(sc.operand1));
                             //printf("%s\t%s\n",b.a1,b.a2);
                 }//means only one operand is present
           }//operand1 is C register


                 else if(strcmp(a.a1,"DL")==0)
                 {
                       strcpy(b.a1,"C");
                       strcpy(b.a2,sc.operand1);
                       //printf("%s\t%s\n",b.a1,b.a2);
                 }//if it is a DS or DC statements
```

```
                else if(strcmp(sc.operand1,"EQ")==0 ||
strcmp(sc.operand1,"NEQ")==0 || strcmp(sc.operand1,"GT")==0 ||
strcmp(sc.operand1,"GE")==0 || strcmp(sc.operand1,"LT")==0 ||
strcmp(sc.operand1,"LE")==0)
                {
                      if(strcmp(sc.operand2,"*")!=0)
                      {
                            FILE *fpco;
                            int sr,code;
                            char operator[20];


      fpco=fopen("cond_operator.txt","r");

      fscanf(fpco,"%d%s%d",&sr,operator,&code);
                                    while(!(feof(fpco)))
                                    {

      if(strcmp(operator,sc.operand1)==0)
                                          {
                                                strcpy(b.a1,"C");

      sprintf(b.a2,"%d",code);

      //printf("%s\t%s\n",b.a1,b.a2);
                                          }//if it is a conditional
operator

      fscanf(fpco,"%d%s%d",&sr,operator,&code);
                                    }//read till the end of file
                      }//if 2nd operand is present
                      else
                      {
                            if(stentry==0)
                            {
                                  writedatatost(sc.operand1,-1);
                            }//if no other symbol is present in
symbol table
                            else
                            {
```

```
      if(issymbolalreadyexist(sc.label)==0)

      writedatatost(sc.operand1,-1);
                                    }//if one of the symbol is
already present in symbol table

                              strcpy(b.a1,"S");

      sprintf(b.a2,"%d",loc_of_sym(sc.operand1));
                              //printf("%s\t%s\n",b.a1,b.a2);
                        }//consider it as a symbol
                  }//if it is a conditional operator


                        else
                        {
                              strcpy(tempstr,sc.operand1);
                              token=strtok(tempstr,"+");
                        if(startflag!=0)
                        {
                              if(stentry==0)
                        {
                              writedatatost(token,-1);
                        }//if no other symbol is present in symbol
table
                        else
                        {

      if(issymbolalreadyexist(token)==0)

      writedatatost(token,-1);

                              }//if one of the symbol is already
present in symbol table

                              strcpy(b.a1,"S");

      sprintf(b.a2,"%d",loc_of_sym(token));
                              //printf("%s\t%s\n",b.a1,b.a2);
```

```
                        }//if
                        else
                        {
                              startflag=1;
                              strcpy(b.a1,"C");
                              sprintf(b.a2,"%d",newadd);
                        }
                  }//else

                  /*else
                  {
                        strcpy(b.a1,"-");
                        strcpy(b.a2,"-");
                  }//if there is no operand
                  */
      }//for operand1
      else
      {
            strcpy(b.a1,"-");
                  strcpy(b.a2,"-");
      }//if operand1 is '*'


            //for operand2 field
            if(strcmp(sc.operand2,"*")!=0)
      {
            //printf("hii\n");
            if(strcmp(sc.operand2,"AREG")==0 ||
strcmp(sc.operand2,"A")==0 || strcmp(sc.operand2,"BREG")==0 ||
strcmp(sc.operand2,"B")==0 || strcmp(sc.operand2,"CREG")==0 ||
strcmp(sc.operand2,"C")==0)
            {
                  //printf("hii\n");
                  if(stentry==0)
                  {
                        writedatatost(sc.operand2,-1);
                  }//if no other symbol is present in symbol
table
                  else
                        {
```

```
                        if(issymbolalreadyexist(sc.operand2)==0)
                                    writedatatost(sc.operand2,-1);
                        }//if one of the symbol is already present
in symbol table

                        strcpy(c.a1,"S");

      sprintf(c.a2,"%d",loc_of_sym(sc.operand2));
                        //printf("%s\t%s\n",c.a1,c.a2);
            }//operand2 is A,B,C register



            else if(sc.operand2[0]=='=')
            {
                  //printf("hii\n");
                        //printf("Returned
Value:%d",isliteralalreadyexist(sc.operand2));
                        if(isliteralalreadyexist(sc.operand2)==0)
                        {
                              //printf("hiih");
                              writedatatolt(sc.operand2,-1);
                              strcpy(c.a1,"l");
                              sprintf(c.a2,"%d",ltrsr-1);
                        }
                        else
                        {
                              strcpy(c.a1,"l");

      sprintf(c.a2,"%d",loc_of_ltr(sc.operand2)); //to convert
integer to string
                        }//if literal is already present in the
literal table
                        //printf("%s\t%s\n",c.a1,c.a2);


            }//if operand2 is a literal


            else
```

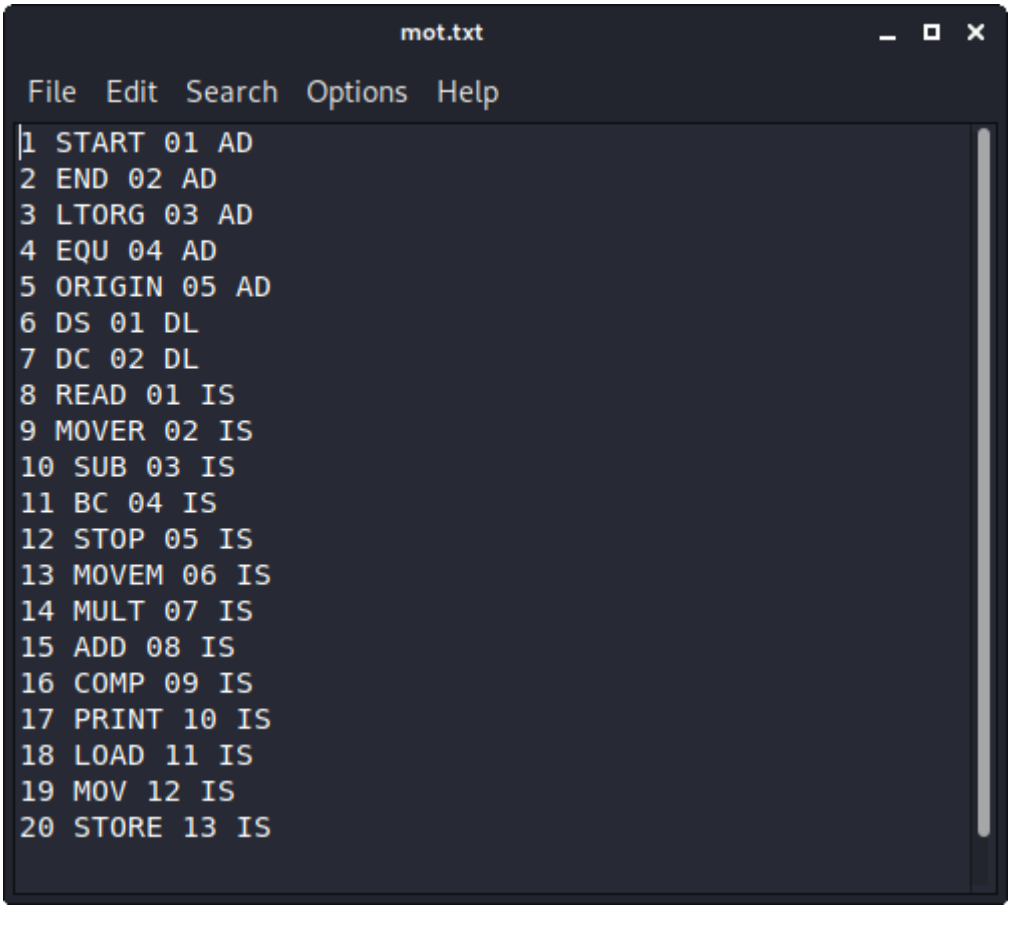
```
            {
                  //printf("hii\n");
                  if(stentry==0)
                  {
                        writedatatost(sc.operand2,-1);
                  }//if no other symbol is present in symbol
table
                  else
                        {
                        if(issymbolalreadyexist(sc.operand2)==0)
                                    writedatatost(sc.operand2,-1);
                              //else

      //updateaddressinst(sc.operand2,address);
                        }//if one of the symbol is already present
in symbol table

                        strcpy(c.a1,"S");

      sprintf(c.a2,"%d",loc_of_sym(sc.operand2));
                        //printf("%s\t%s\n",c.a1,c.a2);
            }//if operand2 is a symbol

      }//for operand2
      else
      {
            strcpy(c.a1,"-");
            strcpy(c.a2,"-");
                  //printf("%s\t%s\n",c.a1,c.a2);
      }//for operand 2



      // for DS i.e. it's value is given
            if(strcmp(a.a1,"DL")==0 && strcmp(a.a2,"1")==0)
            {
                  updateaddvalue=0;
                  updateaddvalue=atoi(sc.operand1)-1;
            }//for DS
```

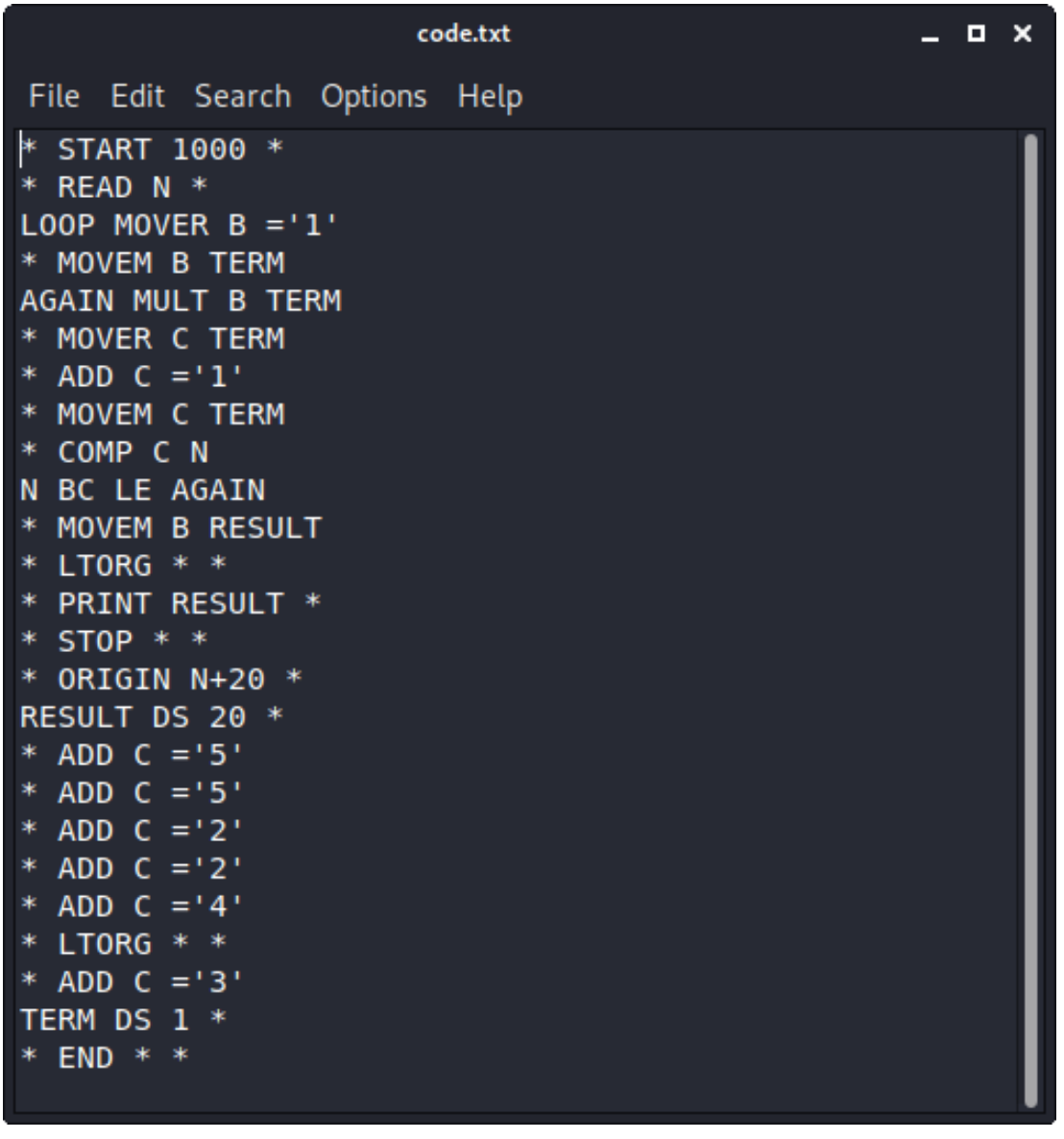
```
      //if mnemonic is a "LTORG" assembler directive
      if(strcmp(a.a1,"AD")==0 && strcmp(a.a2,"3")==0)// for
LTORG
            {
                  //printf("\nMenmonic is LTORG");
                  ltorgflag=1;
            }
      if(ltorgflag==1)
            {
                  updateicbyltrtable();
                  fprintf(fppt,"%d\t#%d\n",ptsr,findtotalPT()+1);
                  //printf("new value in
pt:%d\t#%d\n",ptsr,findtotalPT()+1);

                  ptsr++;

                  ltorgflag=0;
            }//if "LTORG" is encountered


      //if mnemonic is a "EQU" assembler directive
      if(strcmp(a.a1,"AD")==0 && strcmp(a.a2,"4")==0)// for EQU
            {

      updateaddressinst(sc.label,returnsymboladdress(sc.operand1
));
            }//for EQU


      //if mnemonic is a "ORIGIN" assembler directive
      if(strcmp(a.a1,"AD")==0 && strcmp(a.a2,"5")==0)
            {
                  //printf("\nMenmonic is Orgin");
                  if(strcmp(sc.operand1,"*")==0)
                        newadd=address;
                  else
                  {
                        strcpy(tempstr,sc.operand1);
                        token=strtok(tempstr,"+");
```

```
                        newadd=returnsymboladdress(token);
                        //printf("\nreturned value of add
%d",newadd);

      if(strlen(sc.operand1)>strlen(token))
                              {
                                    token=strtok(NULL,"+");
                                    newadd=newadd+atoi(token);

                              }//if

                        if(newadd==-1)
                        {
                              printf("\nError...Forward Reference
Symbol '%s'\n\n",sc.operand1);
                              exit(0);
                        }

                  }//else

            }//for "ORIGIN"



      writedataintoic(a,b,c);
      newadd=newadd+updateaddvalue;
      updateaddvalue=0;

      //printf("%s %s %s
%s\n",sc.label,sc.mnemonic,sc.operand1,sc.operand2);

fscanf(fpcode,"%s%s%s%s",sc.label,sc.mnemonic,sc.operand1,sc.ope
rand2);
    }//outer while


      writeaddofltr();

      fclose(fpcode);
      //fclose(fpltr);
```

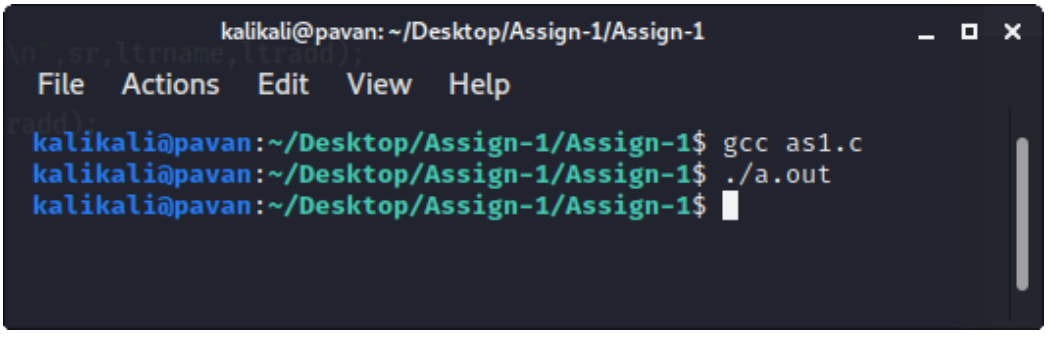
```
      if(issymbolisvalid()==0)
      {
            printf("\nError...Invalid symbol\n");
            exit(0);
      }

fclose(fppt);

}//main
```

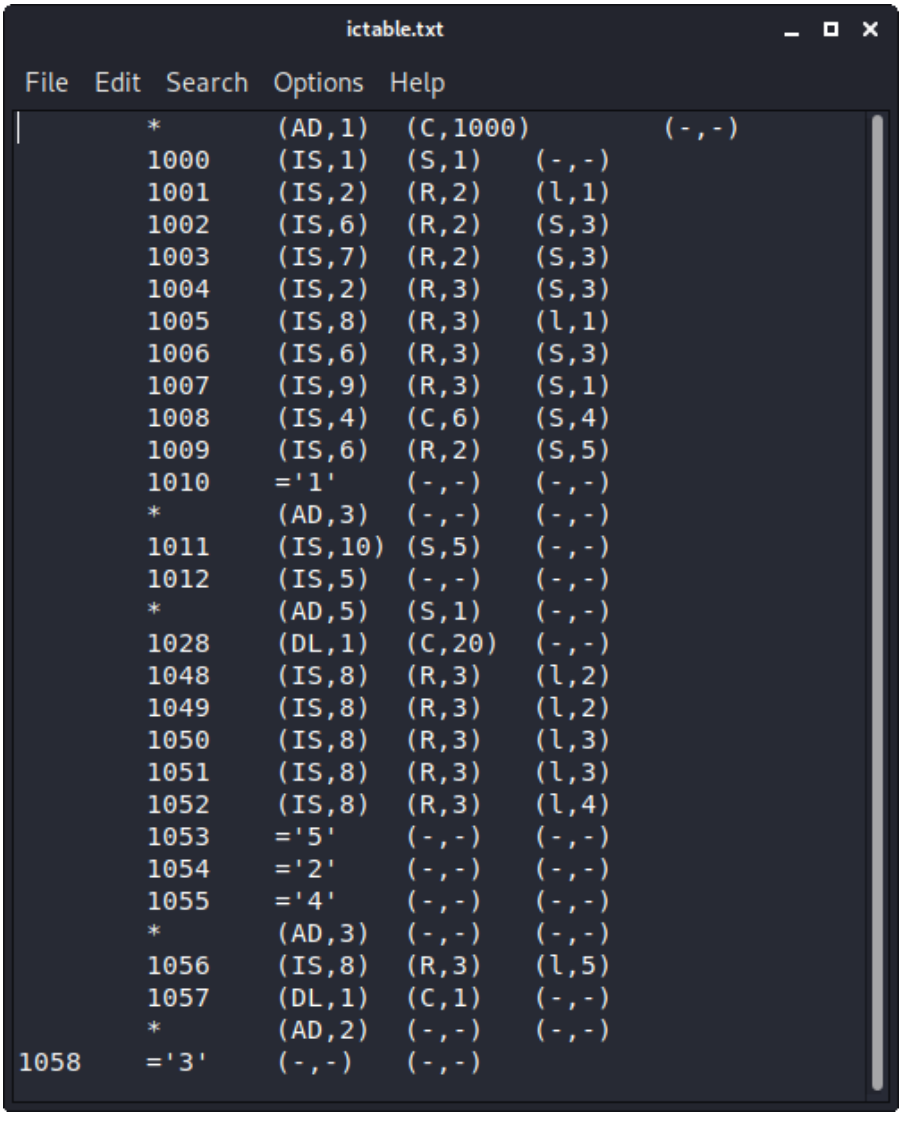
INPUTS :




code.txt
File Edit Search Options Help
* START 1000 *
* READ N *
LOOP MOVER B ='1'
* MOVEM B TERM
AGAIN MULT B TERM
* MOVER C TERM
* ADD C ='1'
* MOVEM C TERM
* COMP C N
N BC LE AGAIN
* MOVEM B RESULT
* LTORG * *
* PRINT RESULT *
* STOP * *
* ORIGIN N+20 *
RESULT DS 20 *
* ADD C ='5'
* ADD C ='5'
* ADD C ='2'
* ADD C ='2'
* ADD C ='4'
* LTORG * *
* ADD C ='3'
TERM DS 1 *
* END * *

OUTPUTS :


kalikali@pavan: ~/Desktop/Assign-1/Assign-1
File Actions Edit View Help
kalikali@pavan:~/Desktop/Assign-1/Assign-1$ gcc as1.c
kalikali@pavan:~/Desktop/Assign-1/Assign-1$ ./a.out
kalikali@pavan:~/Desktop/Assign-1/Assign-1$


ictable.txt
File Edit Search Options Help
* (AD,1) (C,1000) (-,-)
1000 (IS,1) (S,1) (-,-)
1001 (IS,2) (R,2) (l,1)
1002 (IS,6) (R,2) (S,3)
1003 (IS,7) (R,2) (S,3)
1004 (IS,2) (R,3) (S,3)
1005 (IS,8) (R,3) (l,1)
1006 (IS,6) (R,3) (S,3)
1007 (IS,9) (R,3) (S,1)
1008 (IS,4) (C,6) (S,4)
1009 (IS,6) (R,2) (S,5)
1010 ='1' (-,-) (-,-)
* (AD,3) (-,-) (-,-)
1011 (IS,10) (S,5) (-,-)
1012 (IS,5) (-,-) (-,-)
* (AD,5) (S,1) (-,-)
1028 (DL,1) (C,20) (-,-)
1048 (IS,8) (R,3) (l,2)
1049 (IS,8) (R,3) (l,2)
1050 (IS,8) (R,3) (l,3)
1051 (IS,8) (R,3) (l,3)
1052 (IS,8) (R,3) (l,4)
1053 ='5' (-,-) (-,-)
1054 ='2' (-,-) (-,-)
1055 ='4' (-,-) (-,-)
* (AD,3) (-,-) (-,-)
1056 (IS,8) (R,3) (l,5)
1057 (DL,1) (C,1) (-,-)
* (AD,2) (-,-) (-,-)
1058 ='3' (-,-) (-,-)


ltrtable.txt
File Edit Search Options Help
1 ='1' 1010
2 ='5' 1053
3 ='2' 1054
4 ='4' 1055
5 ='3' 1058
*pooltable.txt
File Edit Search Options Help
x1 #1
2 #2
3 #5
symtable.txt
File Edit Search Options Help
1 N 1008
2 LOOP 1001
3 TERM 1057
4 AGAIN 1003
5 RESULT 1028
cond_operator.txt
File Edit Search Options Help
1 EQ 01
2 NEQ 02
3 GT 03
4 GE 04
5 LT 05
6 LE 06

SP Assignment – 2

Pavan S. Patil
Roll no. 51

Page No.
Date / /

Name :- Pavan S. Patil
Roll no. - 51

ASSIGNMENT-2

* Problem Statement: -
Write a program to implement Pass-II of Two-pass assembler for output of Assignment 1
(The subject teacher should provide input file for this assignment)
Objective: -
1. To study basic translation process of assembly language to machine language.
2. To study two pass assembly process.
* Theory: -
Data structures required for pass I:
1. Source file containing assembly program.
2. MOT: A table of mnemonic op-codes and related information. It has the following fields
3. Mnemonic : Such as ADD, END, DC
4. TYPE : IS for imperative, DL for declarative and AD for Assembler directive
5. OPcode : Operation code indicating the operation to be performed.
6. Length : Length of instruction required for Location Counter Processing
7. LITTAB and POOLTAB: Literal table stores the literals used in the program and POOLTAB stores the pointers to the literals in the current literal pool.

Page No.

Date / /

Data Structure used by Pass II:

1. OPTAB: A table of mnemonic opcodes and related information.
2. SYMTAB: The symbol table
3. LITTAB: A table of literals used in the program
4. Intermediate code generated by Pass I
5. Output files containing Target code / error listing.

Conclusion : Thus we have designed and implemented a pass 2 of a 2 pass assembler in C.

Date 2

FAQs

Q.1) what is synthesis phase.

→ a phase of compiler is a distinguishable stage, which take input from the previous stage, processes and yields output that can be used as input for the next stage.

Q.2) what type of error is recognized by Pass 2 of a 2 pass assembler.

→ operand not found
Invalid register ID
Invalid WORD operand.

ASSIGNMENT 2 CODE :

```
//Implimentation of Pass-2 of assembler

#include<stdio.h>
#include<stdlib.h>
#include<string.h>


int returnsymnameaddress(char operandindex[])
{
      FILE *fpsym;
      int symadd;
      char sr[10],symname[30];

            fpsym=fopen("sym.txt","r");
            fscanf(fpsym,"%s%s%d",sr,symname,&symadd);
            while(!feof(fpsym))
            {
                  if(strcmp(sr,operandindex)==0)
                  {
                        return symadd; //if symbol index is found then
return address
                  }
                  fscanf(fpsym,"%s%s%d",sr,symname,&symadd);

            }//check till the end of file
            fclose(fpsym);
return 1; //not found
}//it returns the symbol address from the symbol table


int returnliteraladdress(char operandindex[])
{
      FILE *fpltr;
      int ltradd;
      char sr[10],ltrname[30];

            fpltr=fopen("lit.txt","r");
            fscanf(fpltr,"%s%s%d",sr,ltrname,&ltradd);
            while(!feof(fpltr))
```

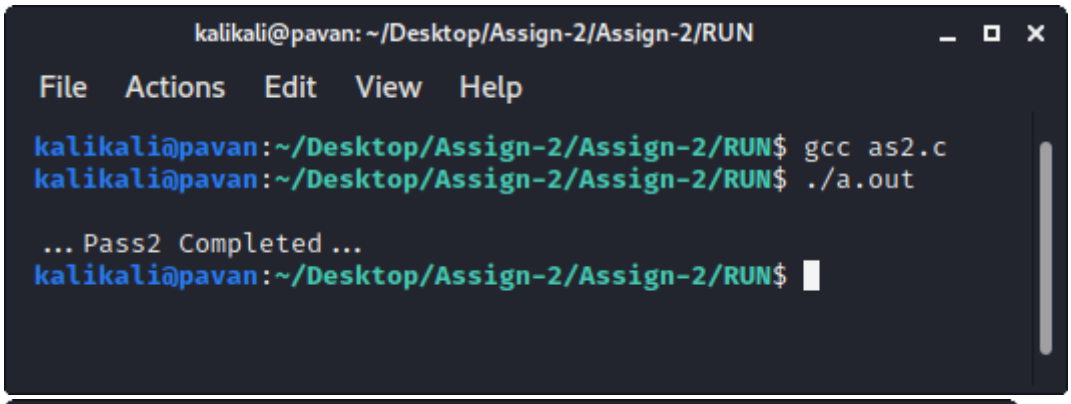
```
           {
                 if(strcmp(sr,operandindex)==0)
                 {
                       return ltradd; //if literal index is found then
return address
                 }
                 fscanf(fpltr,"%s%s%d",sr,ltrname,&ltradd);
           }//check till the end of file
           fclose(fpltr);
return 1;  //not found
}//it returns the literal address from the literal table


int main()
{

      remove("targetcode.txt");

      FILE *fptc,*fpic,*fpsym,*fpltr;
      char address[30],mnemonic[30],operand1[10],operand2[10];
      char
str1_mne[10],str2_mne[10],str3_mne[10],str1_ope1[10],str2_ope1[10],s
tr3_ope1[10],str1_ope2[10],str2_ope2[10],str3_ope2[10];
      int add1,add2;



      fptc=fopen("targetcode.txt","a");
      fpic=fopen("ic.txt","r");
      fpsym=fopen("sym.txt","r");
      fpltr=fopen("lit.txt","r");

      fscanf(fpic,"%s%s%s%s",address,mnemonic,operand1,operand2);
      while(!feof(fpic))
      {
           if(strcmp(address,"*")!=0)
           {
                 if(mnemonic[0]!='=')
                 {
                       //split the string into different tokens
                             //strtok parse the string into a sequence
of tokens...if no more tokens are found then it returns NULL...and
the string does not include delimitong character...
```

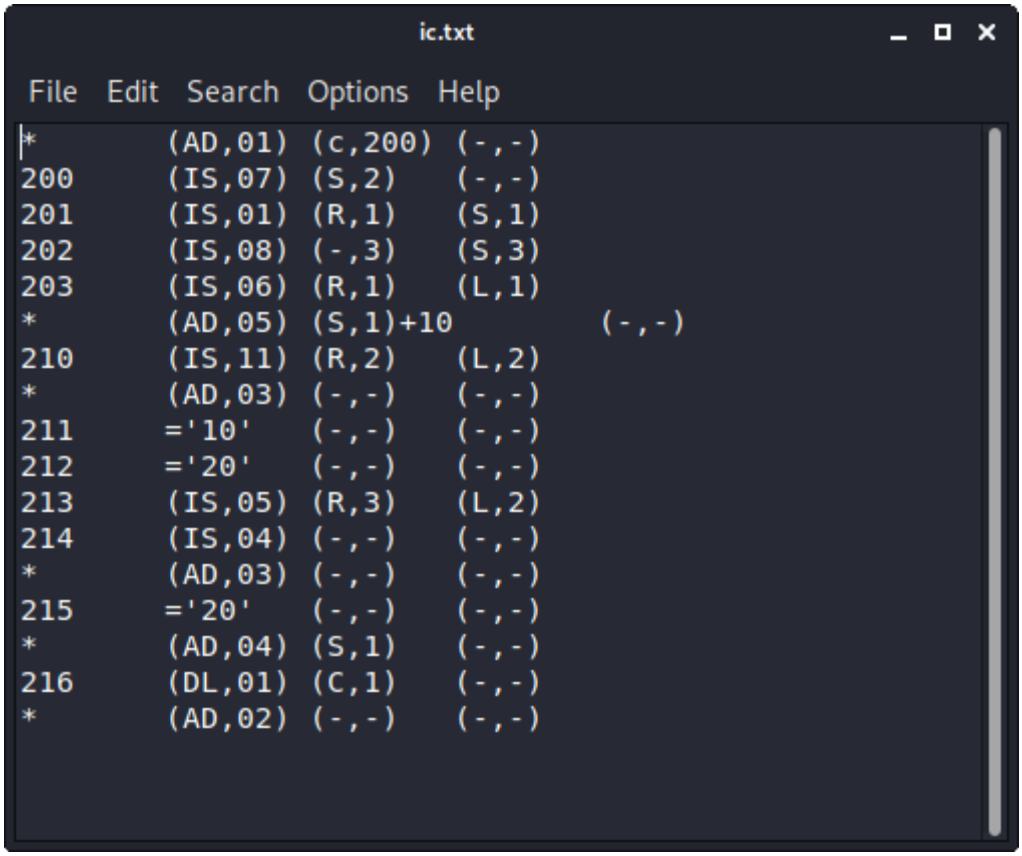
```
                    //for mnemonic
                    strcpy(str1_mne,strtok(mnemonic,","));
                    strcpy(str2_mne,strtok(NULL,","));
                    strcpy(str3_mne,strtok(str2_mne,")"));


                    //for operand1
                    strcpy(str1_ope1,strtok(operand1,","));
                    strcpy(str2_ope1,strtok(NULL,","));
                    strcpy(str3_ope1,strtok(str2_ope1,")"));


                    //for operand2
                    strcpy(str1_ope2,strtok(operand2,","));
                    strcpy(str2_ope2,strtok(NULL,","));
                    strcpy(str3_ope2,strtok(str2_ope2,")"));



                    //now find the address of the symbol or literal
from the literal table
                    //for operand1
                    if(str1_ope1[1]=='S') //bcoz 1st character is
'S'
                    {
                          add1=returnsymnameaddress(str3_ope1);
                          sprintf(str3_ope1,"%d",add1);
                    }//finding address of the symbol from the symbol
table
                    else if(str1_ope1[1]=='L') //bcoz ist character
is 'L'
                    {
                          add1=returnliteraladdress(str3_ope1);
                          sprintf(str3_ope1,"%d",add1);
                    }//finding  address  of  the  literal  from  the
literal table



                    //for operand2
                    if(str1_ope2[1]=='S') //bcoz 1st character is
'S'
                    {
```

```
                        add2=returnsymnameaddress(str3_ope2);
                        sprintf(str3_ope2,"%d",add2);
                    }//finding address of the symbol from the symbol
table
                    else   if(str1_ope2[1]=='L')      //bcoz   ist
character is 'L'
                    {
                        add2=returnliteraladdress(str3_ope2);
                        sprintf(str3_ope2,"%d",add2);
                    }//finding  address  of  the  literal  from  the
literal table


                    else if(str1_mne[1]=='D')
                    {
                        strcpy(str3_mne,"00");  //if  it  is  a
declarative statement then its opcode is "00"
                    }//if it is a declarative statement




    fprintf(fptc,"%s\t%s\t%s\t%s\n",address,str3_mne,str3_ope1,str
3_ope2);

                }//if it is not a literal
                else
                {
                    strcpy(str3_ope1,"-");
                    strcpy(str3_ope2,"-");

    fprintf(fptc,"%s\t%s\t%s\t%s\n",address,mnemonic,str3_ope1,str
3_ope2);
                }//else
           }//if it is not assembler directives then only include it
in the target code



    fscanf(fpic,"%s%s%s%s",address,mnemonic,operand1,operand2);

    }//open ic file and read till the end of file
```

```
        fclose(fptc);
        fclose(fpic);
        fclose(fpsym);
        fclose(fpltr);

printf("\n...Pass2 Completed...\n");
}//main
```

OUTPUT :





lit.txt

File Edit Search Options Help

```
1       ='10'   211
2       ='20'   212
3       ='20'   215
```

sym.txt

File Edit Search Options Help

```
1       C       200
2       A       216
3       LOOP    201
4       S       200
```

targetcode.txt

File Edit Search Options Help

```
200     07      216     -
201     01      1       200
202     08      3       201
203     06      1       211
210     11      2       212
211     ='10'   -       -
212     ='20'   -       -
213     05      3       212
214     04      -       -
215     ='20'   -       -
216     00      1       -
```

SP Assignment – 3

Pavan S. Patil
Roll no. 51

Page No.
Date / /

Name :- Pavan S. Patil
Roll no. - 51

ASSIGNMENT-3

Problem Statement: -
Study Assignment for Macro Processor
Objective: -
1. To understand macro facility, features and its use in assembly language programming.
2. To study how the macro definition is processed and how macro call results in the expansion of code.
Theory:
An assembly language macro facility is to extend the set of operations provided in an assembly language.
In order that programmers can repeat identical parts of their program macro facility can be used. This permits the programmer to define an abbreviation for a part of program & use this abbreviation in the program. This abbreviation is treated as macro definition & saved by the macro processor. For all occurrences the abbreviation i.e. macro call, macro processor substitutes the definition.
Macro definition part:
It consists of
1. Macro Prototype Statement - this declares the name of macro & types of parameters.
2. Model statement - It is statement from which assembly language statement is generated during macro expansion.
3. Preprocessor Statement - It is used to perform auxiliary function during macro expansion.

Page No.
Date / /

Macro Call & Expansion :

The operation defined by a macro can be used by writing a macro name in the mnemonic field and its operand in the operand field. Appearance of the macro name in the mnemonic field leads to a macro call. Macro call replaces such statements by sequence of statement comprising the macro. This is known as macro expansion.

Macro Facilities:

1. Use of AIF & AGO allows us alter the flow of control during expansion.
2. Loops can be implemented using expansion time variables.

Design Procedure

1. Step 1 is definition processing - Scan all macro definitions and for each macro definition enter the macro name in macro name table (MNT). Store entire macro definition in macro definition table (MDT) and add auxiliary information in MNT such as no of positional parameters (#PP) no of key word parameters (#KP), macro definition table position (MDTP) etc.
2. Step 2 is macro expansion - Examine all statement in assembly source program to detect the macro calls. For each macro call locate the macro in MNT, retrieve MDTP, establish the correspondence between formal & actual parameters and expand the macro:

Page No.
Date / /

Data structures required for macro definition processing -

1. Macro Name Table [MNT] - Fields-
Name of macro, #pp (no of positional parameters), # kp(no of keyword parameters),
ev (no of expansion time variables), MDTP (Macro Definition Table Pointer), Keyword Parameters Default Table Position (KPDTP), Sequencing Symbol Table Position (SSTP).

2. Parameter Name Table [PNTAB] - Fields - Parameter Name

3. Expansion Time Variable Name Table [EVNTAB] - Fields - Expansion time variable name

4. Keyword parameter Default Table [KPDTAB] - Fields - Parameter Name, Default value

5. Macro Definition Table [MDT] -
Model Statement are stored in the intermediate code from as: Label, Opcode, Operands.

6. Sequencing Symbol Table Name [SSNTAB] - Fields - Sequencing symbol name.

7. Sequencing Symbol Table [SSTAB] - MDT entry no(#).

Page No.
Date

Algorithm for definition processing:
Before processing any definition initialize KPDTAB_ptr, SSTAB_ptr, MDT_ptr to 0 and MNT_ptr to -1. These table pointers are common to all macro definitions. For each macro definition perform the following steps.
1. Initialize SSNTAB - ptr, PNTAB - ptr to 0 & fields of MNT, # pp, # kp, #ev to 0 and increment MNT_ptr by 1
2. For macro prototype statement from MNT entry
a. Entry name into name field.
b. For each position parameter field
i. Enter name in parameter name table.
ii. Increment PNTAB - ptr by 1.
iii. Increment # pp by 1.
c. KPDTP KPDTAB - ptr d. For each keyword parameter
i. Enter name & default value in KPDTAB.
ii. Increment KPTAB -ptr by 1
iii. Enter name in PNTAB & increment PNTAB - ptr by 1.
iv. Increment # kp by 1.
e. MDTP MDT - ptr
f. # ev Zero and SSTP SSTAB - ptr.
3. DO Begin
Read next statement
a If label field has sequencing symbol (SS) then,
If SS is already present in SSNTAB, retrieve its index in q. Else
Begin

Page No.
Date / /

Entee SS in SSNTAB q SSNTAB - pte
Inceement SSNTAB - pte by one. End
Stoee MDT - pte in SSTAB as STAB [SSTP + q]
MDT - pte. b. Check type of the statement

If Model statement then Begin
i. Foe paeametee geneeate specification (p. # n).
ii. Foe expansion vaeiable geneeate specification (E. # m)
iii. Recoed inteemediate code in MDT.
iv. Inceement MDT - pte by 1.
end
c. If Peepeocessoe Statement - Begin
i. If SET statement then seaech each expansion time vaeiable in EVNTAB & geneeate (E. # m).
ii. If AIF oe AGO then if SS is aleeady peesent in SSNTAB, eeteieve its index in q Else
Begin
Entee SS in SSNTAB q SSNTAB - pte
Inceement SSNTAB - pte by 1 End
Replace symbol by (SS. SSTP+q)

iii) Entee inteemediate code in MDT & inceement MDT - pte by one. End
d. If MEND Begin
Entee MEND in MDT, inceement MDT_pte by 1
If SSNTAB is empty ie.
SSNTAB - pte = = 0 then SSTP = SSTAB_pte
Else

Page No.
Date

SSTAB - pte = SSTAB - pte + SSNTAB - pte.
If #kp == 0 then KPDTP = 0
Return to main logic ie step 6 of main logic.
End.

Data structures required for expansion processing :-
1. Actual parameter table: APTAB
2. Expansion variable table : EVTAB
3. Macro expansion counter : MEC

Conclusion : Thus we have studied macro processor in C.

FAQs

Q.1) Define Term macro.

→ A macro (macroinstruction) used to make sequence of computing instructions available to the programmer as a single program statement.

Q.2) Distinguish between macro & subroutine.

Macro	Subroutine
① macro can be called only in the program it is defined.	Subroutine can be called from other programs also.
② maximum 9 Param.	Any no. of Param.
③ can be called only after its defn.	This is not true for subroutine.
④ macro is defined inside DEFINE ... ... END-OF-DEFN.	Defined outside FORM ... ... ENDFORM

SP Assignment – 4

Pavan S. Patil
Roll no. 51

Page No.
Date / /

Name :- Pavan S. Patil
Roll no. - 51

ASSIGNMENT-4

* Problem Statement: -
Write a program to implement Lexical Analyzer for subset of C
Objective: -
To understand the role of Lexical analyzer
* Theory -
Input:
1) Source File
2) Terminal Symbol Table
Output:
1) Symbol Table
2) Literal Table
3) Universal Table
Theory:
Lexical analysis is the process of converting a sequence of characters into a sequence of tokens.
A program or function which performs lexical analysis is called a lexical analyzer, lexer or scanner.
The specification of a programming language will often include a set of rules which defines the lexer. These rules usually consist of regular expressions and they define the set of possible character sequences that are used to form individual tokens or lexemes whitespace is also defined by a regular expression and influences the recognition of other tokens, but does not itself contribute any tokens.

Page No.
Date

A token is a string of characters, categorized according to the rules as a symbol (e.g. IDENTIFIER, NUMBER, COMMA, etc). The process of forming tokens from an input stream of characters is called tokenization and the lexer categorizes them according to a symbol type. A token can look like anything that is useful for processing an input text stream or text file.

A lexical analyzer generally does nothing with combinations of tokens, a task left for a parser. For example, a typical lexical analyzer recognizes parenthesis as tokens, but does nothing to ensure that each '(' is matched with a ')'. Consider this expression in the C programming language:
sum=3+2;
Tokenized in the following table:

lexeme token	type
sum	Identifier
=	Assignment operator
3	Number
+	Addition operator
2	Number

Tokens are frequently defined by regular expressions, which are understood by a lexical analyzer generator such as lex. The lexical analyzer (either generated automatically by a tool like lex, or hand-crafted) reads in a stream of

Page No.
Date

characters, identifies the lexemes in the steeam and categorizes them into tokens. This is called "tokenizing". If the lexer finds an invalid token, it will report an error

ALGORITHM:

1) Start the program
2) Declare all the variables and file pointers.
3) Display the input program.
4) Separate the keyword in the program and display it
5) Display the header files of the input program.
6) Separate the operators of the input program and display
7) Print the punctuation marks.
8) Print the constant that are present in input program.
9) Print the identifiers of the input program.
10) Stop the program.

FAQs

Q. 1) Explain phase of compiler

→ We basically have two phases of compilers namely Analysis and synthesis phase.

compiler operates in various phases. each phase transforms the source program from one representation to another.

Q. 2) What is identifier ?

→ Identifier or symbol are the names you supply for variables, types, functions and labels in your program.

ASSIGNMENT 4 CODE :

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<ctype.h>

int ustsr=1;
int idsr=1;
int ltrsr=1;

typedef struct ust
{
int srno;
char name[50];
char token[50];
int index;
}ust;

typedef struct idtable
{
int srno;
char name[50];
}idtable;

typedef struct ltrtable
{
int srno;
char name[50];
char base[50];
char scale[50];
int precision;
}ltrtable;

typedef struct tertable
{
int srno;
char terminal[50];
char indicator[50];
}tertable;
```

```
void writedatatoust(ust t1)
{
      FILE *fpust;
      fpust=fopen("ust.txt","a");
      fprintf(fpust,"%d\t\t%-
10s\t\t%10s\t\t%d\n",t1.srno,t1.name,t1.token,t1.index);
      fclose(fpust);
}//writing data into the uniform symbol table



int isidalreadypresent(char key[30])
{
      FILE *fpid;
      int sr;
      char name[30];
      int i=1;
            fpid=fopen("idtable.txt","r");
            while(!feof(fpid))
            {
                  fscanf(fpid,"%d %s ",&sr,name);
                  if(strcmp(key,name)==0)
                  {
                        return i;
                  }
                  i++;
            }
            fclose(fpid);
      return 0 ;
}//searchin for the sr no of the present identifier



void writedatatoltrtable(ltrtable t1)
{
      FILE *fpltr;
      fpltr=fopen("literaltable.txt","a");
      fprintf(fpltr,"%d\t\t%-
10s\t\t%10s\t\t%10s\t\t%d\n",t1.srno,t1.name,t1.base,t1.scale,t1.pre
cision);
      fclose(fpltr);
}//writing data into the literal table
```

```
tertable checkfortokenintertable(char key[30])
{
      tertable t1;
      FILE *fpter;
      int sr;
      char terminal[30];
      char indicator[30];

            t1.srno=-1;
            fpter=fopen("terminaltable.txt","r");
            while(!feof(fpter))
            {
                  fscanf(fpter,"%d %s %s",&sr,terminal,indicator);
                  if(strcmp(key,terminal)==0)
                  {
                        t1.srno=sr;
                        strcpy(t1.terminal,terminal);
                        strcpy(t1.indicator,indicator);
                        return t1;
                  }          //fscanf(f,"%d                              %s
%s",&sr,terminal,indicator);
            }
      fclose(fpter);
return t1 ;
}//check for token is present in the terminal table or not



void writedatatoidtable(idtable t1)
{
      FILE *fpid;
      fpid=fopen("idtable.txt","a");
      fprintf(fpid,"%d\t\t%-10s   \n",t1.srno,t1.name);
fclose(fpid);
}//writing data into the identifier table




int main()
```

```
{

      FILE *f;

      tertable t1;
      ust ust1;
      idtable id;
      ltrtable ltr;

      f=fopen("ust.txt","w");
      fclose(f);
      f=fopen("literaltable.txt","w");
      fclose(f);
      f=fopen("idtable.txt","w");
      fclose(f);

      int flag=0,ltr1,storeltr,i=0,fentry=0,prev_char=0;
      char oldchar='\0',newchar='\0',ch;
      char temp[2],token[50],token1[2];

      FILE *fpsrc,*fpltr,*fpid,*fpust;

      fpsrc=fopen("cprg.txt","r");
      if(fpsrc==NULL)
      {
            printf("Error in Opening Source File");
            return 0;
      }

      while((ch=fgetc(fpsrc))!=EOF) //read complete file
      {
            if(ch=='"')      // If String Found
            {
                  temp[0]='"';
                  temp[1]='\0';
                  ust1.srno=ustsr++;
                  strcpy(ust1.name,temp);
                  strcpy(ust1.token,"TRM");
                  t1=checkfortokenintertable(temp);
                  ust1.index=t1.srno;
                  writedatatoust(ust1);

                  while(1)
```

```
                {
                      ch=fgetc(fpsrc);
                      if(ch=='"')
                      {
                            ch=fgetc(fpsrc);
                            token[i]='\0';
                            break;
                      }
                      else
                            token[i++]=ch;

                }           //read Complete String
                i=0;


                if(strstr(token,"%d")==NULL )
                            {
                            ltr.srno=ltrsr++;
                            strcpy(ltr.name,token);
                            strcpy(ltr.base,"String");
                            strcpy(ltr.scale,"FIXED");
                            ltr.precision=strlen(token);
                            writedatatoltrtable(ltr);;

                                  ust1.srno=ustsr++;
                                  strcpy(ust1.name,token);
                                  strcpy(ust1.token,"LIT");
                                  t1=checkfortokenintertable(temp);
                                  ust1.index=t1.srno;

                                  writedatatoust(ust1);

                            }//If %d Occured in String Then Avoid The
String

                                        strcpy(ust1.name,temp);
                                        strcpy(ust1.token,"TRM");

      ust1.index=isidalreadypresent(temp);
                                        writedatatoust(ust1);

                      }     // For Strings
                      else if(isdigit(ch))
```

```
	{
		while(isdigit(ch))
		{
			if(ch==' ')
			{
				token[i]='\0';
				break;
			}
			else
			token[i++]=ch;
			ch=fgetc(fpsrc);

		}
			i=0;

		ltr.srno=ltrsr++;
		strcpy(ltr.name,token);
		strcpy(ltr.base,"Decimal");
		strcpy(ltr.scale,"FIXED");
		ltr.precision=strlen(token);
		writedatatoltrtable(ltr);;

			ust1.srno=ustsr++;
			strcpy(ust1.name,token);
			strcpy(ust1.token,"LIT");
			ust1.index=ltr.srno;
			writedatatoust(ust1);

	}	// For Constants


if(isalnum(ch))
{
	if(flag==1)
	{
		token[i++]=ch;
		token[i]='\0'; //terminate token
		i=0;           //again read token
		temp[0]=ch;
		temp[1]='\0';
		flag==0;
		ust1.srno=ustsr++;
		strcpy(ust1.name,token);
```

```
                        strcpy(ust1.token,"IDT");
                        ust1.index=isidalreadypresent(temp);
                        writedatatoust(ust1);
                  }
                  else
                        token[i++]=ch;
            }
            else if(ch==' ' || ch=='\n' || ch==',' || ch=='(' ||
ch ==')' || ch==';' )
            {

                  token[i]='\0'; //terminate token
                  i=0;           //again read token
                  //printf("\nToken is: %-10s",token);

                  t1=checkfortokenintertable(token);
                  if(t1.srno!=-1)
                  {
                        if(strcmp(t1.indicator,"N")==0)
                        {
                              ust1.srno=ustsr++;
                              strcpy(ust1.name,token);
                              strcpy(ust1.token,"KWT");
                              ust1.index=t1.srno;
                              writedatatoust(ust1);

                        }
                        if(strcmp(t1.indicator,"Y")==0)
                        {
                              ust1.srno=ustsr++;
                              strcpy(ust1.name,token);
                              strcpy(ust1.token,"TRM");
                              ust1.index=t1.srno;
                              writedatatoust(ust1);
                        }
                  }    // If Found In Terminal Table Then Update
UST
                  else
                  {
                        if(strcmp(token,"  ")==1  &&  (fentry==1))
//check whether first entry in identifier happened or not
                        {
```

```
                              //printf("Identifier  Search  Result
%d",isidalreadypresent(token));

                              if(isidalreadypresent(token)==0)
      //if identifier not found in idtable
                              {
                                    id.srno=idsr++;
                                    strcpy(id.name,token);
                                    writedatatoidtable(id);

                                    ust1.srno=ustsr++;
                                    strcpy(ust1.name,token);
                                    strcpy(ust1.token,"IDN");
                                    ust1.index=id.srno;
                                    writedatatoust(ust1);
                              }
                              else
                              {
                                    ust1.srno=ustsr++;
                                    strcpy(ust1.name,token);
                                    strcpy(ust1.token,"IDN");

      ust1.index=isidalreadypresent(token);
                                    writedatatoust(ust1);
                              }

                        }
                        else if(strcmp(token," ")==1)
                        {
                              id.srno=idsr++;
                              strcpy(id.name,token);
                              //printf("First Entry");
                              writedatatoidtable(id);
                              fentry=1;

                              ust1.srno=id.srno;
                              strcpy(ust1.name,token);
                              strcpy(ust1.token,"IDN");
                              ust1.index=id.srno;
                              writedatatoust(ust1);

                              //printf("\nWritten %s",token);
                        }
```

```
                        } //Else IT is Identifier Update Indentifier
Table

                        token[i++]=ch;
                        token[i]='\0'; //terminate token
                        //printf("\nToken:%-10s",token);
                        t1=checkfortokenintertable(token);
                        if(t1.srno!=-1)
                        {
                              if(strcmp(t1.indicator,"Y")==0)
                              {
                                    //printf("   ->Terminal");

                                    ust1.srno=ustsr++;
                                    strcpy(ust1.name,token);
                                    strcpy(ust1.token,"TRM");
                                    ust1.index=t1.srno;
                                    writedatatoust(ust1);
                              }
                        }
                        i=0;       // This Is For Condtion operators
inclusions in token else those were lost

                  }
                  else       // Special Case Characters
                  {
                        if(ch=='&')
                        {
                              token[i++]=ch;
                              flag=1;
                        }
                        else if(ch=='+' || ch=='-')
                        {
                              flag=0;
                              token[i++]=ch;
                        }
                        else
                        {
                              flag=0;
                              token[i++]=ch;
                              token[i]='\0';
                              i=0;
```

```
                        t1=checkfortokenintertable(token);
                        if(t1.srno!=-1)
                        {
                              if(strcmp(t1.indicator,"Y")==0)
                              {
                                    ust1.srno=ustsr++;
                                    strcpy(ust1.name,token);
                                    strcpy(ust1.token,"TRM");
                                    ust1.index=t1.srno;
                                    writedatatoust(ust1);
                              }
                        }

                  }


            }

      }//whiile
return 0;
}//main
```

INPUTS :

cprg.txt

File Edit Search Options Help

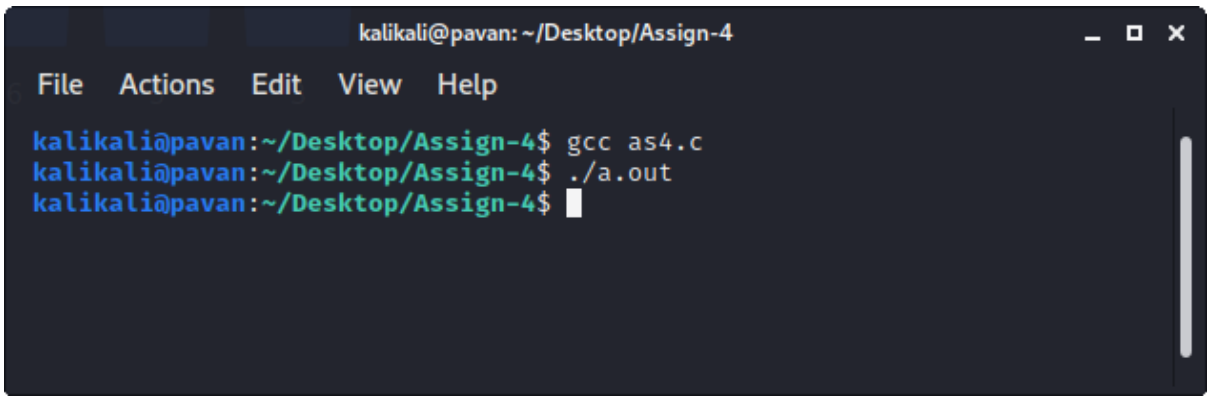
```
#include<stdio.h>
void main(){
        int a,b,c;
        a=5;
        b=6;
        c=a+b;
        printf("Addition: %d",&c);
}
```

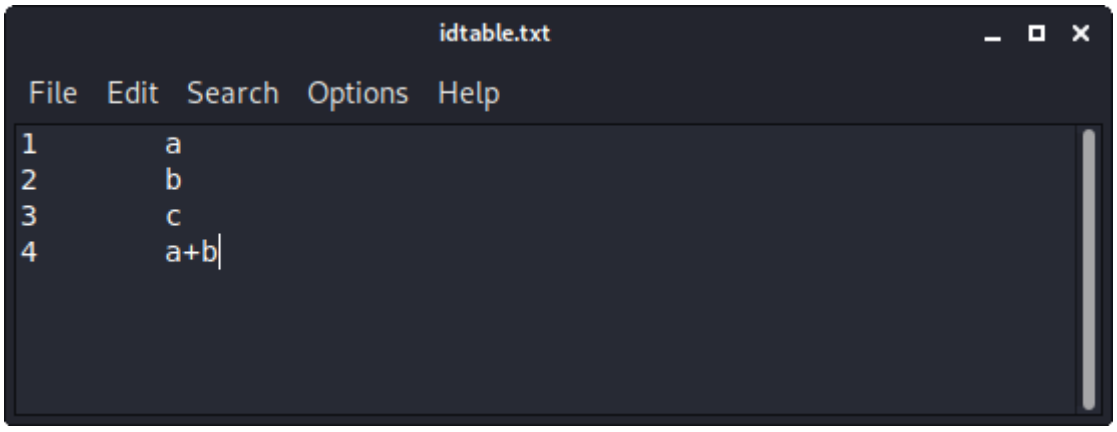
terminaltable.txt

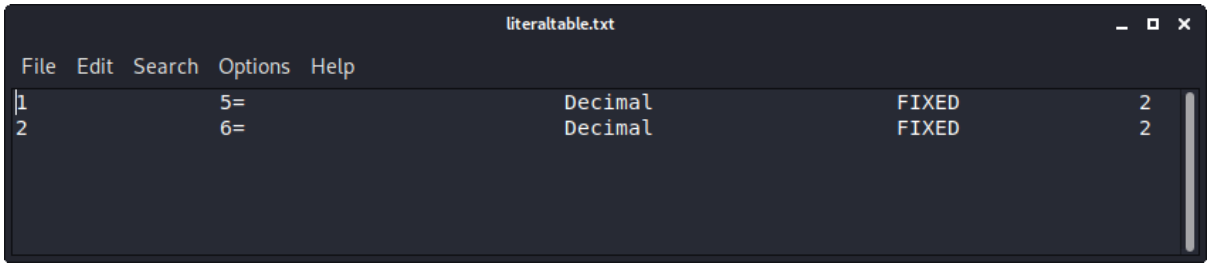
File Edit Search Options Help

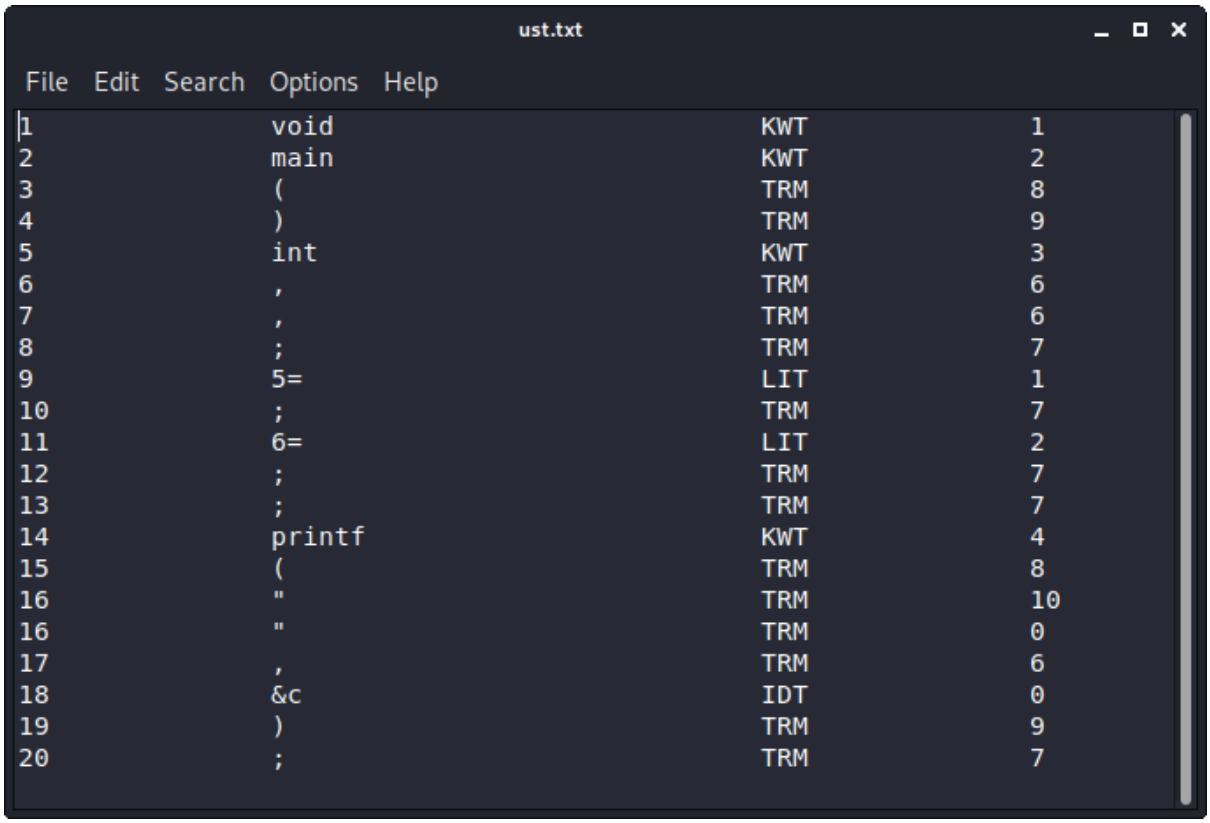
```
1       void    N
2       main    N
3       int     N
4       printf  N
5       scanf   N
6       ,       Y
7       ;       Y
8       (       Y
9       )       Y
10      "       Y
11      =       Y
```

OUTPUTS :


kalikali@pavan: ~/Desktop/Assign-4
File Actions Edit View Help
kalikali@pavan:~/Desktop/Assign-4$ gcc as4.c
kalikali@pavan:~/Desktop/Assign-4$ ./a.out
kalikali@pavan:~/Desktop/Assign-4$


idtable.txt
File Edit Search Options Help
1 a
2 b
3 c
4 a+b


literaltable.txt
File Edit Search Options Help
1 5= Decimal FIXED 2
2 6= Decimal FIXED 2


ust.txt
File Edit Search Options Help
1 void KWT 1
2 main KWT 2
3 (TRM 8
4) TRM 9
5 int KWT 3
6 , TRM 6
7 , TRM 6
8 ; TRM 7
9 5= LIT 1
10 ; TRM 7
11 6= LIT 2
12 ; TRM 7
13 ; TRM 7
14 printf KWT 4
15 (TRM 8
16 " TRM 10
16 " TRM 0
17 , TRM 6
18 &c IDT 0
19) TRM 9
20 ; TRM 7

SP Assignment – 5

Pavan S. Patil
Roll no. 51

Page No.
Date / /

Name :- Pavan S. Patil
Roll no. - 51

ASSIGNMENT-5

Problem Statement: - Recursive Descent parser for assignment statement

Objective: -
To understand the process of parsing & to study bottom-up & top down parsing.

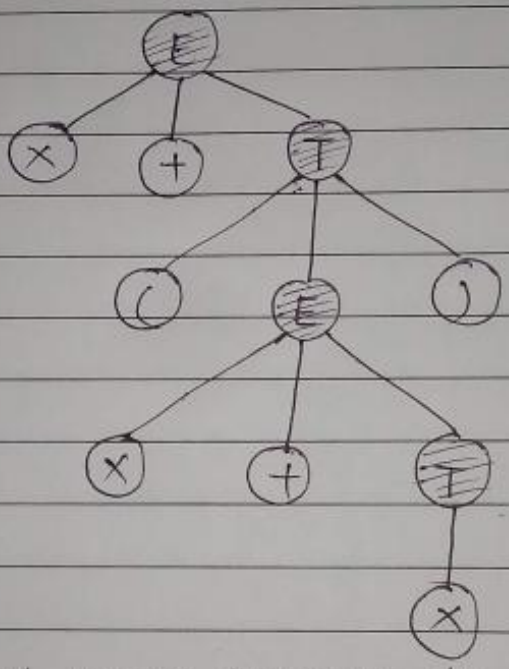
Theory:
One of the most straightforward forms of parsing is recursive descent parsing. This is a top-down process in which the parser attempts to verify that the syntax of the input stream is correct as it is read from left to right. A basic operation necessary for this involves reading characters from the input stream and matching them with terminals from the grammar that describes the syntax of the input. Our recursive descent parsers will look ahead one character and advance the input stream reading pointer when proper matches occur.
What a recursive descent parser actually does is to perform a depth-first search of the derivation tree for the string being parsed. This provides the "descent" portion of the name. The recursive portion comes from the parser's form, a collection of recursive procedures.
As our first example, consider the simple grammar

E = X + T

T = (E)

T = X

cund the derivation tree in figure 2 for the expression x + (x+x)



A recursive descent parser traverses the tree by first calling a procedure to recognize an E. This procedure reads an 'x' and a '+' and then calls a procedure to recognize a T.

Input:

1. Grammar Rules 2. String

Output: Evaluation of String

Algorithm: 1 Give input string. 2. Mention Grammar.

3. Compare each token of string with grammar.

4. If token satisfies the grammar rule then return the evaluation of string else evaluation not possible.

Date 9.

FAQs

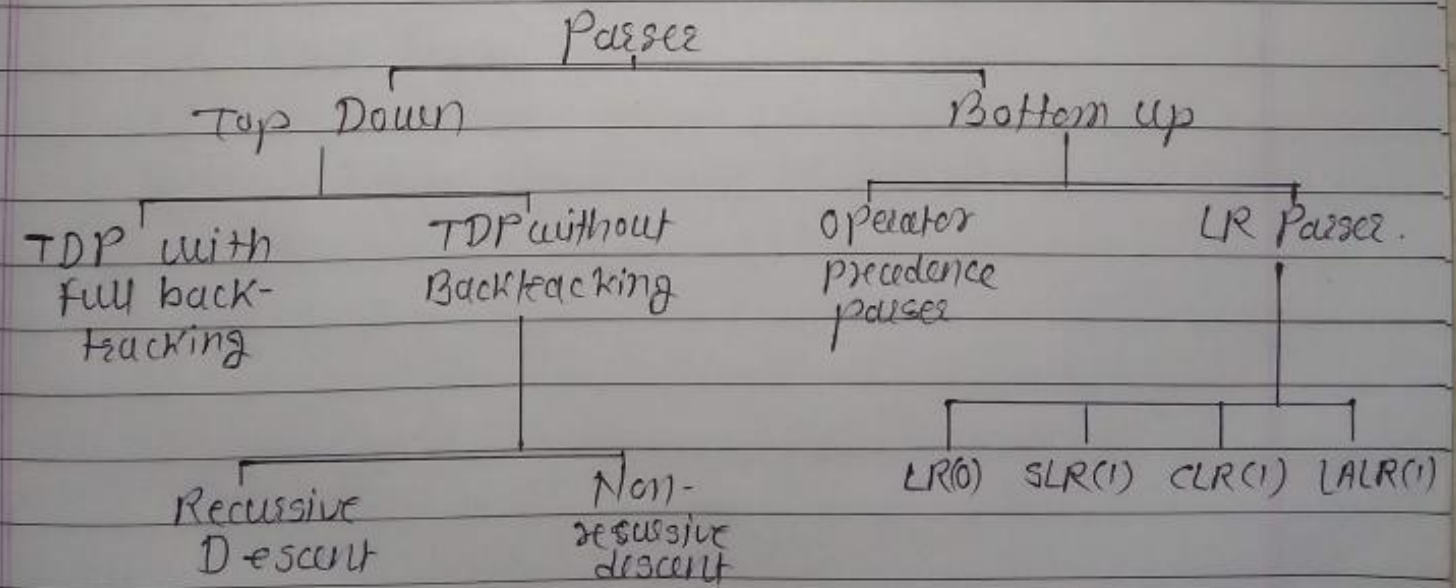
Q.1) What is Parser ?

→ A parser is a compiler or interpreter component that breaks data into smaller elements for easy translation into another language.

Q.2) What is functionality of Parser ?

→ A parser takes input in the form of sequence of tokens, interactive commands or program instructions and breaks them up into parts that can be used by other components in programming.

Q.3) Types of parser -

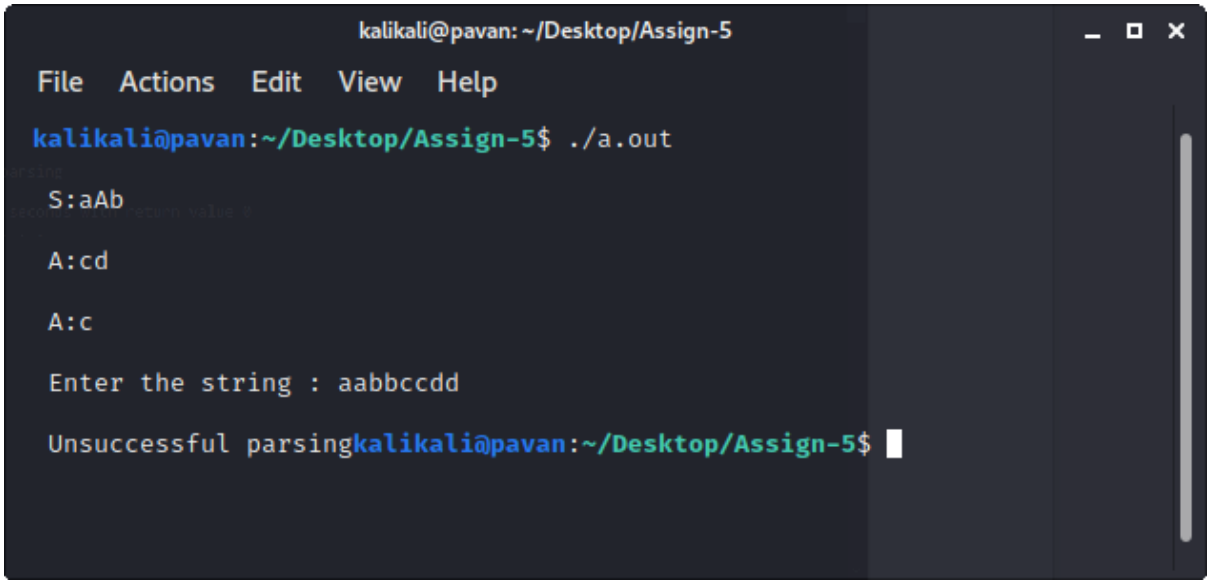


ASSIGNMENT 5 CODE :

```
#include<stdio.h>
#include<string.h>
char inp[30];
int ptr=0;
int A()

{
  if (inp[ptr] == 'c') {
    ptr++;
    if (inp[ptr] == 'd') {
      ptr++;
    } else return 0;
  }
  return 1;
}
int S() {
  int s;
  if (inp[ptr] == 'a') {
    ptr++;
    s = A();
    if (A() != 0) {
      if (inp[ptr] == 'b') {
        ptr++;
        if (inp[ptr] == '$') {
          return 1;
        } else return 0;
      } else return 0;
    }
  } else return 0;
}
int main() {
  int length;
  printf("\n S:aAb \n");
  printf("\n A:cd \n");
  printf("\n A:c \n");
  printf("\n Enter the string : ");
  scanf("%s", inp);
  length = strlen(inp);
  inp[length] = '$';
  if (S() != 0) {
    printf("\n Successful completion of parsing");
```

```
  } else {
    printf("\n Unsuccessful parsing");
  }
} //end main
```

OUTPUTS :



SP Assignment – 6

Pavan S. Patil
Roll no. 51

Page No.
Date / /

Name :- Pavan S. Patil
Roll no. - 51

ASSIGNMENT-6

Problem Statement: -
Implementation of Calculator using LEX and YACC.

Objective: -
1. To understand the role of LEX and YACC.

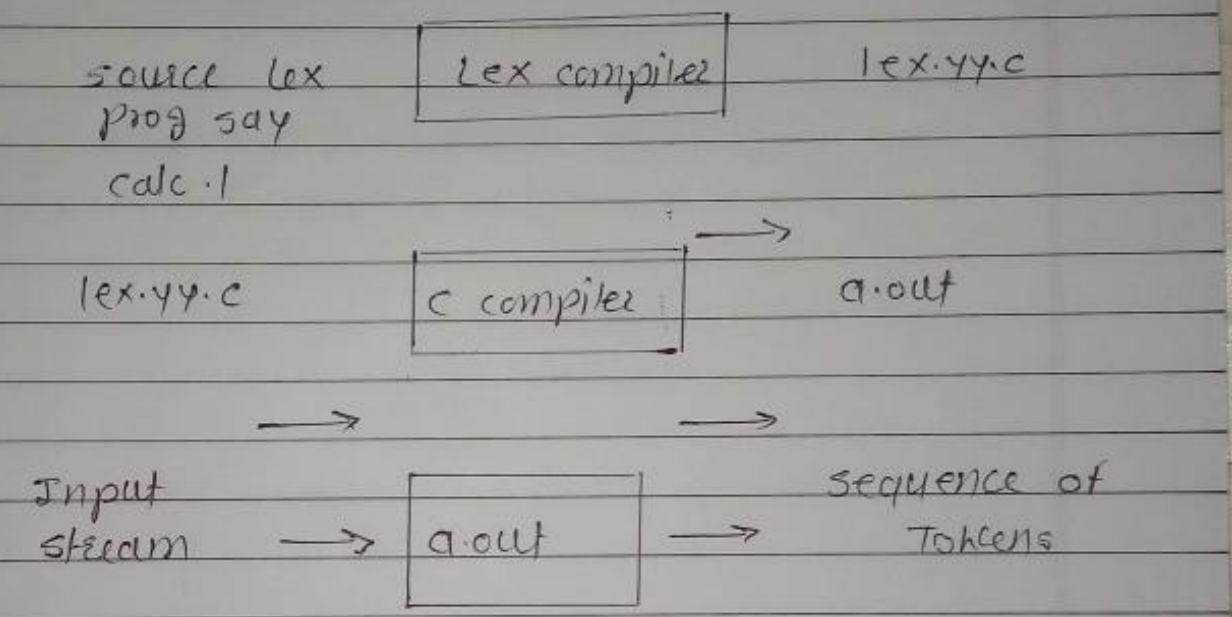
Theory
Lex several tools have been built for constructing lexical analyzers from special purpose notations based on regular expressions several algorithms exist for compiling regular expressions into pattern matching algorithms Lex is a tool which uses such algorithm

Lex is a tool widely used to specify lexical analyzers for variety of languages. The tool shows how the speciation of patterns using regular expression can be combined with actions that LA may be required to perform.

Specification of Lexical Analyzer is prepared by creating a program lex1 in Lex language. Lex1 is run through Lex compilers to produce yy.c which consist of a tabular representation of transition diagram constructed from regular expression of Lex1, together with a standard routine that uses

Page No.
Date

the table to recognize lexemes actions associated with regular expressions in Lex.l R pieces of C code and are carried over directly to Lex.yy.c finally lex.yy.c is run through C compiler to produce a.out which is the lexical analyzer that transforms an input stream into a sequence of tokens



Lex Specifications:

Lex program consist of 3 parts

Declarations

%% %%

Transition Rules

%% %%

Auxiliary Procedures

Declaration Section Includes

Declarations of variables manifest constants and regular expressions.

/*

Example of Declaration

*/

Page No. _____

Date ___/___/___

```
%{
#include "y.tab.h"
extern int yylval;
%}
/*
Example of Regular Definitions */
delim
[ \t \n ]
ws
[ delim ] +
letter
[a-z to A-Z]
digit
[0-9]
id
{letter} ( {letter} / {digit} )*
number
{digit} + (\.{digit}+) ? (E [+\-] ? {digit} +) ?
%%
/*
Transition Rules
*/
{ws}
{ /* No action, no return */ }
{number}
{ yylval = install [] num (); return (NUMBER); }
{id}
```

Page No.
Date / /

{yylval = install() id(); return (ID); }
%%
/*
Auxiliary Procedures
*/
install id()
{
/*
*/
}
install_num()
{
/*
*/
}
Explanation

Declarations are surrounded by special brackets %{ and %}.
Anything appearing between these brackets is copied directly into lex.yy.c & is not treated as a part of regular definitions or translation rules. Same is applied to the auxiliary procedures in the 3 section & these procedures are copied into lex.yy.c
Regular Definitions Each such definitions consist of a name and a regular expression denoted by that name.
? meta symbol - Zero or more occurrences of \ .

Shrikrupa

Page No.
Date

- decimal point , since "⊠ , itself eepeesents chaeactee class of chaeactees
[+\-] - slash is eequieed befoee -, since - also eepeesents eange.

Teanslation Rules Steuctuee of LA is such that it keeps teying to eecognize tokens, until action associated with one found causes a eetuen.

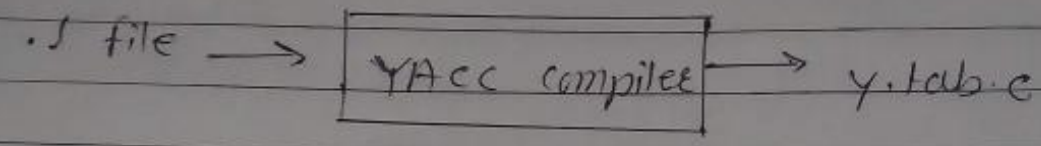
Yylval is a vaeiable whose definition appeaes in Lex output lex.yy.c and which is also available to paesee. Puepose of yylval is to hold the lexical value eetuened. "eetuen statement can only eetuen a code foe token class.

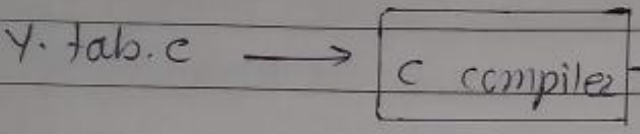
Vaeiable yytext coeeesponds to the pointee to the fiest chaeactee of lexeme & yyleng is integee telling how long the lexeme is.

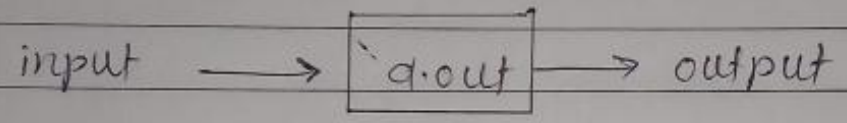
Paesee Geneeatoes
YACC - yet anothee compilee - compilee facilitates the consteuction of feont end of the Yacc specification

Page No.

Date







YACC source prog has 3 parts

Declarations

%%

Translation rules

%%

Supporting C routines

Declaration Part

1. Ordinary "C" declarations within %{ %}
2. Token Declarations

Translation rules part %% & then rules each translation rule consist of a grammar production & associated semantic action.

Eg. expr : expr "+" expr { $$ = $1+ $3; } Expr "-" expr {$$ = $1 - $3; }

;

Page No. _____

Date ____/____/____

1) aee quoted single chaeactees of type 'c☒ and aee teeated as teeminal symbols

2) Unquoted steings of lettees and digits not declaeed to be tokens aee taken as non-teeminals -expe.

3) Alteenative eight sides can be sepaeated by a veetical bae and a semicolon follows each left side with its alteenatives & theie semantic actions fiest left side is taken as staet symbol.

YACC semantic action is a sequence of 'C statements. In semantic action $$ eefees to the value associated with non teeminal on left and while $i eefees to the value associated with ith geammae symbol [T \ NT] on eight.
Suppoeting C - eOutines - othee peoceduees such as eeeoe eecoveey eOutines may be peOvided.

Communication between Lex and YACC
YyJexO) peOduces paies containing token and its associated value. If token NUMBER is eetuen token NUMBER must be declaeed in fiest section of YACC. The atteibute value associated with a token is communicated to the paesee theough YACC defined vaeiable yylval.

Conclusion: Thus we have implemented a calculatoe using LEX and YACC.

Date

FAQs

Q.1) Explain lex.

→ lex is a program that generates lexical analyzer it is used with YACC Parser generator.

Q.2) what is yacc.

→ Yacc (yet another compiler compiler) is standard parser generator for unix operating system.

an open source program, yacc generates code for the parser in C language.

ASSIGNMENT 6 CODE :

//Calci.y:

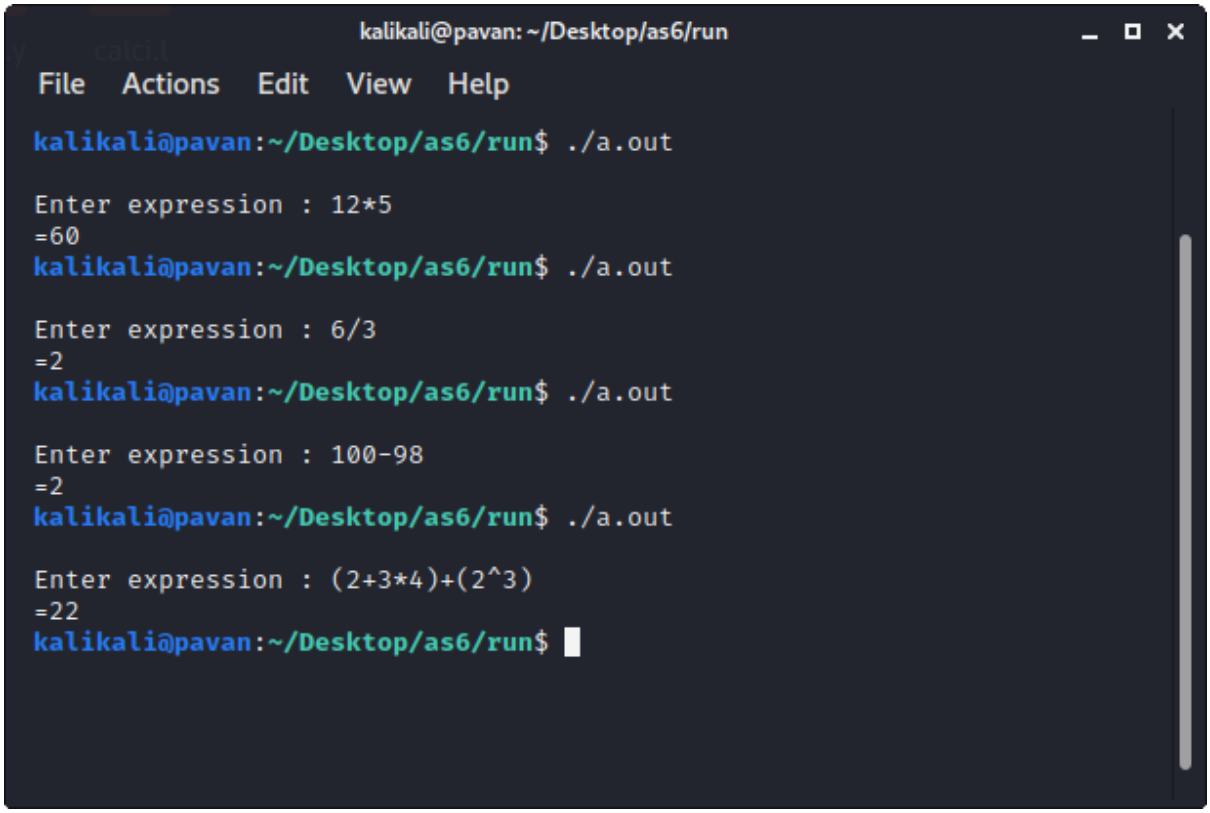
```
%{
      #include<stdio.h>
      #include<stdlib.h>
      int yylex(void);
      void yyerror(char *);
      int i=0,j=0,k=0,n=0;
%}

%token NUM
%left '+''-'
%left '*''/'
%left '^'
%left '(')'

%%

op : expn  { printf("=%d\n",$1); }
;
expn : expn '+' expn  { $$=$1+$3 ; }
     | expn '-' expn  { $$=$1-$3 ; }
     | expn '*' expn  { $$=$1*$3 ; }
     | expn '/' expn  {
                        if($3==0)
                        {
                              printf("\nDivide by 0 error\n");
                              exit(0);
                        }
                        else
                              $$=$1/$3 ;
                   }
     | expn '^' expn  { /*$$=pow($1,$3)*/
                  if($3==0)
                        $$=1;
                  else
                  {
                        n=$3;
                        j=$1;
```

```
                        k=j;
                        for(i=1;i<n;i++)
                              k=k*j;
                        $$=k;
                  }
                   }
      | '(' expn ')'   { $$=$2 ; }
      | '-' expn       { $$=-$2 ;}
      | '+' expn       { $$=+$2 ;}
      | NUM      { $$=$1 ; }
;
%%

int main()
{
      printf("\nEnter expression : ");
      yyparse();
}

void yyerror(char *str)
{
      printf("%s",str);
      exit(0);
}
```

//Calci.l:

```
%{
#include "y.tab.h"
extern int yylval;
void yyerror(char *);
%}
%%
[0-9]+ { yylval=atoi(yytext); return NUM; }
[\t] ;
[-+*/^()] { return yytext[0]; }
\n { return 0; }
. { ECHO ; yyerror("Unexpected character...."); }
%%
```

OUTPUT :


kalikali@pavan: ~/Desktop/as6/run
File Actions Edit View Help
kalikali@pavan:~/Desktop/as6/run$ ./a.out
Enter expression : 12*5
=60
kalikali@pavan:~/Desktop/as6/run$ ./a.out
Enter expression : 6/3
=2
kalikali@pavan:~/Desktop/as6/run$ ./a.out
Enter expression : 100-98
=2
kalikali@pavan:~/Desktop/as6/run$ ./a.out
Enter expression : (2+3*4)+(2^3)
=22
kalikali@pavan:~/Desktop/as6/run$

SP Assignment – 7

Pavan S. Patil
Roll no. 51

Page No.
Date / /

Name :- Pavan S. Patil
Roll no. - 51

ASSIGNMENT-7

Problem Statement: -
Intermediate code generation using LEX &YACC for Control Flow and Switch Case statements.

Objective: -
1 To understand intermediate code generation using LEX and YACC

Theory
LEX
The unix utility lex parses a file of characters. It uses regular expression matching; typically it is used to "tokenize" the contents of the file. In that context, it is often used together with the yacc utility. However, there are many other applications possible.
Structure of a lex file
A lex file looks like
definitions
%%
rules
%%
code

Definition Section:
C code: Any indented code between %{ and %} is copied to the C file. This is typically used for defining file variables, and

Page No.
Date

foe peototypes of eoutines that aee defined in the code segment.
definitions A definition is veey much like a #define cpp diective. Foe example lettee [a-zA-Z]

digit [0-9] punct [.:!?] nonblank [^ \t]
These definitions can be used in the eules section: one could staet a eule {lettee}+ {..
state definitions If a eule depends on context, it's possible to inteoduce states and incoepoeate those in the eules. A state definition looks like %s STATE, and by default a state INITIAL is aleeady given.

Rules section
The eules section has a numbee of patteen-action paies. The patteens aee eegulae expeessions , and the actions aee eithee a single C command, oe a sequence enclosed in beaces.
Usee code section
If the lex peogeam is to be used on its own, this section will contain a main peogeam. If you leave this section empty you will get the default main:
int main()
{
yylex(); eetuen 0;
}
Wheee yylex is the paesee that is built feom the eules.

Page No.
Date / /

YACC

The unix utility yacc (Yet Another Compiler Compiler) parses a stream of token, typically generated by lex, according to a user-specified grammar.

Structure of a yacc file:

A yacc file looks much like a lex file:

```
definitions
%%
rules
%%
code
```

Definitions :

All code between %{ and %} is copied to the beginning of the resulting C file.

Rules:

A number of combinations of pattern and action: if the action is more than a single command it needs to be in braces.

Code:

This can be very elaborate, but the main ingredient is the call to yylex, the lexical analyser. If the code segment is left out, a default main is used which only calls yylex.

Intermediate Code Generation

We could translate the source program directly into the target language. However, there are benefits to having an intermediate, machine-independent representation.

1. A clear distinction between the machine-independent and machine-dependent parts of the compiler

Page No.
Date

3-ADDRESS CODE

The general form is x := y op z, where "op" is an operator, x is the result, and y and z are operands. x, y, z are variables, constants, or "temporaries". A three-address instruction consists of at most 3 addresses for each statement.

A TAC instruction has at most one operator on the RHS of an instruction; no built-up arithmetic expressions are permitted.

e.g. x + y * z can be translated as t1 = y * z
t2 = x + t1

Where t1 & t2 are compiler-generated temporary names.

Types of three address code

There are different types of statements in source program to which three address code has to be generated. Along with operands and operators, three address code also use labels to provide flow of control for statements like if-then-else, for and while. The different types of three address code statements are:3-ADDRESS CODE

The general form is x := y op z, where "op" is an operator, x is the result, and y and z are operands. x, y, z are variables, constants, or "temporaries". A three-address instruction consists of at most 3 addresses for each statement.

o provide flow of control for statements like if-then-else,

Page No.
Date / /

for and while. The different types of three address code statements are: A TAC instruction has at most one operator on the RHS of an instruction; no built-up arithmetic expressions are permitted.

e.g. x + y * z can be translated as t1 = y * z
t2 = x + t1
Where t1 & t2 are compiler-generated temporary names.

Types of three address code
There are different types of statements in source program to which three address code has to be generated. Along with operands and operators, three address code also use labels labels to provide flow of control for statements like if-then-else, for and while. The different types of three address code statements are:

Assignment statement a = b op c
In the above case b and c are operands, while op is binary or logical operator. The result of applying op on b and c is stored in a
Unary operation
a = op b This is used for unary minus or logical negation.
Example: a = b * (- c) + d
Three address code for the above example will be t1 = -c
t2 = t1 * b
t3 = t2 + d a = t3
Copy Statement a = b
The value of b is stored in variable a.

Page No.
Date

Unconditional jump goto L
Ceeates label L and geneeates theee-addeess code "goto L"
v. Ceeates label L, geneeate code foe expeession exp. If the exp eetuens value teue then go to the statement labelled L. exp eetuens a value false go to the statement immediately following the if statement.
Function call
Foe a function fun with n aeguments a1,a2,a3...an ie, fun(a1, a2, a3...an),
the theee addeess code will be Paeam a1
Paeam a2
...
Paeam an

Call fun, n
Wheee paeam defines the aeguments to function.
Aeeay indexing
In oedee to access the elements of aeeay eithee single dimension oe
multidimension, theee addeess code eequiees base addeess and offset value. Base addeess consists of the addeess of fiest element in an aeeay. Othee elements of the aeeay can be accessed using the base addeess and offset value.
Example: x = y[i]
Memoey location m = Base addeess of y + Displacement i
x = contents of memoey location m
similaely x[i] = y
Memoey location m = Base addeess of x + Displacement i

Page No.
Date / /

The value of y is stored in memory location m
Pointer assignment

x = &y x stores the address of memory location y
x = *y y is a pointer whose r-value is location
*x = y sets r-value of the object pointed by x to the r-value of y

Intermediate representation should have an operator set which is rich to implement most of the operations of source language. It should also help in mapping to restricted instruction set of target machine.

Implementation of 3 address code
There address code is represented as record structure with fields for operator and operands. These records can be stored as array or linked list. Most common implementations of there address code are- Quadruples, Triples and Indirect triples.

Quadruples-
Quadruples consists of four fields in the record structure. One field to store operator op, two fields to store operands or arguments arg1 and arg2 and one field to store result res. res = arg1 op arg2 Example: a = b + c
b is represented as arg1, c is represented as arg2, + as op and a as res.

Page No.

Date

Tripples

Tripples uses only three fields in the record structure. One field for operator, two fields for operands named as arg1 and arg2. Value of temporary variable can be accessed by the position of the statement the computes it and not by location as in quadruples.

Example: a = -b * d + c + (-b) * d

Indirect Tripples

Indirect tripples are used to achieve indirection in listing of pointers. That is, it uses pointers to tripples than listing of tripples themselves.

Syntax Trees

Syntax trees are high level IR. They depict the natural hierarchical structure of the source program. Nodes represent constructs in source program and the children of a node represent meaningful components of the construct. Syntax trees are suited for static type checking3 address code for control flow statements: If then else

if(a<b) then x=y+z

else p=q+r

3 address code:

1. if(a<b) goto 3
2. goto 6
3. t1=y+z
4. x=t1
5. goto 8
6. t2=q+r
7. p=t2

Page No.

Date / /

While statement while(a<b)

do x=y+z

3 address code

1. if(a<b) goto 3
2. goto 6
3. t1=y+z
4. x=t1
5. goto 1

do while statement do

x=y+z while a<b

3 address code

1. t1=y+z
2. x=t1
3. if a<b goto 1
4. goto

For statement for(i=1;i<=20;i++) x=y+z

3 address code

1. i=1
2. if i<=20 goto 7
3. goto 10
4. t1=i+1
5. i=t1
6. goto 2
7. t2=y+z
8. x=t2
9. goto 4
10.

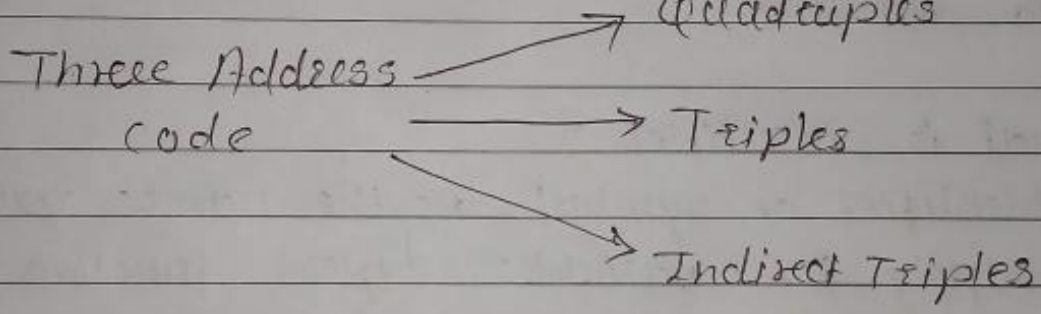
Page No.
Date 6 / /

FAQs

Q.1) What is meant by triples & quadruples ?

→ - Three address code is a form of an intermediate code.

- It is generated by compiler for implementing code optimization.



Q. 2) What are different types of intermediate Representations

→ (1) Structured (graph or tree based)
(2) Flat tuple - based
(3) Flat stack - based
(4) or any combination of all above.

ASSIGNMENT 7 CODE :

//vif.y (if condition)

```
%{

#include<ctype.h>
#include<stdio.h>
#include<string.h>
int yylex(void);
void yyerror(char *);

extern char *yytext;
char st[100][10];
int top=0;
char i_[2]="0";
char temp[2]="t";

int label[20];
int lnum=0;
int ltop=0;



int push()
 {
  strcpy(st[++top],yytext);
 }

int codegen()
 {
 strcpy(temp,"t");
 strcat(temp,i_);
  printf("%s = %s %s %s\n",temp,st[top-2],st[top-1],st[top]);
  top-=2;
 strcpy(st[top],temp);
 i_[0]++;
 }

int codegen_umin()
 {
 strcpy(temp,"t");
 strcat(temp,i_);
 printf("%s = -%s\n",temp,st[top]);
 top--;
 strcpy(st[top],temp);
 i_[0]++;
 }
```

```
int codegen_assign()
 {
 printf("%s = %s\n",st[top-2],st[top]);
 top-=2;
 }


int lab1()
{
 lnum++;
 strcpy(temp,"t");
 strcat(temp,i_);
 printf("%s = not %s\n",temp,st[top]);
 printf("if %s goto L%d\n",temp,lnum);
 i_[0]++;
 label[++ltop]=lnum;
}

int lab2()
{

int x;
lnum++;
x=label[ltop--];
printf("goto L%d\n",lnum);
printf("L%d: \n",x);
label[++ltop]=lnum;
}

int lab3()
{
int y;
y=label[ltop--];
printf("L%d: \n",y);
}


void yyerror(char *s) {
    fprintf(stderr, "%s\n", s);

}



%}

%token ID NUM IF THEN ELSE
%right '='
```

```
%left '+' '-'
%left '*' '/'
%left UMINUS
%%

S : IF'('E')'{lab1();}THEN E';'{lab2();}ELSE E';'{lab3();}

E : E '+' {push();} E{codegen();}
  | E '-' {push();} E{codegen();}
  | E '*' {push();} E{codegen();}
  | E '/' {push();} E{codegen();}
  | '(' E ')'
  | '-'{push();} E{codegen_umin();} %prec UMINUS
  | V
  | NUM{push();}
  ;
V : ID {push();}
  ;
%%

int main()
 {
 printf("Enter the expression : ");
 yyparse();
 }
```

//vif.l

```
%{

  #include<stdio.h>
  #include "y.tab.h"
  void yyerror(char *);
  extern int yylval;


%}


ALPHA [A-Za-z]
DIGIT [0-9]
%%
if                  return IF;
```

```
then                    return THEN;
else                    return ELSE;
{ALPHA}({ALPHA}|{DIGIT})*    return ID;
{DIGIT}+                {yylval=atoi(yytext); return NUM;}
[ \t]                    ;
\n                  yyterminate();
.                   return yytext[0];
%%


int yywrap(void) {
    return 1;
}
```

//vwhile.y (while loop)

```
%{

#include<ctype.h>
#include<stdio.h>
#include<string.h>
int yylex(void);
void yyerror(char *);
char st[100][10];
int top=0;
char i_[2]="0";
char temp[2]="t";
int lnum=1;
int start=1;
extern char *yytext;


int push()
{
  strcpy(st[++top],yytext);
 }

 int codegen()
 {
 strcpy(temp,"t");
 strcat(temp,i_);
  printf("%s = %s %s %s\n",temp,st[top-2],st[top-1],st[top]);
  top-=2;
 strcpy(st[top],temp);
```

```
 i_[0]++;
 }

int codegen_umin()
 {
 strcpy(temp,"t");
 strcat(temp,i_);
 printf("%s = -%s\n",temp,st[top]);
 top--;
 strcpy(st[top],temp);
 i_[0]++;
 }

 int codegen_assign()
 {
 printf("%s = %s\n",st[top-2],st[top]);
 top-=2;
 }

 int lab1()
{
printf("L%d: \n",lnum++);
}

int lab2()
{
strcpy(temp,"t");
strcat(temp,i_);
printf("%s= not %s\n",temp,st[top]);
printf("if %s goto L %d \n",temp,lnum);
i_[0]++;
}

int lab3()
{
printf("goto L%d \n",start);
printf("L%d: \n",lnum);

}

 void yyerror(char *s)
 {
    fprintf(stderr, "%s\n", s);
}


%}
```

```
%token ID NUM WHILE
%right '='
%left '+' '-'
%left '*' '/'
%left UMINUS
%%

 S : WHILE{lab1();}'(' E ')'{lab2();}E';'{lab3();}
   ;
E : E '+'{push();} T{codegen();}
   | E '-'{push();} T{codegen();}
   | T
   ;
T : T '*'{push();} F{codegen();}
   | T '/'{push();} F{codegen();}
   | F
   ;
F : '(' E ')'
   | '-'{push();} F{codegen_umin();} %prec UMINUS
   | ID{push();}
   | NUM{push();}
   ;
%%



int main()
 {
 printf("Enter the expression : ");
 yyparse();
 }
```

//vwhile.l

```
%{

  #include<stdio.h>
  #include "y.tab.h"
  void yyerror(char *);
  extern int yylval;


%}

ALPHA [A-Za-z]
DIGIT [0-9]
```

```
%%
while  return WHILE;
{ALPHA}({ALPHA}|{DIGIT})* return ID;
{DIGIT}+ {yylval=atoi(yytext); return NUM;}
[\n\t] yyterminate();
. return yytext[0];

%%

int yywrap(void) {
    return 1;
}
```

//sw.y (switch case)

```
%{
#include <stdio.h>
#include <stdlib.h>
#include<ctype.h>
#include<string.h>

char stack[50][50];
extern char *yytext;
char temp[20];
int top=0;
int tempcount=0;
int lablecount=0;
int numb;
char number[10];




int generate(){
     printf("\nt%d = %s %s %s",tempcount,stack[top-3],stack[top-
2],stack[top-1]);
     top=top-3;
     strcpy(stack[top],"t");
     sprintf(temp,"%d",tempcount);
     strcat(stack[top],temp);
     top++;
     tempcount++;
}

int generate_minus(){
     printf("\nt%d = -%s",tempcount,stack[top]);
```

```
      top--;
      strcpy(stack[top],"t");
      sprintf(temp,"%d",tempcount);
      strcat(stack[top],temp);
      tempcount;
}
int generate_equal(){
      printf("\n%s = %s",stack[top-3],stack[top-1]);
      top=top-3;
}

int fun1(){
      printf("\nif %s is not equal to %s goto L%d",stack[top-
1],stack[top-2],lablecount++);
      top--;
}
int fun2(){
      printf("\ngoto L%d",lablecount);  //for break
      printf("\nL%d:",lablecount-1);         //for default
}

int fun3(){
      printf("\nL%d:",lablecount);
      exit(0);
}

int push(){
      strcpy(stack[top++],yytext);
}

int yyerror (char *msg){
      return fprintf (stderr, "YACC: %s\n", msg);
}

int yywrap(){
}



%}

%token ID NUM SWITCH CASE BREAK DEFAULT
%right '='
%left '+' '-'
%left '*' '/'
%left UMINUS

%%
```

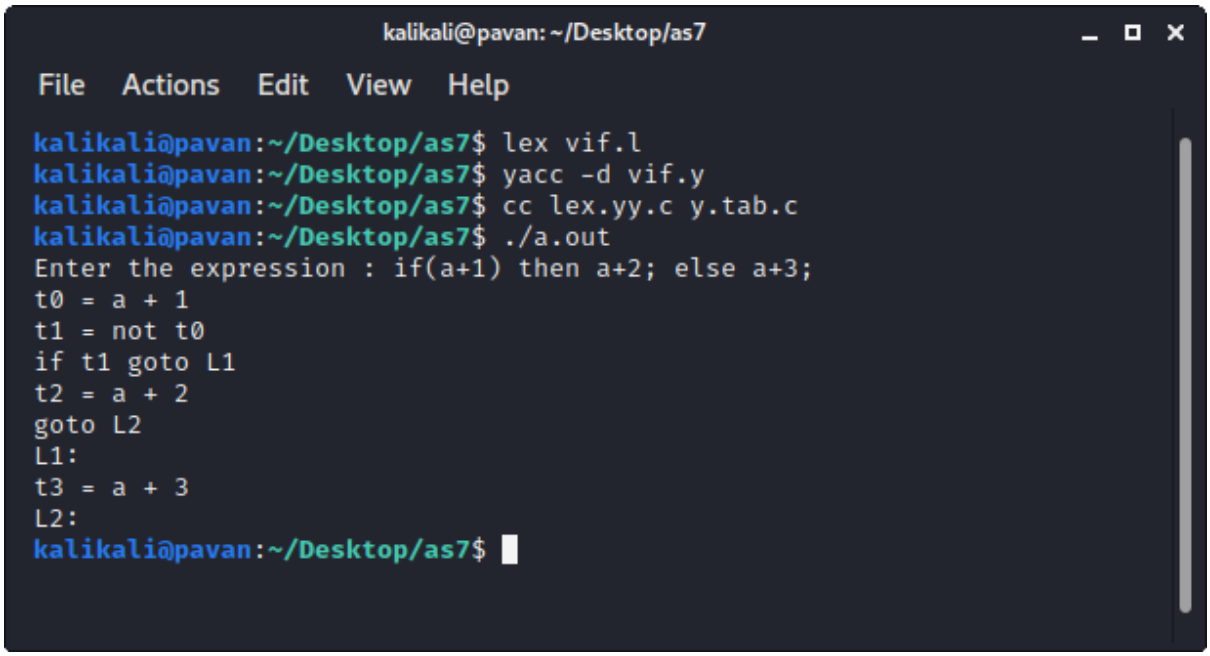
```
S     : SWITCH '('E')''{' CASE N {fun1();}':' E ';' BREAK
{fun2();}';'
                 DEFAULT ':' E {fun3();}';''}'
      ;
 E    : T '=' {push();} E{generate_equal();}
      | E '+' {push();} E{generate();}
      | E '-' {push();} E{generate();}
      | E '*' {push();} E{generate();}
      | E '/' {push();} E{generate();}
      | '(' E ')'
      | '-'{push();} E{generate_minus();} %prec UMINUS
      | T
      | NUM{numb=$1;push();}
      ;
T     : ID {push(yytext);}
      ;
N     :NUM{numb=$1;push();}
      ;
%%

int main(){
      printf("\nEnter the expression: ");
      yyparse();
}
```

//sw.l

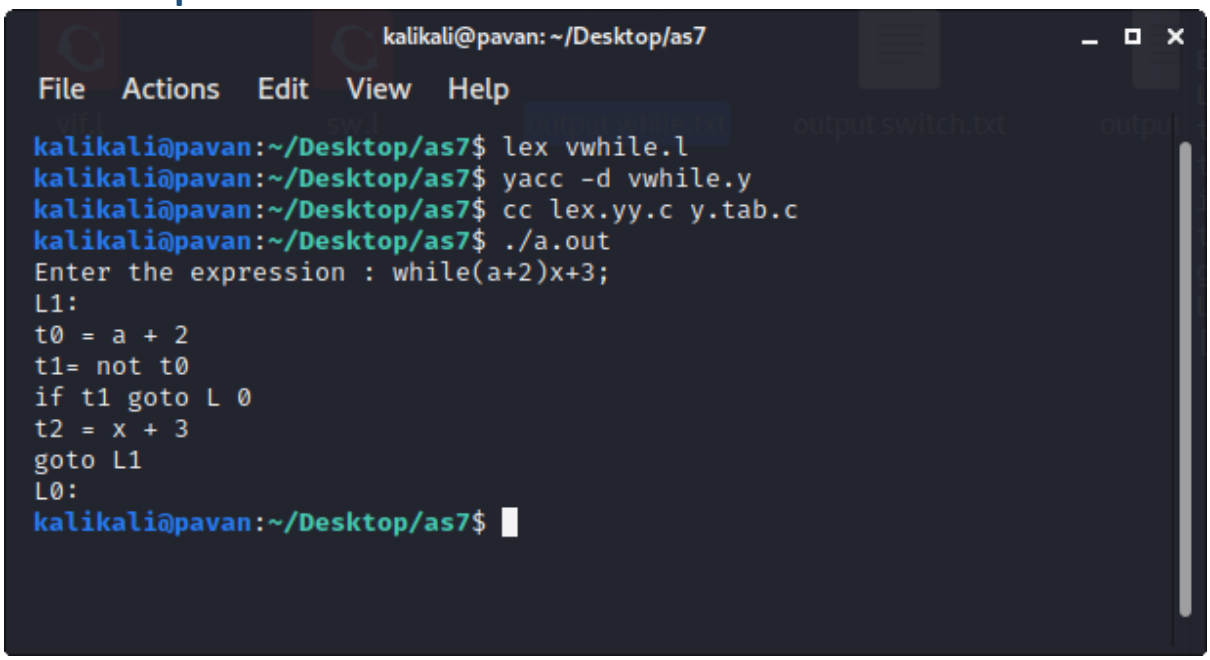
```
%{
#include "y.tab.h"
extern int yylval;
%}
VAR [A-Za-z]
DIGIT [0-9]
%%
[ \t\n]
switch                {return SWITCH;}
case                  {return CASE;}
break                 {return BREAK;}
default               {return DEFAULT;}
{VAR}({VAR}|{DIGIT})* {return ID;}
{DIGIT}+              {return NUM;}
.                     {return yytext[0];}
%%
```

OUTPUTS :

If –



While loop -



Switch case –


kalikali@pavan: ~/Desktop/as7
File Actions Edit View Help
kalikali@pavan:~/Desktop/as7$ lex sw.l
kalikali@pavan:~/Desktop/as7$ yacc -d sw.y
kalikali@pavan:~/Desktop/as7$ cc lex.yy.c y.tab.c
y.tab.c: In function 'yyparse':
y.tab.c:1153:16: warning: implicit declaration of function 'yylex' [-Wimplicit-function-declaration]
1153 | yychar = yylex ();
^~~~~
kalikali@pavan:~/Desktop/as7$ ./a.out
Enter the expression: switch (b+1) { case 1:x=y+z ; break; default: p=q+r;}
t0 = b + 1
if 1 is not equal to t0 goto L0
t1 = y + z
x = t1
goto L1
L0:
t2 = q + r
p = t2
kalikali@pavan:~/Desktop/as7$