COMPUTER DEPARTMENT

Subject :- SPCC Class :- TE Sem :- VI

Experiment Number :- 1

Title: Develop a C Program for the Implementation of a Symbol Table with functions to CREATE, INSERT, MODIFY, SEARCH and DISPLAY.

Tools / Softwares used :- Turbo C / Dev C++

Programming Language :- C

Experiment / Program :-

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

int size=0;
void Insert();
void Display();
void Delete();
int Search(char lab[]);
void Modify();
struct SymbTab
{
 char label[10],symbol[10];
 int addr;
struct SymbTab *next;};
struct SymbTab *first,*last;
void main()
{
 int op,y;
 char la[10];

 do
 {
  printf("\n\tSYMBOL TABLE IMPLEMENTATION\n");

printf("\n\t1.INSERT\n\t2.DISPLAY\n\t3.DELETE\n\t4.SEARCH\n\t5.MODIFY\n\t6.END
\n");
  printf("\n\tEnter your option : ");
  scanf("%d",&op);
  switch(op)
  {
   case 1:
      Insert();
      break;
   case 2:
      Display();
      break;
   case 3:
      Delete();
      break;
   case 4:
      printf("\n\tEnter the label to be searched : ");
      scanf("%s",la);
      y=Search(la);
      printf("\n\tSearch Result:");
      if(y==1)
    printf("\n\tThe label is present in the symbol table\n");
      else
    printf("\n\tThe label is not present in the symbol table\n");
      break;
   case 5:
      Modify();
      break;
   case 6:
      exit(0);
  }
 }while(op<6);
 getch();
}
void Insert()
{
  int n;
  char l[10];
  printf("\n\tEnter the label : ");
  scanf("%s",l);
  n=Search(l);
  if(n==1)
   printf("\n\tThe label exists already in the symbol table\n\tDuplicate can't
be inserted");
  else
   {
    struct SymbTab *p;
    p=malloc(sizeof(struct SymbTab));
    strcpy(p->label,l);
    printf("\n\tEnter the symbol : ");
    scanf("%s",p->symbol);
    printf("\n\tEnter the address : ");
    scanf("%d",&p->addr);
    p->next=NULL;
    if(size==0)
     {
      first=p;
      last=p;
     }
    else
     {
      last->next=p;
      last=p;
      }
     size++;
    }
   printf("\n\tLabel inserted\n");
}
void Display()
{
  int i;
  struct SymbTab *p;
  p=first;
  printf("\n\tLABEL\t\tSYMBOL\t\tADDRESS\n");
  for(i=0;i<size;i++)
   {
    printf("\t%s\t\t%s\t\t%d\n",p->label,p->symbol,p->addr);
    p=p->next;
   }
}
int Search(char lab[])
{
 int i,flag=0;
 struct SymbTab *p;
 p=first;
  for(i=0;i<size;i++)
   {
    if(strcmp(p->label,lab)==0)
     flag=1;
    p=p->next;
   }
 return flag;
}
void Modify()
{
  char l[10],nl[10];
  int add,choice,i,s;
  struct SymbTab *p;
  p=first;
  printf("\n\tWhat do you want to modify?\n");
  printf("\n\t1.Only the label\n\t2.Only the address\n\t3.Both the label and
address\n");
  printf("\tEnter your choice : ");
  scanf("%d",&choice);
  switch(choice)
   {
     case 1:
       printf("\n\tEnter the old label : ");
       scanf("%s",l);
       s=Search(l);
       if(s==0)
     printf("\n\tLabel not found\n");
       else
```

Experiment Number :- 1

```
{
 printf("\n\tEnter the new label : ");
 scanf("%s",nl);
 for(i=0;i<size;i++)
  {
   if(strcmp(p->label,l)==0)
     strcpy(p->label,nl);
   p=p->next;
  }
 printf("\n\tAfter Modification:\n");
 Display();
}
break;
case 2:
   printf("\n\tEnter the label where the address is to be modified : ");
   scanf("%s",l);
   s=Search(l);
   if(s==0)
 printf("\n\tLabel not found\n");
   else
{
 printf("\n\tEnter the new address : ");
 scanf("%d",&add);
 for(i=0;i<size;i++)
  {
   if(strcmp(p->label,l)==0)
    p->addr=add;
   p=p->next;
  }
 printf("\n\tAfter Modification:\n");
 Display();
}
break;
case 3:
   printf("\n\tEnter the old label : ");
   scanf("%s",l);
   s=Search(l);
   if(s==0)
 printf("\n\tLabel not found\n");
   else
{
 printf("\n\tEnter the new label : ");
 scanf("%s",nl);
 printf("\n\tEnter the new address : ");
 scanf("%d",&add);
 for(i=0;i<size;i++)
  {
   if(strcmp(p->label,l)==0)
    {
     strcpy(p->label,nl);
```

Experiment Number :- 1

```
        p->addr=add;
       }
      p=p->next;
     }
    printf("\n\tAfter Modification:\n");
    Display();
   }
   break;
   }
}
void Delete()
{
  int a;
  char l[10];
  struct SymbTab *p,*q;
  p=first;
  printf("\n\tEnter the label to be deleted : ");
  scanf("%s",l);
  a=Search(l);
  if(a==0)
    printf("\n\tLabel not found\n");
  else
   {
    if(strcmp(first->label,l)==0)
    first=first->next;
    else if(strcmp(last->label,l)==0)
     {
      q=p->next;
      while(strcmp(q->label,l)!=0)
      {
       p=p->next;
       q=q->next;
      }
      p->next=NULL;
      last=p;
     }
    else
     {
      q=p->next;
      while(strcmp(q->label,l)!=0)
      {
       p=p->next;
       q=q->next;
      }
      p->next=q->next;
    }
    size--;
    printf("\n\tAfter Deletion:\n");
    Display();
   }
```

```
}
```

Output :-

```
 SYMBOL TABLE IMPLEMENTATION

1.INSERT
2.DISPLAY
3.DELETE
4.SEARCH
5.MODIFY
6.END

Enter your option : 1

Enter the label : z

Enter the symbol : add

Enter the address : 1234

Label inserted

SYMBOL TABLE IMPLEMENTATION

1.INSERT
2.DISPLAY
3.DELETE
4.SEARCH
5.MODIFY
6.END

Enter your option : 2

LABEL       SYMBOL      ADDRESS
z           add         1234

SYMBOL TABLE IMPLEMENTATION

1.INSERT
2.DISPLAY
3.DELETE
4.SEARCH
5.MODIFY
6.END

Enter your option : 4

Enter the label to be searched : z

Search Result:
```

```
The label is present in the symbol table

SYMBOL TABLE IMPLEMENTATION

1.INSERT
2.DISPLAY
3.DELETE
4.SEARCH
5.MODIFY
6.END

Enter your option : 5

What do you want to modify?

1.Only the label
2.Only the address
3.Both the label and address
Enter your choice : 3

Enter the old label : z

Enter the new label : x

Enter the new address : 8765

After Modification:

LABEL         SYMBOL        ADDRESS
x             add           8765

SYMBOL TABLE IMPLEMENTATION

1.INSERT
2.DISPLAY
3.DELETE
4.SEARCH
5.MODIFY
6.END

Enter your option : 2

LABEL         SYMBOL        ADDRESS
x             add           8765

SYMBOL TABLE IMPLEMENTATION

1.INSERT
2.DISPLAY
3.DELETE
4.SEARCH
5.MODIFY
6.END
```

```
Enter your option : 3

Enter the label to be deleted : x

After Deletion:

LABEL          SYMBOL         ADDRESS

SYMBOL TABLE IMPLEMENTATION

1.INSERT
2.DISPLAY
3.DELETE
4.SEARCH
5.MODIFY
6.END

Enter your option : 2

LABEL          SYMBOL         ADDRESS

SYMBOL TABLE IMPLEMENTATION

1.INSERT
2.DISPLAY
3.DELETE
4.SEARCH
5.MODIFY
6.END
Enter your option : 6

Enter your option :
```

Conclusion :-

We have Developed a C Program which Implements a Symbol Table with functions to CREATE, INSERT, MODIFY, SEARCH and DISPLAY.

Experiment Number :- 2

Title: Develop a C Program for the Implementation of Pass One of a Two Pass Assembler.

Tools / Softwares used :- Turbo C / Dev C++

Programming Language :- C

Experiment / Program :-
Pass1.c

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>
int main()
{
 char opcode[10],operand[10],label[10],code[10][10],ch; char
mnemonic[10][10]={"START","LDA","STA","LDCH","STCH","END"};
 int locctr,start,len,i=0,j=0;
 FILE *fp1,*fp2,*fp3;
 //clrscr();
 fp1=fopen("INPUT.DAT","r");
 fp2=fopen("SYMTAB.DAT","w");
 fp3=fopen("OUT.DAT","w");
 fscanf(fp1,"%s%s%s",label,opcode,operand);
 //fprintf(fp3,"%s\t%s\t%s",label,opcode,operand);
 if(strcmp(opcode,"START")==0)
  {
   start=atoi(operand);
   locctr=start;
   fprintf(fp3,"%s\t%s\t%s\n",label,opcode,operand);
   fscanf(fp1,"%s%s%s",label,opcode,operand);
  }
 else
  locctr=0;
 while(strcmp(opcode,"END")!=0)
  {
   fprintf(fp3,"%d",locctr);
   if(strcmp(label,"**")!=0)
     fprintf(fp2,"%s\t%d\n",label,locctr);
   strcpy(code[i],mnemonic[j]);
   while(strcmp(mnemonic[j],"END")!=0)
    {
     if(strcmp(opcode,mnemonic[j])==0)
     {
      locctr+=3;
      break;
     }
     strcpy(code[i],mnemonic[j]);
```

Experiment Number :- 2

```
     j++;
    }
   if(strcmp(opcode,"WORD")==0)
    locctr+=3;
   else if(strcmp(opcode,"RESW")==0)
    locctr+=(3*(atoi(operand)));
   else if(strcmp(opcode,"RESB")==0)
    locctr+=(atoi(operand));
   else if(strcmp(opcode,"BYTE")==0)
    ++locctr;
   fprintf(fp3,"\t%s\t%s\t%s\n",label,opcode,operand);
   fscanf(fp1,"%s%s%s",label,opcode,operand);
  }
  fprintf(fp3,"%d\t%s\t%s\t%s\n",locctr,label,opcode,operand);

  fclose(fp1);
  fclose(fp2);
  fclose(fp3);
  printf("\n\nThe contents of Input Table :\n\n");
  fp1=fopen("INPUT.DAT","r");
  ch=fgetc(fp1);
  while(ch!=EOF)
   {
    printf("%c",ch);
    ch=fgetc(fp1);
   }
  printf("\n\nThe contents of Output Table :\n\n\t");
  fp3=fopen("OUT.DAT","r");
  ch=fgetc(fp3);
  while(ch!=EOF)
   {
    printf("%c",ch);
    ch=fgetc(fp3);
   }
  len=locctr-start;
  printf("\nThe length of the program is %d.\n\n",len);
  printf("\n\nThe contents of Symbol Table :\n\n");
  fp2=fopen("SYMTAB.DAT","r");
  ch=fgetc(fp2);
  while(ch!=EOF)
   {
    printf("%c",ch);
    ch=fgetc(fp2);
   }
  fclose(fp1);
  fclose(fp2);
  fclose(fp3);
  getch();
}
```

Experiment Number :- 2

INPUT.DAT file

```
** START 2000
** LDA FIVE
** STA ALPHA
** LDCH CHARZ
** STCH C1
ALPHA RESW 1
FIVE WORD 5
CHARZ BYTE C'Z'
C1 RESB 1
** END **
```

Output :-

SYMTAB.DAT

```
ALPHA 2012
FIVE  2015
CHARZ 2018
C1    2019
```

OUT.DAT

```
**    START 2000
2000  **    LDA   FIVE
2003  **    STA   ALPHA
2006  **    LDCH  CHARZ
2009  **    STCH  C1
2012  ALPHA RESW  1
2015  FIVE  WORD  5
2018  CHARZ BYTE  C'Z'
2019  C1    RESB  1
2020  **    END   **
```

Conclusion :-
We have Developed a C Program which Implements Pass One of a Two Pass Assembler.

Experiment Number :- 3

Title: Develop a C Program for the Implementation of Pass Two of a Two Pass Assembler.

Tools / Softwares used :- Turbo C / Dev C++

Programming Language :- C

Experiment / Program :-
Pass2.c

```
include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>
int main()
{
  char a[10],ad[10],label[10],opcode[10],operand[10],symbol[10],ch;  int
st,diff,i,address,add,len,actual_len,finaddr,prevaddr,j=0;
  char mnemonic[15][15]={"LDA","STA","LDCH","STCH"};
  char code[15][15]={"33","44","53","57"};
  FILE *fp1,*fp2,*fp3,*fp4;

  fp1=fopen("ASSMLIST.DAT","w");
  fp2=fopen("SYMTAB.DAT","r");
  fp3=fopen("INTERMED.DAT","r");
  fp4=fopen("OBJCODE.DAT","w");
  fscanf(fp3,"%s%s%s",label,opcode,operand);

  while(strcmp(opcode,"END")!=0)
  {
   prevaddr=address;
   fscanf(fp3,"%d%s%s%s",&address,label,opcode,operand);
  }
  finaddr=address;
  fclose(fp3);
  fp3=fopen("INTERMED.DAT","r");

  fscanf(fp3,"%s%s%s",label,opcode,operand);
  if(strcmp(opcode,"START")==0)
  {
   fprintf(fp1,"\t%s\t%s\t%s\n",label,opcode,operand);
   fprintf(fp4,"H^%s^00%s^00%d\n",label,operand,finaddr);
   fscanf(fp3,"%d%s%s%s",&address,label,opcode,operand);
   st=address;
   diff=prevaddr-st;
   fprintf(fp4,"T^00%d^%d",address,diff);
  }
  while(strcmp(opcode,"END")!=0)
  {
```

Experiment Number :- 3

```
    if(strcmp(opcode,"BYTE")==0)
    {
     fprintf(fp1,"%d\t%s\t%s\t%s\t",address,label,opcode,operand);
     len=strlen(operand);
     actual_len=len-3;
     fprintf(fp4,"^");
     for(i=2;i<(actual_len+2);i++)
     {
      itoa(operand[i],ad,16);
      fprintf(fp1,"%s",ad);
      fprintf(fp4,"%s",ad);
     }
     fprintf(fp1,"\n");
    }
    else if(strcmp(opcode,"WORD")==0)
    {
     len=strlen(operand);
     itoa(atoi(operand),a,10);
     fprintf(fp1,"%d\t%s\t%s\t%s\t00000%s\n",address,label,opcode,operand,a);
     fprintf(fp4,"^00000%s",a);
    }
    else if((strcmp(opcode,"RESB")==0)||(strcmp(opcode,"RESW")==0))
     fprintf(fp1,"%d\t%s\t%s\t%s\n",address,label,opcode,operand);
    else
    {
     while(strcmp(opcode,mnemonic[j])!=0)
      j++;
     if(strcmp(operand,"COPY")==0)

fprintf(fp1,"%d\t%s\t%s\t%s\t%s0000\n",address,label,opcode,operand,code[j]);
     else
     {
      rewind(fp2);
      fscanf(fp2,"%s%d",symbol,&add);
       while(strcmp(operand,symbol)!=0)
        fscanf(fp2,"%s%d",symbol,&add);

fprintf(fp1,"%d\t%s\t%s\t%s\t%s%d\n",address,label,opcode,operand,code[j],add)
;
      fprintf(fp4,"^%s%d",code[j],add);
     }
    }
    fscanf(fp3,"%d%s%s%s",&address,label,opcode,operand);
   }
   fprintf(fp1,"%d\t%s\t%s\t%s\n",address,label,opcode,operand);
   fprintf(fp4,"\nE^00%d",st);
   printf("\n Intermediate file is converted into object code");

   fclose(fp1);
   fclose(fp2);
```

```
  fclose(fp3);
  fclose(fp4);

  printf("\n\nThe contents of Intermediate file:\n\n\t");
  fp3=fopen("INTERMED.DAT","r");
  ch=fgetc(fp3);
  while(ch!=EOF)
  {
   printf("%c",ch);
   ch=fgetc(fp3);
  }
  printf("\n\nThe contents of Symbol Table :\n\n");
  fp2=fopen("SYMTAB.DAT","r");
  ch=fgetc(fp2);
  while(ch!=EOF)
  {
   printf("%c",ch);
   ch=fgetc(fp2);
  }
  printf("\n\nThe contents of Output file :\n\n");
  fp1=fopen("ASSMLIST.DAT","r");
  ch=fgetc(fp1);
  while(ch!=EOF)
  {
   printf("%c",ch);
   ch=fgetc(fp1);
  }
  printf("\n\nThe contents of Object code file :\n\n");
  fp4=fopen("OBJCODE.DAT","r");
  ch=fgetc(fp4);
  while(ch!=EOF)
  {
   printf("%c",ch);
   ch=fgetc(fp4);
  }
  fclose(fp1);
  fclose(fp2);
  fclose(fp3);
  fclose(fp4);

  getch();
}
```

SYMTAB.DAT file

```
ALPHA    2012
FIVE    2015
CHARZ    2018
```

```
C1    2019
```

INTERMED.DAT file

```
COPY    START    2000
2000    **    LDA    FIVE
2003    **    STA    ALPHA
2006    **    LDCH    CHARZ
2009    **    STCH    C1
2012    ALPHA    RESW    1
2015    FIVE    WORD    5
2018    CHARZ    BYTE    C'EOF'
2019    C1    RESB    1
2020    **    END    **
```

Output :-

ASSMLIST.DAT

```
     COPY  START 2000
2000 **    LDA   FIVE  332015
2003 **    STA   ALPHA 442012
2006 **    LDCH  CHARZ 532018
2009 **    STCH  C1    572019
2012 ALPHA RESW  1
2015 FIVE  WORD  5     000005
2018 CHARZ BYTE  C'EOF'     454f46
2019 C1    RESB  1
2020 **    END   **
```

OBJCODE.DAT

```
H^COPY^002000^002020
T^002000^19^332015^442012^532018^572019^000005^454f46
E^002000
```

Conclusion :-
We have Developed a C Program which Implements Pass Two of a Two Pass Assembler.

Experiment Number :- 4

Title: Develop a java Program for the Implementation of Pass One of a Two Pass Macro Processor.

Tools / Softwares used :- jdk 1.8, Netbeans/Eclipse

Programming Language :- Java

Experiment / Program :-

Input
macro_input.asm

```
MACRO
M1      &X, &Y, &A=AREG, &B=
MOVER   &A, &X
ADD     &A, ='1'
MOVER   &B, &Y
ADD     &B, ='5'
MEND
MACRO
M2      &P, &Q, &U=CREG, &V=DREG
MOVER   &U, &P
MOVER   &V, &Q
ADD     &U, ='15'
ADD     &V, ='10'
MEND
START   100
M1      10, 20, &B=CREG
M2      100, 200, &V=AREG, &U=BREG
END
```

MacroP1.java

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Iterator;
import java.util.LinkedHashMap;

public class MacroP1 {

        public static void main(String[] args) throws IOException{
                BufferedReader br=new BufferedReader(new
FileReader("macro_input.asm"));

                FileWriter mnt=new FileWriter("mnt.txt");
                FileWriter mdt=new FileWriter("mdt.txt");
                FileWriter kpdt=new FileWriter("kpdt.txt");
                FileWriter pnt=new FileWriter("pntab.txt");
                FileWriter ir=new FileWriter("intermediate.txt");
                LinkedHashMap<String, Integer> pntab=new LinkedHashMap<>();
                String line;
```

Experiment Number :- 4

```
		String Macroname = null;
		int mdtp=1,kpdtp=0,paramNo=1,pp=0,kp=0,flag=0;
		while((line=br.readLine())!=null)
		{

			String parts[]=line.split("\\s+");
			if(parts[0].equalsIgnoreCase("MACRO"))
			{
				flag=1;
				line=br.readLine();
				parts=line.split("\\s+");
				Macroname=parts[0];
				if(parts.length<=1)
				{

	mnt.write(parts[0]+"\t"+pp+"\t"+kp+"\t"+mdtp+"\t"+(kp==0?kpdtp:(kpdtp+1))+"\n"
);
					continue;
				}
				for(int i=1;i<parts.length;i++) //processing of 
parameters
				{
					parts[i]=parts[i].replaceAll("[&,]", "");
					//System.out.println(parts[i]);
					if(parts[i].contains("="))
					{
						++kp;
						String 
keywordParam[]=parts[i].split("=");
						pntab.put(keywordParam[0], paramNo++);
						if(keywordParam.length==2)
						{

	kpdt.write(keywordParam[0]+"\t"+keywordParam[1]+"\n");
						}
						else
						{
							kpdt.write(keywordParam[0]+"\t-
\n");
						}
					}
					else
					{
						pntab.put(parts[i], paramNo++);
						pp++;
					}
				}

	mnt.write(parts[0]+"\t"+pp+"\t"+kp+"\t"+mdtp+"\t"+(kp==0?kpdtp:(kpdtp+1))+"\n"
);
				kpdtp=kpdtp+kp;
				//System.out.println("KP="+kp);


			}
```

```
                        else if(parts[0].equalsIgnoreCase("MEND"))
                        {
                                mdt.write(line+"\n");
                                flag=kp=pp=0;
                                mdtp++;
                                paramNo=1;
                                pnt.write(Macroname+":\t");
                                Iterator<String> itr=pntab.keySet().iterator();
                                while(itr.hasNext())
                                {
                                        pnt.write(itr.next()+"\t");
                                }
                                pnt.write("\n");
                                pntab.clear();
                        }
                        else if(flag==1)
                        {
                                for(int i=0;i<parts.length;i++)
                                {
                                        if(parts[i].contains("&"))
                                        {
                                                parts[i]=parts[i].replaceAll("[&,]",
"");

        mdt.write("(P,"+pntab.get(parts[i])+")\t");
                                        }
                                        else
                                        {
                                                mdt.write(parts[i]+"\t");
                                        }
                                }
                                mdt.write("\n");
                                mdtp++;
                        }
                        else
                        {
                                ir.write(line+"\n");
                        }
                }
                br.close();
                mdt.close();
                mnt.close();
                ir.close();
                pnt.close();
                kpdt.close();
                System.out.println("MAcro PAss1 Processing done. :)");
        }

}
```

Output :-

Experiment Number :- 4

intermediate.txt

```
START   100
M1      10, 20, &B=CREG
M2      100, 200, &V=AREG, &U=BREG
END
```

kpdt.txt

```
A       AREG
B       -
U       CREG
V       DREG
```

mdt.txt

```
MOVER   (P,3)   (P,1)
ADD     (P,3)   ='1'
MOVER   (P,4)   (P,2)
ADD     (P,4)   ='5'
MEND
MOVER   (P,3)   (P,1)
MOVER   (P,4)   (P,2)
ADD     (P,3)   ='15'
ADD     (P,4)   ='10'
MEND
```

mnt.txt

```
M1      2       2       1       1
M2      2       2       6       3
```

pntab.txt

```
M1:     X       Y       A       B
M2:     P       Q       U       V
```

Conclusion :-

We have Developed a java Program that Implements Pass One of a Two Pass Macro Processor.

Experiment Number :- 5

Title: Develop a java Program for the Implementation of Pass Two of a Two Pass Macro Processor.

Tools / Softwares used :- jdk 1.8, Netbeans/Eclipse

Programming Language :- Java

Experiment / Program :-

Input

intermediate.txt

```
START   100
M1      10, 20, &B=CREG
M2      100, 200, &V=AREG, &U=BREG
END
```

mdt.txt

```
MOVER   (P,3)   (P,1)
ADD     (P,3)   ='1'
MOVER   (P,4)   (P,2)
ADD     (P,4)   ='5'
MEND
MOVER   (P,3)   (P,1)
MOVER   (P,4)   (P,2)
ADD     (P,3)   ='15'
ADD     (P,4)   ='10'
MEND
```

mnt.txt

```
M1      2       2       1       1
M2      2       2       6       3
```

MacroP2.java

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.util.HashMap;
import java.util.Vector;

public class MacroP2 {

        public static void main(String[] args) throws Exception {
                BufferedReader irb=new BufferedReader(new
FileReader("intermediate.txt"));
                BufferedReader mdtb=new BufferedReader(new FileReader("mdt.txt"));
                BufferedReader kpdtb=new BufferedReader(new FileReader("kpdt.txt"));
                BufferedReader mntb=new BufferedReader(new FileReader("mnt.txt"));

                FileWriter fr=new FileWriter("pass2.txt");

                HashMap<String, MNTEntry> mnt=new HashMap<>();
                HashMap<Integer, String> aptab=new HashMap<>();
```

```
		HashMap<String,Integer> aptabInverse=new HashMap<>();

		Vector<String>mdt=new Vector<String>();
		Vector<String>kpdt=new Vector<String>();

		int pp,kp,mdtp,kpdtp,paramNo;
		String line;
		while((line=mdtb.readLine())!=null)
		{
			mdt.addElement(line);
		}
		while((line=kpdtb.readLine())!=null)
		{
			kpdt.addElement(line);
		}
		while((line=mntb.readLine())!=null)
		{
			String parts[]=line.split("\\s+");
			mnt.put(parts[0], new MNTEntry(parts[0],
Integer.parseInt(parts[1]), Integer.parseInt(parts[2]), Integer.parseInt(parts[3]),
Integer.parseInt(parts[4])));

		}

		while((line=irb.readLine())!=null)
		{
			String []parts=line.split("\\s+");
			if(mnt.containsKey(parts[0]))
			{
				pp=mnt.get(parts[0]).getPp();
				kp=mnt.get(parts[0]).getKp();
				kpdtp=mnt.get(parts[0]).getKpdtp();
				mdtp=mnt.get(parts[0]).getMdtp();
				paramNo=1;
				for(int i=0;i<pp;i++)
				{
					parts[paramNo]=parts[paramNo].replace(",", "");
					aptab.put(paramNo, parts[paramNo]);
					aptabInverse.put(parts[paramNo], paramNo);
					paramNo++;
				}
				int j=kpdtp-1;
				for(int i=0;i<kp;i++)
				{
					String temp[]=kpdt.get(j).split("\t");
					aptab.put(paramNo,temp[1]);
					aptabInverse.put(temp[0],paramNo);
					j++;
					paramNo++;
				}

				for(int i=pp+1;i<parts.length;i++)
				{
					parts[i]=parts[i].replace(",", "");
					String splits[]=parts[i].split("=");
```

```
                                        String name=splits[0].replaceAll("&", "");
                                        aptab.put(aptabInverse.get(name),splits[1]);
                                }
                                int i=mdtp-1;
                                while(!mdt.get(i).equalsIgnoreCase("MEND"))
                                {
                                        String splits[]=mdt.get(i).split("\\s+");
                                        fr.write("+");
                                        for(int k=0;k<splits.length;k++)
                                        {
                                                if(splits[k].contains("(P,"))
                                                {

        splits[k]=splits[k].replaceAll("[^0-9]", "");//not containing number
                                                        String
value=aptab.get(Integer.parseInt(splits[k]));
                                                        fr.write(value+"\t");
                                                }
                                                else
                                                {
                                                        fr.write(splits[k]+"\t");
                                                }
                                        }
                                        fr.write("\n");
                                        i++;
                                }

                                aptab.clear();
                                aptabInverse.clear();
                        }
                        else
                        {
                                fr.write(line+"\n");
                        }

        }

        fr.close();
        mntb.close();
        mdtb.close();
        kpdtb.close();
        irb.close();
        }
}
```

MNTEntry.java

```
public class MNTEntry {
String name;
int pp,kp,mdtp,kpdtp;


public MNTEntry(String name, int pp, int kp, int mdtp, int kpdtp) {
        super();
        this.name = name;
        this.pp = pp;
        this.kp = kp;
        this.mdtp = mdtp;
        this.kpdtp = kpdtp;
}
public String getName() {
        return name;
}
public void setName(String name) {
        this.name = name;
}
public int getPp() {
        return pp;
}
public void setPp(int pp) {
        this.pp = pp;
}
public int getKp() {
        return kp;
}
public void setKp(int kp) {
        this.kp = kp;
}
public int getMdtp() {
        return mdtp;
}
public void setMdtp(int mdtp) {
        this.mdtp = mdtp;
}
public int getKpdtp() {
        return kpdtp;
}
public void setKpdtp(int kpdtp) {
        this.kpdtp = kpdtp;
}

}
```

Output :-

pass2.asm

```
START   100
+MOVER  AREG    10
+ADD    AREG    ='1'
+MOVER  CREG    20
+ADD    CREG    ='5'
+MOVER  BREG    100
+MOVER  AREG    200
+ADD    BREG    ='15'
+ADD    AREG    ='10'
END
```

Conclusion :-
We have Developed a java Program that Implements Pass Two of a Two Pass Macro Processor.

Experiment Number :- 6

Title: Study of Lex - A Lexical Analyzer Generator.

Tools / Softwares used :- bison-2.4.1-src-setup.exe , flex-2.5.4a-1.exe , Dev C++

Programming Language :- lex

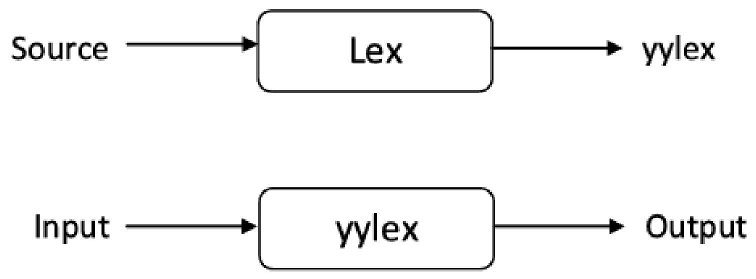
Introduction
Lex is a program generator designed for lexical processing of character input streams. It accepts a high-level, problem oriented specification for character string matching, and produces a program in a general purpose language which recognizes regular expressions. The regular expressions are specified by the user in the source specifications given to Lex. The Lex written code recognizes these expressions in an input stream and partitions the input stream into strings matching the expressions. At the boundaries between strings program sections provided by the user are executed. The Lex source file associates the regular expressions and the program fragments. As each expression appears in the input to the program written by Lex, the corresponding fragment is executed.

The user supplies the additional code beyond expression matching needed to complete his tasks, possibly including code written by other generators. The program that recognizes the expressions is generated in the general purpose programming language employed for the user's program fragments. Thus, a high level expression language is provided to write the string expressions to be matched while the user's freedom to write actions is unimpaired. This avoids forcing the user who wishes to use a string manipulation language for input analysis to write processing programs in the same and often inappropriate string handling language.

Lex is not a complete language, but rather a generator representing a new language feature which can be added to different programming languages, called ``host languages.'' Just as general purpose languages can produce code to run on different computer hardware, Lex can write code in different host languages. The host language is used for the output code generated by Lex and also for the program fragments added by the user. Compatible run-time libraries for the different host languages are also provided. This makes Lex adaptable to different environments and different users. Each application may be directed to the combination of hardware and host language appropriate to the task, the user's background, and the properties of local implementations. At present, the only supported host language is C, although Fortran (in the form of Ratfor [2] has been available in the past. Lex itself exists on UNIX, GCOS, and OS/370; but the code generated by Lex may be taken anywhere the appropriate compilers exist.

Lex turns the user's expressions and actions (called source in this memo) into the host general- purpose language; the generated program is named yylex. The yylex program will recognize expressions in a stream (called input in this memo) and perform the specified actions for each expression as it is detected. See below Figure.

Experiment Number :- 6



For a trivial example, consider a program to delete from the input all blanks or tabs at the ends of lines.

```
%%

[ \t]+$ ;
```

is all that is required. The program contains a %% delimiter to mark the beginning of the rules, and one rule. This rule contains a regular expression which matches one or more instances of the characters blank or tab (written \t for visibility, in accordance with the C language convention) just prior to the end of a line. The brackets indicate the character class made of blank and tab; the + indicates ``one or more ...''; and the $ indicates ``end of line,'' as in QED. No action is specified, so the program generated by Lex (yylex) will ignore these characters. Everything else will be copied. To change any remaining string of blanks or tabs to a single blank, add another rule:

```
%%

[ \t]+$ ;

[ \t]+      printf(" ");
```

The finite automaton generated for this source will scan for both rules at once, observing at the termination of the string of blanks or tabs whether or not there is a newline character, and executing the desired rule action. The first rule matches all strings of blanks or tabs at the end of lines, and the second rule all remaining strings of blanks or tabs.
Lex Source Definitions
The general format of Lex source is:

```
{definitions}
%%
{rules}
%%
{user subroutines}
```

where the definitions and the user subroutines are often omitted. The second %% is optional, but the first is required to mark the beginning of the rules. The absolute minimum Lex program is thus

```
%%
```

(no definitions, no rules) which translates into a program which copies the input to the output unchanged.

Remember that Lex is turning the rules into a program. Any source not intercepted by Lex is copied into the generated program. There are three classes of such things.

1) Any line which is not part of a Lex rule or action which begins with a blank or tab is copied into the Lex generated program. Such source input prior to the first %% delimiter will be external to any function in the code; if it appears immediately after the first %%, it appears in an appropriate place for declarations in the function written by Lex which contains the actions. This material must look like program fragments, and should precede the first Lex rule. As a side effect of the above, lines which begin with a blank or tab, and which contain a comment, are passed through to the generated program. This can be used to include comments in either the Lex source or the generated code. The comments should follow the host language convention.

2) Anything included between lines containing only %{ and %} is copied out as above. The delimiters are discarded. This format permits entering text like preprocessor statements that must begin in column 1, or copying lines that do not look like programs.

3) Anything after the third %% delimiter, regardless of formats, etc., is copied out after the Lex output.

Definitions intended for Lex are given before the first %% delimiter. Any line in this section not contained between %{ and %}, and begining in column 1, is assumed to define Lex substitution strings. The format of such lines is name translation and it causes the string given as a translation to be associated with the name. The name and translation must be separated by at least one blank or tab, and the name must begin with a letter. The translation can then be called out by the {name} syntax in a rule. Using {D} for the digits and {E} for an exponent field, for example, might abbreviate rules to recognize numbers:

```
D       [0-9]
E       [DEde][-+]?{D}+
%%
{D}+    printf("integer");
{D}+"."{D}*({E})?       |
{D}*"."{D}+({E})?       |
{D}+{E}
```

Usage:
There are two steps in compiling a Lex source program. First, the Lex source must be turned into a generated program in the host general purpose language. Then this program must be compiled and loaded, usually with a library of Lex subroutines. The generated program is on a file named lex.yy.c. The I/O library is defined in terms of the C standard library.

The C programs generated by Lex are slightly different on OS/370, because the OS compiler is less powerful than the UNIX or GCOS compilers, and does less at compile time. C programs generated on GCOS and UNIX are the same.

UNIX. The library is accessed by the loader flag -ll. So an appropriate set of commands is lex source cc lex.yy.c -ll The resulting program is placed on the usual file a.out for later execution. To use Lex with Yacc see below. Although the default Lex I/O routines use the C standard library, the Lex automata themselves do not do so; if private versions of input, output and unput are given, the library can be avoided.

Compiling and running a Lex program in Windows machine

1. Install Flex at "C:\GnuWin32"
2. Install Bison at "C:\GnuWin32"
3. Install DevC++ at "C:\Dev-Cpp"
4. Open Environment Variables.
5. Add "C:\GnuWin32\bin;C:\Dev-Cpp\bin;" to path.

Compilation & Execution of your Program:

1. Open Command prompt and switch to your working directory where you have stored your lex file (".l") and yacc file (".y")
2. Let your lex and yacc files be "hello.l" and "hello.y". Now, follow the preceding steps to compile and run your program.
   - For Compiling Lex file only on command prompt:
     ```
     flex hello.l
     gcc lex.yy.c
     ```
   - For Executing the Program on command prompt:
     ```
     a.exe
     ```

Conclusion :-
We have Studied Lex - A Lexical Analyzer Generator.

Experiment Number :- 7

Title: Write a Lex program to count the number of vowels and consonants in a given string.

Tools / Softwares used :- bison-2.4.1-src-setup.exe , flex-2.5.4a-1.exe , Dev C++

Programming Language :- lex

Experiment / Program :-

```
%{
     #include<stdio.h>i
     nt vowels=0;
     int cons=0;
%}
%%
[aeiouAEIOU]
{vowels++;} [a-zA-Z]
{cons++;}
%%
intyywrap()
{
     return 1;
}
main()
{
     printf("Enter the string.. at end press
     ^d\n"); yylex();
     printf("No of vowels=%d\nNo of
     consonants=%d\n", vowels,cons);
}
```

Output :-

E:\Comp department\Jan to May 2022\Subjects\SPCC\Programs\Lex and YAAC\Vowels>flex vowels.l

Experiment Number :- 7

```
E:\Comp department\Jan to May 2022\Subjects\SPCC\Programs\Lex and YAAC\Vowels>gcc lex.yy.c

E:\Comp department\Jan to May 2022\Subjects\SPCC\Programs\Lex and YAAC\Vowels>a.exe
Enter the string of vowels and consonents: at the end press ctrl+z and enter
metropolitan institute of Technology

^Z
Number of vowels are: 13
Number of consonants are: 20
```

Conclusion :-
We have written a Lex program to count the number of vowels and consonants in a given string.

COMPUTER DEPARTMENT

Subject :- SPCC Class :- TE Sem :- VI

Experiment Number :- 8

Title: Write a lex code to count the number of lines, tabs and spaces used in the input,

Tools / Softwares used :- bison-2.4.1-src-setup.exe , flex-2.5.4a-1.exe , Dev C++

Programming Language :- lex

Experiment / Program :-

```
%{
#include<stdio.h>
int lc=0, sc=0, tc=0, ch=0; /*Global variables*/
%}

/*Rule Section*/
%%
\n lc++; //line counter
([ ])+ sc++; //space counter
\t tc++; //tab counter
. ch++;     //characters counter
%%

int yywrap(void){}

int main()
{
           printf("No. of words, lines and spaces \n");
      // The function that starts the analysis
      yylex();

      printf("\nNo. of lines=%d", lc);
      printf("\nNo. of spaces=%d", sc);
      printf("\nNo. of tabs=%d", tc);
      printf("\nNo. of other characters=%d", ch);

}
```

Output :-

```
E:\Comp department\Jan to May 2022\Subjects\SPCC\Programs\Lex and YAAC\No of lin
es>flex lines.l

E:\Comp department\Jan to May 2022\Subjects\SPCC\Programs\Lex and YAAC\No of lin
es>gcc lex.yy.c

E:\Comp department\Jan to May 2022\Subjects\SPCC\Programs\Lex and YAAC\No of lin
es>a.exe
No. of words, lines and spaces
```

```
metropolitan institute  of	technology
and management
oros
^Z

No. of lines=3
No. of spaces=2
No. of tabs=2
No. of other characters=50
```

Conclusion :-
We have written a lex code to count the number of lines, tabs and spaces used in the input.

COMPUTER DEPARTMENT

Subject :- SPCC Class :- TE Sem :- VI

Experiment Number :- 9

Title: Write a lex program to count number of words.

Tools / Softwares used :- bison-2.4.1-src-setup.exe , flex-2.5.4a-1.exe , Dev C++

Programming Language :- lex

Experiment / Program :-

```
%{
#include<stdio.h>
#include<string.h>
int i = 0;
%}

/* Rules Section*/
%%
([a-zA-Z0-9])* {i++;} /* Rule for counting
                                 number of words*/

"\n" {printf("%d\n", i); i = 0;}
%%

int yywrap(void){}

int main()
{
      // The function that starts the analysis
      printf("Enter no. of words \n");
      yylex();

      return 0;
}
```

Output :-

Conclusion :-
We have written a lex program to count number of words.