INDEX

List of Experiments

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S.NO.** | **Experiments** | **Page no.** | **Submission Date** | **Remarks** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Experiment-1**

**Introduction :**

Systems programming, or system programming, is the activity of programming computer system software. The primary distinguishing characteristic of systems programming when compared to application programming is that application programming aims to produce software which provides services to the user directly (e.g. word processor), whereas systems programming aims to produce software and software platforms which provide services to other software, are performance constrained, or both (e.g. operating systems, computational science applications, game engines, industrial automation, and software as a service applications)

Systems programming requires a great degree of hardware awareness. Its goal is to achieve efficient use of available resources, either because the software itself is performance critical or because even small efficiency improvements directly transform into significant monetary savings for the service provider (cloud based word processors).

**Components :**

1:- compiler: A program that converts instructions into a machine code or lower level from so that they can be read and executed by a computer. The process of converting high level programming into machine language is known as compilation.

2:- Assembler: An assembler is a type of computer program that interprets software programs written in assembly language into machine language, code and instructions that can be executed by a computer.

3:- Linker: It is the process of combining various pieces of code and data together to form a single executable file that can be reloaded into the memory.

4:- Loader: Loader is an operating system utility that copies the programs on hard disk to main memory (RAM).

In addition loader also replaces physical addresses with logical addresses .

5:- Macros : A macro is a rule or pattern that specifies how a certain input sequence should be mapped to a replacement output sequence according to a defined procedure .

6:- Editors : Editors or text editors are software programs that enable the user to create and edit text files. Editor is a program which insert, delete, update and add the data.

7:- Debuggers: It is a program used to find and test bugs (errors ) in the programs (other programs ). A debugger is also known as debugging tool.

A debugger allows a programmer to stop a program at any point and examine and change the values of the program.

**Experiment -2**

Aim: To implement a Symbol table with functions to create, insert, modify, search and displaying C language.

Program:

//Compiler Design program to implement a Symbol table with functions to create, insert, modify, search and display.

#include<stdio.h>

#include<conio.h>

#include<alloc.h>

#include<string.h>

#include<stdlib.h>

#define NULL 0

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];

clrscr();

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

{

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);

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--;

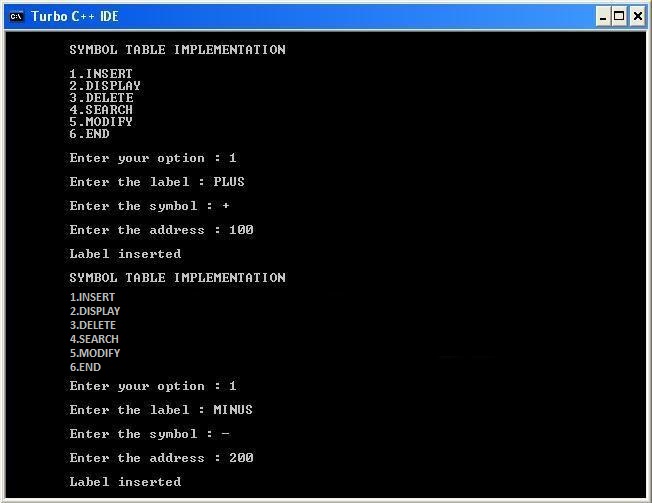
printf("\n\tAfter Deletion:\n");

Display();

}

}

**Output :**

****

**Experiment-3**

Aim : Implementation of single pass assembler.

Program:

//Compiler Design program for implementaion of single pass assembler.

#include<stdio.h>

#include<conio.h>

#include<string.h>

void main()

{

char opcode[10],operand[10],label[10],a[10],ad[10],symbol[10],ch;

char code[10][10],code1[10][10]={"33","44","53","57"};

char mnemonic[10][10]={"START","LDA","STA","LDCH","STCH","END"};

char mnemonic1[10][10]={"LDA","STA","LDCH","STCH"};

int locctr,start,length,i=0,j=0,k,l=0;

int st,diff,address,add,len,actual\_len,finaddr,prevaddr;

FILE \*fp1,\*fp2,\*fp3,\*fp4,\*fp5,\*fp6,\*fp7;

clrscr();

fp1=fopen("INPUT.DAT","r");

fp2=fopen("SYMTAB.DAT","w");

fp3=fopen("INETERMED.DAT","w");

fscanf(fp1,"%s%s%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]);

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);

length=locctr-start;

fcloseall();

printf("\n\nThe contents of Input file:\n\n");

fp1=fopen("INPUT.DAT","r");

ch=fgetc(fp1);

while(ch!=EOF)

{

printf("%c",ch);

ch=fgetc(fp1);

}

printf("\n\nLength of the input program is %d.",length);

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);

}

fcloseall();

fp4=fopen("ASSMLIST.DAT","w");

fp5=fopen("SYMTAB.DAT","r");

fp6=fopen("INTERMED.DAT","r");

fp7=fopen("OBJCODE.DAT","w");

fscanf(fp6,"%s%s%s",label,opcode,operand);

while(strcmp(opcode,"END")!=0)

{

prevaddr=address;

fscanf(fp6,"%d%s%s%s",&address,label,opcode,operand);

}

finaddr=address;

fclose(fp6);

fp6=fopen("INTERMED.DAT","r");

fscanf(fp6,"%s%s%s",label,opcode,operand);

if(strcmp(opcode,"START")==0)

{

fprintf(fp4,"\t%s\t%s\t%s\n",label,opcode,operand);

fprintf(fp7,"H^%s^00%s^00%d\n",label,operand,finaddr);

fscanf(fp6,"%d%s%s%s",&address,label,opcode,operand);

st=address;

diff=prevaddr-st;

fprintf(fp7,"T^00%d^%d",address,diff);

}

while(strcmp(opcode,"END")!=0)

{

if(strcmp(opcode,"BYTE")==0)

{

fprintf(fp4,"%d\t%s\t%s\t%s\t",address,label,opcode,operand);

len=strlen(operand);

actual\_len=len-3;

fprintf(fp7,"^");

for(k=2;k<(actual\_len+2);k++)

{

itoa(operand[k],ad,16);

fprintf(fp4,"%s",ad);

fprintf(fp7,"%s",ad);

}

fprintf(fp4,"\n");

}

else if(strcmp(opcode,"WORD")==0)

{

len=strlen(operand);

itoa(atoi(operand),a,10);

fprintf(fp4,"%d\t%s\t%s\t%s\t00000%s\n",address,label,opcode,operand,a);

fprintf(fp7,"^00000%s",a);

}

else if((strcmp(opcode,"RESB")==0)||(strcmp(opcode,"RESW")==0))

fprintf(fp4,"%d\t%s\t%s\t%s\n",address,label,opcode,operand);

else

{

while(strcmp(opcode,mnemonic1[l])!=0)

l++;

if(strcmp(operand,"COPY")==0)

fprintf(fp4,"%d\t%s\t%s\t%s\t%s0000\n",address,label,opcode,operand,code1[l]);

else

{

rewind(fp5);

fscanf(fp5,"%s%d",symbol,&add);

while(strcmp(operand,symbol)!=0)

fscanf(fp5,"%s%d",symbol,&add);

fprintf(fp4,"%d\t%s\t%s\t%s\t%s%d\n",address,label,opcode,operand,code1[l],add);

fprintf(fp7,"^%s%d",code1[l],add);

}

}

fscanf(fp6,"%d%s%s%s",&address,label,opcode,operand);

}

fprintf(fp4,"%d\t%s\t%s\t%s\n",address,label,opcode,operand);

fprintf(fp7,"\nE^00%d",st);

printf("\nObject Program has been generated.");

fcloseall();

printf("\n\nObject Program:\n\n");

fp7=fopen("OBJCODE.DAT","r");

ch=fgetc(fp7);

while(ch!=EOF)

{

printf("%c",ch);

ch=fgetc(fp7);

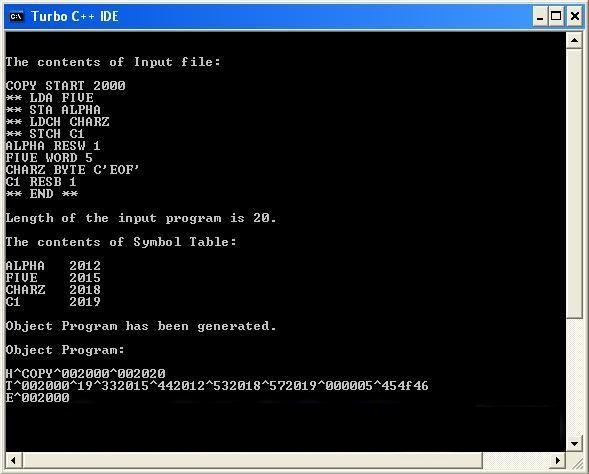
}

fcloseall();

getch();

}

**Output:**



**Experiment- 4**

Aim: Implementation of pass 2 compiler.

Program:

//Compiler design program for implementation of pass two compiler.

#include<stdio.h>

#include<conio.h>

#include<string.h>

#include<stdlib.h>

void main()

{

char a[10],ad[10],label[10],opcode[10],operand[10],mnemonic[10],symbol[10];

int i,address,code,add,len,actual\_len;

FILE \*fp1,\*fp2,\*fp3,\*fp4;

clrscr();

fp1=fopen("assmlist.dat","w");

fp2=fopen("symtab.dat","r");

fp3=fopen("intermediate.dat","r");

fp4=fopen("optab.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);

fscanf(fp3,"%d%s%s%s",&address,label,opcode,operand);

}

while(strcmp(opcode,"END")!=0)

{

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;

for(i=2;i<(actual\_len+2);i++)

{

itoa(operand[i],ad,16);

fprintf(fp1,"%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);

}

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

{

rewind(fp4);

fscanf(fp4,"%s%d",mnemonic,&code);

while(strcmp(opcode,mnemonic)!=0)

fscanf(fp4,"%s%d",mnemonic,&code);

if(strcmp(operand,"\*\*")==0)

{

fprintf(fp1,"%d\t%s\t%s\t%s\t%d0000\n",address,label,opcode,operand,code);

}

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%d%d\n",address,label,opcode,operand,code,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);

printf("Finished");

fclose(fp1);

fclose(fp2);

fclose(fp3);

fclose(fp4);

getch();

}

**INTERMEDIATE.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'EOF'

2019 C1 RESB 1

2020 \*\* END \*\*

**ASSMLIST.DAT**

\*\* 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 \*\*

**OPTAB.DAT**

LDA 33

STA 44

LDCH 53

STCH 57

END \*

**SYMTAB.DAT**

ALPHA 2012

FIVE 2015

CHARZ 2018

C1 2019

**Experiment – 5**

**Aim** : Write a program for implementing a Lexical analyzer using LEX tool in Linux platform.

**Algorithm**:

**Step1**: Lex program contains three sections: definitions, rules, and user subroutines. Each section must be separated from the others by a line containing only the delimiter, %%. The format is as follows: definitions %% rules %% user\_subroutines

**Step2**: In definition section, the variables make up the left column, and their definitions make up the right column. Any C statements should be enclosed in %{..}%. Identifier is defined such that the first letter of an identifier is alphabet and remaining letters are alphanumeric.

**Step3**: In rules section, the left column contains the pattern to be recognized in an input file to yylex(). The right column contains the C program fragment executed when that pattern is recognized. The various patterns are keywords, operators, new line character, number, string, identifier, beginning and end of block, comment statements, preprocessor directive statements etc.

**Step4**: Each pattern may have a corresponding action, that is, a fragment of C source code to execute when the pattern is matched.

**Step5**: When yylex() matches a string in the input stream, it copies the matched text to an external character array, yytext, before it executes any actions in the rules section.

**Step6**: In user subroutine section, main routine calls yylex(). yywrap() is used to get more input.

**Step7**: The lex command uses the rules and actions contained in file to generate a program, lex.yy.c, which can be compiled with the cc command. That program can then receive input, break the input into the logical pieces defined by the rules in file, and run program fragments contained in the actions in file.

**Program**:

//Implementation of Lexical Analyzer using Lex tool

%

{

int COMMENT = 0; %

}

identifier[a - zA - Z][a - zA - Z0 - 9] \*

%

% #.\*{

printf("\n%s is a preprocessor directive", yytext);

}

int |

float |

char |

double |

while |

for |

struct |

typedef |

do |

if |

break |

continue |

void |

switch |

return |

else |

goto {

printf("\n\t%s is a keyword", yytext);

}

"/\*" {

COMMENT = 1;

} {

printf("\n\t %s is a COMMENT", yytext);

} {

identifier

}\({

if (!COMMENT) printf("\nFUNCTION \n\t%s", yytext);

}\ {

{

if (!COMMENT) printf("\n BLOCK BEGINS");

}\

} {

if (!COMMENT) printf("BLOCK ENDS ");

} {

identifier

}(\[

[0 - 9] \* \

]) ? {

if (!COMMENT) printf("\n %s IDENTIFIER", yytext);

}\

".\*\" {if(!COMMENT)printf("\

n\ t % s is a STRING ",yytext);} [0 - 9] + {

if (!COMMENT) printf("\n %s is a NUMBER ", yytext);

}\)(\: ) ? {

if (!COMMENT) printf("\n\t");ECHO;printf("\n");

}\(ECHO; = {

if (!COMMENT) printf("\n\t %s is an ASSIGNMENT OPERATOR", yytext);

}\ <= | \ >= | \ < |

==

| \ > {

if (!COMMENT) printf("\n\t%s is a RELATIONAL OPERATOR", yytext);

} %

%

int main(int argc, char \*\* argv) {

FILE \* file;

file = fopen("var.c", "r");

if (!file) {

printf("could not open the file");

exit(0);

}

yyin = file;

yylex();

printf("\n");

return (0);

}

int yywrap() {

return (1);

}

**Var.c**

#include<stdio.h>

#include<conio.h>

void main()

{

int a,b,c;

a=1;

b=2;

c=a+b;

printf("Sum:%d",c);

}

**Experiment-6**

**Aim:** Write a program to compute FIRST function.

**Program:**

//Compiler design program to to compute FIRST function.

#include<stdio.h>

#include<ctype.h>

void FIRST(char[],char );

void add\_result\_set(char[],char);

int num\_productions;

char productionSet[10][10];

main()

{

int i;

char choice;

char c;

char result[20];

printf("How many number of productions ? :");

scanf(" %d",&num\_productions);

for(i=0;i<num\_productions;i++)//read production string eg: E=E+T

{

printf("Enter productions Number %d : ",i+1);

scanf(" %s",productionSet[i]);

}

do

{

printf("\n Find the FIRST of :");

scanf(" %c",&c);

FIRST(result,c); //Compute FIRST; Get Answer in 'result' array

printf("\n FIRST(%c)= { ",c);

for(i=0;result[i]!='\0';i++)

printf(" %c ",result[i]); //Display result

printf("}\n");

printf("press 'y' to continue : ");

scanf(" %c",&choice);

}

while(choice=='y'||choice =='Y');

}

/\*

\*Function FIRST:

\*Compute the elements in FIRST(c) and write them

\*in Result Array.

\*/

void FIRST(char\* Result,char c)

{

int i,j,k;

char subResult[20];

int foundEpsilon;

subResult[0]='\0';

Result[0]='\0';

//If X is terminal, FIRST(X) = {X}.

if(!(isupper(c)))

{

add\_result\_set(Result,c);

return ;

}

//If X is non terminal

//Read each production

for(i=0;i<num\_productions;i++)

{

//Find production with X as LHS

if(productionSet[i][0]==c)

{

//If X → ε is a production, then add ε to FIRST(X).

if(productionSet[i][2]=='$') add\_result\_set(Result,'$');

//If X is a non-terminal, and X → Y1 Y2 … Yk

//is a production, then add a to FIRST(X)

//if for some i, a is in FIRST(Yi),

//and ε is in all of FIRST(Y1), …, FIRST(Yi-1).

else

{

j=2;

while(productionSet[i][j]!='\0')

{

foundEpsilon=0;

FIRST(subResult,productionSet[i][j]);

for(k=0;subResult[k]!='\0';k++)

add\_result\_set(Result,subResult[k]);

for(k=0;subResult[k]!='\0';k++)

if(subResult[k]=='$')

{

foundEpsilon=1;

break;

}

//No ε found, no need to check next element

if(!foundEpsilon)

break;

j++;

}

}

}

}

return ;

}

/\* add\_result\_set adds the computed

\*element to result set.

\*This code avoids multiple inclusion of elements

\*/

void add\_result\_set(char Result[],char val)

{

int k;

for(k=0 ;Result[k]!='\0';k++)

if(Result[k]==val)

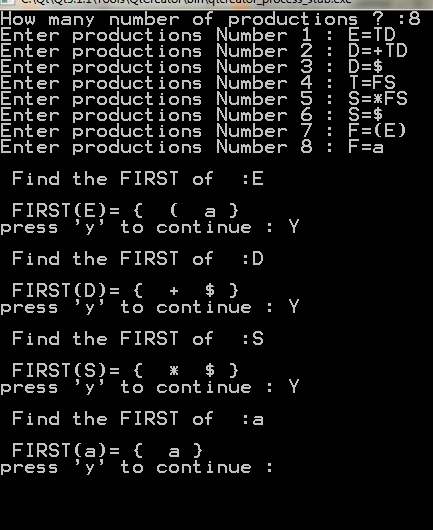
return;

Result[k]=val;

Result[k+1]='\0';

}

**Output:**



**Experiment -7**

**Aim:** Write a program to compute FOLLOW function.

**Program:**

//Compiler design program to find the FOLLOW of given grammer.

#include<stdio.h>

#include<ctype.h>

#include<string.h>

int n,m=0,p,i=0,j=0;

char a[10][10],follow\_result[10];

void follow(char c);

void first(char c);

void add\_results(char);

int main()

{

int i;

int choice;

char c,ch;

clrscr();

printf("Enter the no.of productions: ");

scanf("%d", &n);

printf(" Enter %d productions\nProduction with multiple terms should be give as separate productions \n", n);

for(i=0;i<n;i++)

scanf("%s%c",a[i],&ch);

do

{

m=0;

printf("Find FOLLOW of -->");

scanf(" %c",&c);

follow(c);

printf("FOLLOW(%c) = { ",c);

for(i=0;i<m;i++)

printf("%c ",follow\_result[i]);

printf(" }\n");

printf("Do you want to continue(Press 1 to continue....)?");

scanf("%d",&choice);

}

while(choice==1);

}

void follow(char c)

{

if(a[0][0]==c)add\_results('$');

for(i=0;i<n;i++)

{

for(j=2;j<strlen(a[i]);j++)

{

if(a[i][j]==c)

{

if(a[i][j+1]!='\0')first(a[i][j+1]);

if(a[i][j+1]=='\0'&&c!=a[i][0])

follow(a[i][0]);

}

}

}

}

void first(char c)

{

int k;

if(!(isupper(c)))

/\*f[m++]=c; \*/

add\_results(c);

for(k=0;k<n;k++)

{

if(a[k][0]==c)

{

if(a[k][2]=='#') follow(a[i][0]);

else if(islower(a[k][2]))

/\*f[m++]=a[k][2]; \*/

add\_results(a[k][2]);

else first(a[k][2]);

}

}

}

void add\_results(char c)

{

int i;

for( i=0;i<=m;i++)

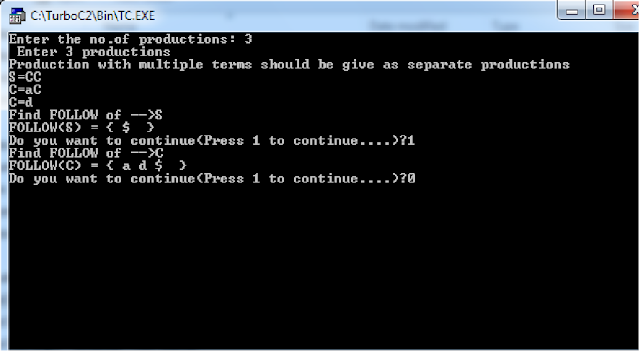
if(follow\_result[i]==c)

return;

follow\_result[m++]=c;

}

**Output :**

****