الجامعة الإسلامية العالمية شيتاغونغ

International Islamic University Chittagong

LAB MANUAL

CSE-3528 (Compiler Sessional)

Abstract

This is a course to familiarize the students with knowledge and ideas of the construction of computer program translators called compilers. To take this course, students should be able to gain the skills of automata theory which need to design and implementation of the compiler. The competitive and standard programming ability with some mathematical maturity also will be expected; students should have some idea of computer language grammar and computing skills will also be helpful. After studying this course the students will have the knowledge of the design and implementation of compilers several elements, maintaining the standard steps.

Md. Nazmul Arefin

nazmul.arefin@iiuc.ac.bd

Table of Contents

Experiment 1: Write a C program to construct a lexical analyzer that should ignore redundant spaces, tabs, new lines and comments.

Instructions:

a. Read the input Expression
b. Check whether input is alphabet or digits then store it as identifier
c. If the input is operator store it as symbol
d. Check the input for keywords

Code:

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

// fun which separate the key and id
void sep_key_id(char str[10]){
      if(strcmp("for",str)==0||strcmp("while",str)==0||strcmp("do",str)==0||strcmp(
"int",str)==0||strcmp("float",str)==0||strcmp("char",str)==0||strcmp("double",str)=
=0||strcmp("static",str)==0||strcmp("switch",str)==0||strcmp("case",str)==0){
        printf("%s is a keyword\n",str);
    }
    else{
        printf("%s is an identifier\n",str);
    }
}

int main(){

    FILE *f1, *f2;
    char c;
    int totalLine = 0;

    // Reading program from user
    // and store it to a file: f1
    f1 = fopen("input.txt","w");
    while((c=getchar())!=EOF){
        putc(c,f1);
    }
    fclose(f1);

    //checking digit and alpha and store
    // it to another file: f2
    f2 = fopen("key_id.txt","w");
    f1 = fopen("input.txt","r");
    while((c=getc(f1)) != EOF){
        if(isalpha(c)){
            putc(c,f2);
            c = getc(f1);
            while(isdigit(c) || isalpha(c) || c == '_'){
                putc(c,f2);
                c = getc(f1);
            }
            putc(' ',f2);
        }
        else if(c == ' ' || c == '\t')
            printf(" ");
        else if(c == '\n')
            totalLine++;
    }
    fclose(f1);
    fclose(f2);
    //calling fun for separated keywords and
    // identifiers
    f2 = fopen("key_id.txt","r");
    char str[10];
    int j = 0;

    while((c=getc(f2))!= EOF){
        if(c != ' '){
            str[j++] = c;
        }else{
            str[j] = '\0';
            sep_key_id(str);
            j = 0;
        }
    }
    fclose(f2);
    printf("total line: %d \n", totalLine);

    return 0;
}
```

Input	Output

`Enter the c program`
`#include<stdio.h>`
`int main(){`
`int a = 5;`
`printf("%d",a);`
`}`
	`include is an identifier`
	`stdio is an identifier`
	`h is an identifier`
	`int is a keyword`
	`main is an identifier`
	`int is a keyword`
	`a is an identifier`
	`printf is an identifier`
	`d is an identifier`
	`a is an identifier`
	`total line: 6`

Experiment 2: Write a C program to identify whether a given line is a comment or not.

Instructions:

1. Read the input string.
2. Check whether the string is starting with '/' and check next character is '/' or'*'.
3. If condition satisfies print comment.
4. Else not a comment.

Code:

```
#include<stdio.h>
void main()
{
    char str[30];
    int i=2,a=0;
    printf("\n Enter comment:");
    gets(str);
    if(str[0]=='/')
    {
        if(str[1]=='/')
            printf("\n It is a comment");
        else if(str[1]=='*')
        {
            for(i=2; i<=30; i++)
            {
                if(str[i]=='*'&&str[i+1]=='/')
                {
                    printf("\n It is a comment");
                    a=1;
                    break;
                }
                else
                    continue;
            }
            if(a==0)
                printf("\n It is not a comment");
        }
        else
            printf("\n It is not a comment");
    }
    else
        printf("\n It is not a comment");
}
```

Experiment 3: Write a C program to recognize strings under a^+, a^*b^+

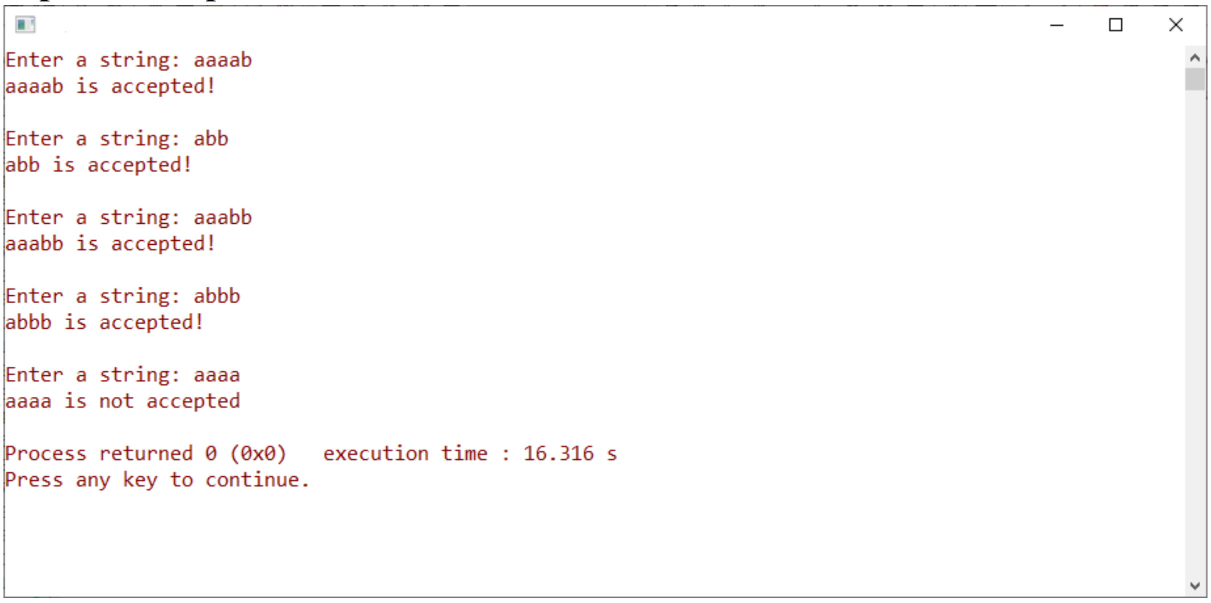
Instruction:

1. By using transition diagram we verify input of the state.
2. If the state recognize the given pattern rule.
3. Then print string is accepted under a^+ or a^*b^+
4. Else print string not accepted.

Code:

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

int main()
{
    char str[10];
    printf("Enter a string: ");
    while(gets(str)){
        char c;
        int state = 0, i = 0;
        while(str[i] != '\0'){
            switch(state){
                case 0:
                    c = str[i++];
                    if(c == 'a')
                        state = 1;
                    else if (c == 'b')
                        state = 2;
                    else
                        state = 3;
                    break;
                case 1:
                    c = str[i++];
                    if(c == 'a')
                        state = 1;
                    else if(c == 'b')
                        state = 2;
                    else
                        state = 3;

                    break;
                case 2:
                    c = str[i++];
                    if(c == 'a')
                        state = 3;
                    else if(c == 'b')
                        state = 2;
                    else
                        state = 3;

                    break;
                case 3:
                    printf("%s is not accepted!\n",str);
                    exit(0);
            }

        }
        if(state == 2){
            printf("%s is accepted!\n\nEnter a string: ",str);
        }else
        {
                printf("%s is not accepted\n",str);
                exit(0);
           }
      }
      return 0;
}
```

Input & Output:



```
Enter a string: aaaab
aaaab is accepted!

Enter a string: abb
abb is accepted!

Enter a string: aaabb
aaabb is accepted!

Enter a string: abbb
abbb is accepted!

Enter a string: aaaa
aaaa is not accepted

Process returned 0 (0x0)   execution time : 16.316 s
Press any key to continue.
```

Experiment 4: Write a C program to test whether a given identifier is valid or not

Instruction:

1. Read the given input string.
2. Check the initial character of the string is numerical or any special character except ‘_’ then print it is not a valid Identifier.
3. Otherwise print it as valid identifier if remaining characters of string doesn’t contains any special characters except ‘_’.

Code:

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

void main()
{
    char a[10];
    int id, i=1;

    printf("\n Enter an identifier:");
    gets(a);
    if(isalpha(a[0]))
        id=1;
    else
        printf("\n Not a valid identifier");
    while(a[i]!='\0')
    {
        if(!isdigit(a[i])&&!isalpha(a[i]))
        {
            id=0;
            break;
        }
        i++;
    }
    if(id==1)
        printf("\n Valid identifier");
}
```

Experiment 5: Write a C program to simulate lexical analyzer for validating operators.

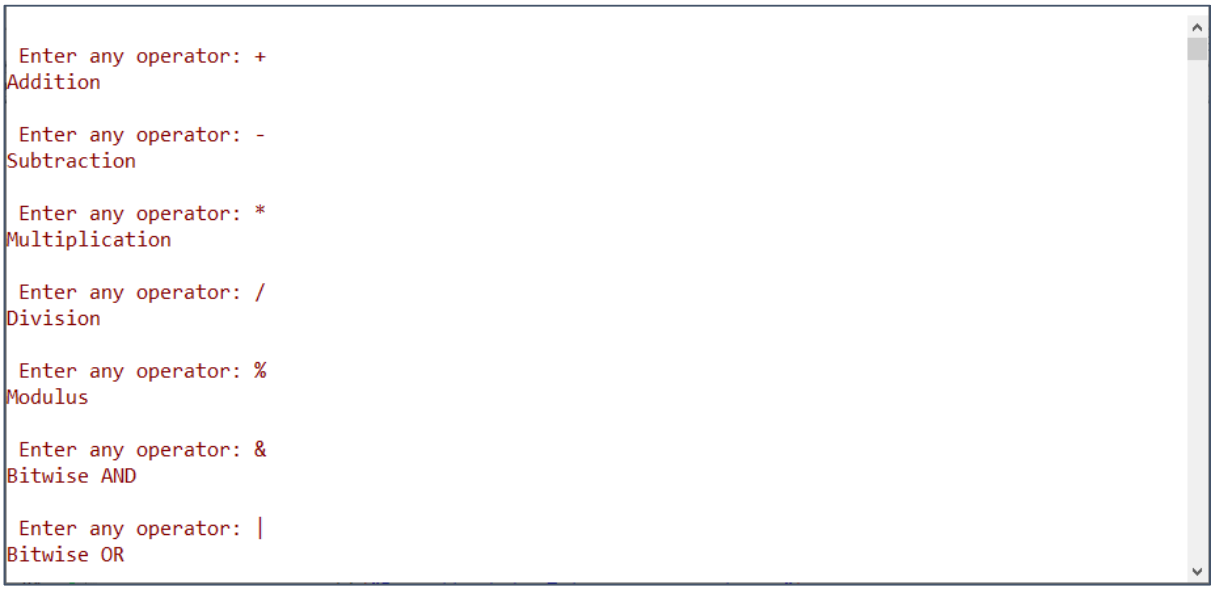
Instruction:

1. Read the given input.
2. If the given input matches with any operator symbol.
3. Then display in terms of words of the particular symbol.
4. Else print not an operator.

Code:

```
#include<stdio.h>

int main()
{
    char s[5];
    printf("\n Enter any operator: ");
    while(gets(s))
    {
        switch(s[0])
        {
        case'>':
            if(s[1]=='=')
                printf("Greater than or equal\n\n Enter any operator: ");
            else
                printf("Greater than\n\n Enter any operator: ");
            break;
        case'<':
            if(s[1]=='=')
                printf("Less than or equal\n\n Enter any operator: ");
            else
                printf("Less than\n\n Enter any operator: ");
            break;
        case'=':
            if(s[1]=='=')
                printf("Equal to\n\n Enter any operator: ");
            else
                printf("Assignment\n\n Enter any operator: ");
            break;
        case'!':
            if(s[1]=='=')
                printf("Not Equal\n\n Enter any operator: ");
            else
                printf("Bit Not\n\n Enter any operator: ");
            break;
        case'&':
            if(s[1]=='&')
                printf("Logical AND\n\n Enter any operator: ");
            else
                printf("Bitwise AND\n\n Enter any operator: ");
            break;
        case'|':
            if(s[1]=='|')
                printf("Logical OR\n\n Enter any operator: ");
            else
                printf("Bitwise OR\n\n Enter any operator: ");
            break;
        case'+':
            printf("Addition\n\n Enter any operator: ");
            break;
        case'-':
            printf("Subtraction\n\n Enter any operator: ");
            break;
        case'*':
            printf("Multiplication\n\n Enter any operator: ");
            break;
        case'/':
            printf("Division\n\n Enter any operator: ");
            break;
        case'%':
            printf("Modulus\n\n Enter any operator: ");
            break;
        default:
            printf("Not an operator\n\n Enter any operator: ");
        }
    }
}
```

Input & Output:



```
 Enter any operator: +
Addition

 Enter any operator: -
Subtraction

 Enter any operator: *
Multiplication

 Enter any operator: /
Division

 Enter any operator: %
Modulus

 Enter any operator: &
Bitwise AND

 Enter any operator: |
Bitwise OR
```

Experiment 6: Write a C program to implement a symbol table.

Instruction:

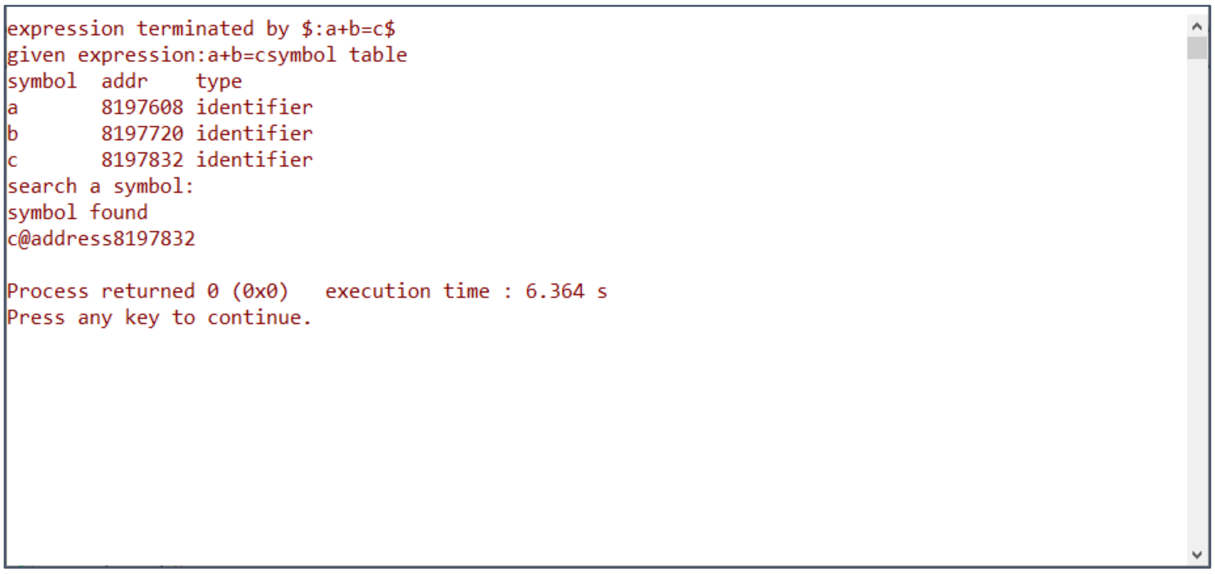
A Symbol table is a data structure used by a language translator such as a compiler or interpreter, where each identifier in a program's source code is associated with information relating to its declaration or appearance in the source.

1. Get the input from the user with the terminating symbol '$'.
2. Allocate memory for the variable by dynamic memory allocation function.
3. If the next character of the symbol is an operator then only the memory is allocated.
4. While reading, the input symbol is inserted into symbol table along with its memory address.
5. The steps are repeated till "$" is reached.
6. To reach a variable, enter the variable to the searched and symbol table has been checked for corresponding variable, the variable along its address is displayed as result.

Code:

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>
#include<string.h>
#include<math.h>
#include<ctype.h>
int main()
{
    int i=0,j=0,x=0,n,flag=0;
    void *p,*add[15];
    char ch,srch,b[15],d[15],c;

    printf("expression terminated by $:");
    while((c=getchar())!='$')
    {
        b[i]=c;
        i++;
    }
    n=i-1;
    printf("given expression:");
    i=0;
    while(i<=n)
    {
        printf("%c",b[i]);
        i++;
    }
    printf("symbol table\n");
    printf("symbol\taddr\ttype\n");
    while(j<=n)
    {
        c=b[j];
        if(isalpha('c'-0))
        {
            if(j==n)
            {
                p=malloc(c);
                add[x]=p;
                d[x]=c;
                printf("%c\t%d\tidentifier\n",c,p);
            }
            else
            {
                ch=b[j+1];
                if(ch=='+'||ch=='-'||ch=='*'||ch=='=')
                {
                    p=malloc(c);
                    add[x]=p;
                    d[x]=c;
                    printf("%c\t%d\tidentifier\n",c,p);
                    x++;
                }
            }
        }
        j++;
    }
    printf("the symbol is to be searched\n");
    srch=getch();
    for(i=0; i<=x; i++)
    {
        if(srch==d[i])
        {
            printf("symbol found\n");
            printf("%c%s%d\n",srch,"@address",add[i]);
            flag=1;
        }
    }
    if(flag==0)
        printf("symbol not found\n");
}
```

Input & Output:



```
expression terminated by $:a+b=c$
given expression:a+b=csymbol table
symbol  addr    type
a       8197608 identifier
b       8197720 identifier
c       8197832 identifier
search a symbol:
symbol found
c@address8197832

Process returned 0 (0x0)   execution time : 6.364 s
Press any key to continue.
```

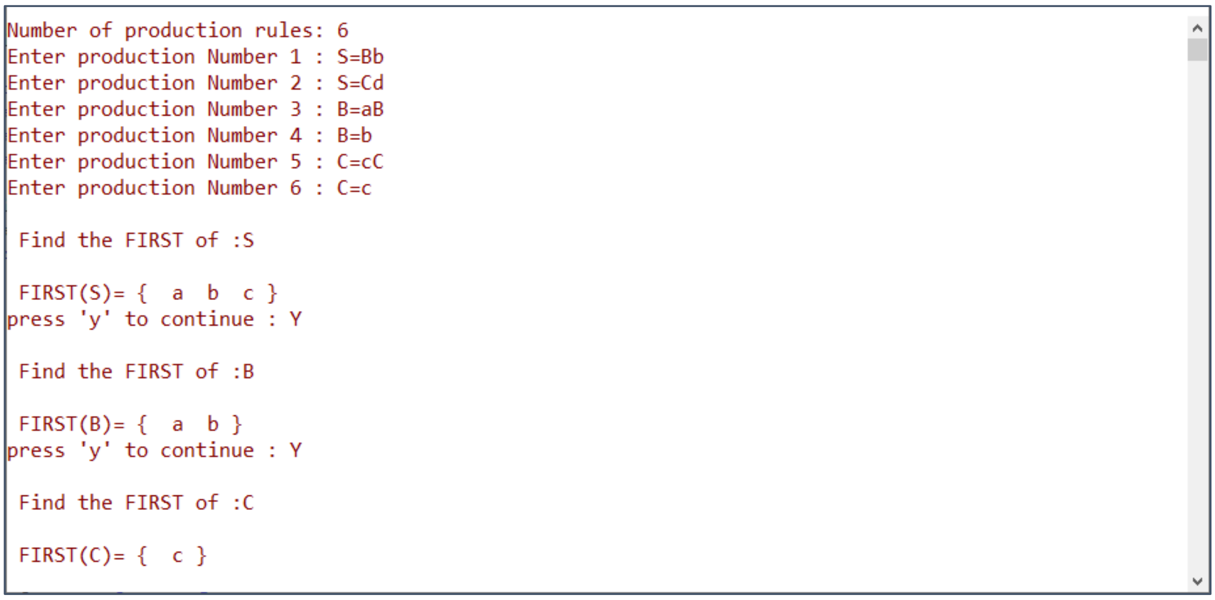
Experiment 7: Find the “first” of a given grammar

Code:

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

int nop;
char production[10][10];

void result(char res[],char val)
{
    int k;
    for(k=0 ; res[k]!='\0'; k++)
        if(res[k]==val)
            return;
    res[k]=val;
    res[k+1]='\0';
}

void FIRST(char res[],char c)
{
    int i,j,k;
    char subres[5];
    int eps;
    subres[0]='\0';
    res[0]='\0';
    if(!(isupper(c)))
    {
        result(res,c);
        return ;
    }
    for(i=0; i<nop; i++)
    {
        if(production[i][0]==c)
        {
            if(production[i][2]=='$')
                result(res,'$');
            else
            {
                j=2;
                while(production[i][j]!='\0')
                {
                    eps=0;
                    FIRST(subres,production[i][j]);
                    for(k=0; subres[k]!='\0'; k++)
                        result(res,subres[k]);
                    for(k=0; subres[k]!='\0'; k++)
                        if(subres[k]=='$')
                        {
                            eps=1;
                            break;
                        }
                    if(!eps)
                        break;
                    j++;
                }
            }
        }
    }
    return;
}

int main()
```

```
{
    int i;
    char choice;
    char c;
    char res1[20];

    printf("Number of production rules: ");
    scanf(" %d",&nop);
    for(i=0; i<nop; i++)
    {
        printf("Enter production Number %d : ",i+1);
        scanf(" %s",production[i]);
    }
    do
    {
        printf("\n Find the FIRST of :");
        scanf(" %c",&c);
        FIRST(res1,c);
        printf("\n FIRST(%c)= { ",c);
        for(i=0; res1[i]!='\0'; i++)
            printf(" %c ",res1[i]);
        printf("}\n");
        printf("press 'y' to continue : ");
        scanf(" %c",&choice);
    }
    while(choice=='y'||choice =='Y');

    return 0;
}
```

Input & output:



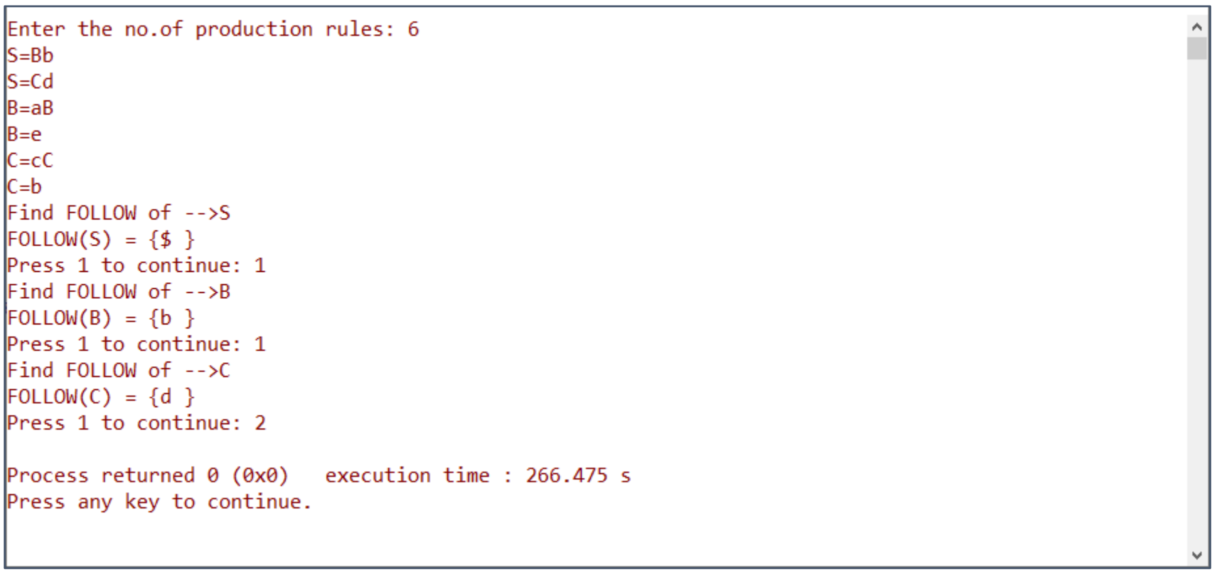
```
Number of production rules: 6
Enter production Number 1 : S=Bb
Enter production Number 2 : S=Cd
Enter production Number 3 : B=aB
Enter production Number 4 : B=b
Enter production Number 5 : C=cC
Enter production Number 6 : C=c

 Find the FIRST of :S

 FIRST(S)= {  a  b  c }
press 'y' to continue : Y

 Find the FIRST of :B

 FIRST(B)= {  a  b }
press 'y' to continue : Y

 Find the FIRST of :C

 FIRST(C)= {  c }
```

Experiment 8: Find the "Follow" of a given grammar.

Code:

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

int nop,m=0,p,i=0,j=0;
char prod[10][10],res[10];
void FOLLOW(char c);
void first(char c);
void result(char);
int main()
{
    int i;
    int choice;
    char c,ch;
    printf("Enter the no.of production rules: ");
    scanf("%d", &nop);
    for(i=0; i<nop; i++)
    {
        //printf("Enter productions Number %d : ",i+1);
        scanf(" %s",prod[i]);
    }
    do
    {
        m=0;
        printf("Find FOLLOW of -->");
        scanf(" %c",&c);
        FOLLOW(c);
        printf("FOLLOW(%c) = {",c);
        for(i=0; i<m; i++)
            printf("%c ",res[i]);
        printf("}\n");
        printf("Press 1 to continue: ");
        scanf("%d%c",&choice,&ch);
    }
    while(choice==1);
}
void FOLLOW(char c)
{
    if(prod[0][0]==c)
        result('$');
    for(i=0; i<nop; i++)
    {
        for(j=2; j<strlen(prod[i]); j++)
        {
            if(prod[i][j]==c)
            {
                if(prod[i][j+1]!='\0')
                    first(prod[i][j+1]);
                if(prod[i][j+1]=='\0'&&c!=prod[i][0])
                    FOLLOW(prod[i][0]);
            }
        }
    }
}
void first(char c)
{
    int k;
    if(!(isupper(c)))
        result(c);
    for(k=0; k<nop; k++)
    {
        if(prod[k][0]==c)
        {
            if(prod[k][2]=='$')
                FOLLOW(prod[i][0]);
            else if(islower(prod[k][2]))
                result(prod[k][2]);
            else
                first(prod[k][2]);
        }
    }
}
void result(char c)
{
    int i;
    for( i=0; i<=m; i++)
        if(res[i]==c)
            return;
    res[m++]=c;
}

//input
/*
6
S=Bb
S=Cd
B=aB
B=e
C=cC
C=b
*/
```

Input & Output:



```
Enter the no.of production rules: 6
S=Bb
S=Cd
B=aB
B=e
C=cC
C=b
Find FOLLOW of -->S
FOLLOW(S) = {$ }
Press 1 to continue: 1
Find FOLLOW of -->B
FOLLOW(B) = {b }
Press 1 to continue: 1
Find FOLLOW of -->C
FOLLOW(C) = {d }
Press 1 to continue: 2

Process returned 0 (0x0)   execution time : 266.475 s
Press any key to continue.
```

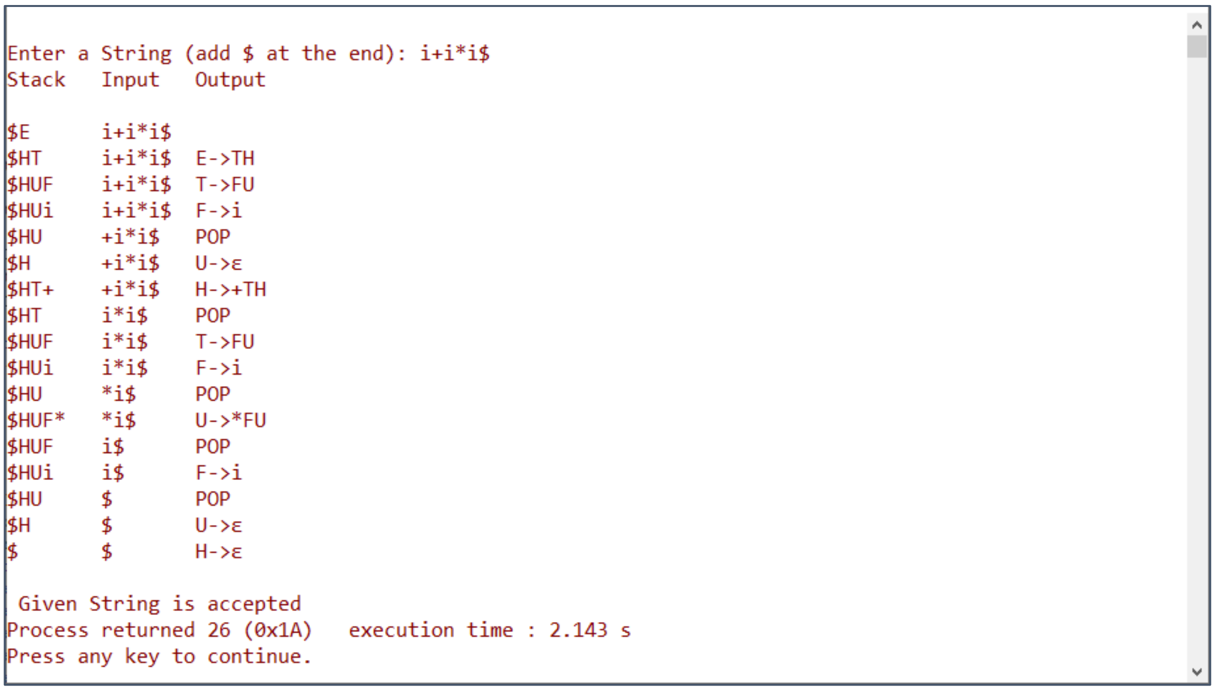
Experiment 9: Construct LL (1) parser of a following grammar-

```
E->TH
T->FU
F->i
U->*FU/ε
H->+TH/ε
```

Code:

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

char a[10];
int top=-1,i;
void error()
{
    printf("Syntax Error");
}
void push(char k[])
{
    for(i=0; k[i]!='\0'; i++)
    {
        if(top<9)
            a[++top]=k[i];
    }
}
char TOS()
{
    return a[top];
}
void pop()
{
    if(top>=0)
        a[top--]='\0';
}
void display()
{
    for(i=0; i<=top; i++)
        printf("%c",a[i]);
}
void display1(char p[],int m)
{
    int l;
    printf("\t");
    for(l=m; p[l]!='\0'; l++)
        printf("%c",p[l]);
}
char* stack()
{
    return a;
}
void main()
{
    char ip[20],r[20],nt,cin;
    int ir,ic,j=0,k;
    char t[5][6][10]= {"$","$","TH","$","TH","$",
                       "+TH","$","$","e","$","e",
                       "$","$","FU","$","FU","$",
                       "e","*FU","$","e","$","e",
                       "$","$","(E)","$","i","$"
                      };

    printf("\nEnter a String (add $ at the end): ");
    gets(ip);
    printf("Stack\tInput\tOutput\n\n");
    push("$E");
    display();
    printf("\t%s\n",ip);
    for(j=0; ip[j]!='\0';)
    {
        if(TOS()==cin)
        {
            pop();
            display();
            display1(ip,j+1);
            printf("\tPOP\n");
            j++;
        }
        cin=ip[j];
        nt=TOS();
        if(nt=='E')ir=0;
        else if(nt=='H')ir=1;
        else if(nt=='T')ir=2;
        else if(nt=='U')ir=3;
        else if(nt=='F')ir=4;
        else
        {
            error();
            break;
        }
        if(cin=='+')ic=0;
        else if(cin=='*')ic=1;
        else if(cin=='(')ic=2;
        else if(cin==')')ic=3;
        else if(isalpha(cin))
        {
            ic=4;
            cin='i';
        }
        else if(cin=='$')ic=5;
        strcpy(r,strrev(t[ir][ic]));
        strrev(t[ir][ic]);
        pop();
        push(r);
        if(TOS()=='e')
        {
            pop();
            display();
            display1(ip,j);
            printf("\t%c->%c\n",nt,238);
        }
        else
        {
            display();
            display1(ip,j);
            printf("\t%c->%s\n",nt,t[ir][ic]);
        }
        if(TOS()=='$'&&cin=='$')
            break;
        if(TOS()=='$')
        {
            error();
            break;
        }
    }
    k=strcmp(stack(),"$");
    if(k==0)
        printf("\n Given String is accepted");
    else
        printf("\n Given String is not accepted");
}
```

Input & Output:



```
Enter a String (add $ at the end): i+i*i$
Stack   Input   Output

$E      i+i*i$
$HT     i+i*i$  E->TH
$HUF    i+i*i$  T->FU
$HUi    i+i*i$  F->i
$HU     +i*i$   POP
$H      +i*i$   U->ε
$HT+    +i*i$   H->+TH
$HT     i*i$    POP
$HUF    i*i$    T->FU
$HUi    i*i$    F->i
$HU     *i$     POP
$HUF*   *i$     U->*FU
$HUF    i$      POP
$HUi    i$      F->i
$HU     $       POP
$H      $       U->ε
$       $       H->ε

 Given String is accepted
Process returned 26 (0x1A)   execution time : 2.143 s
Press any key to continue.
```

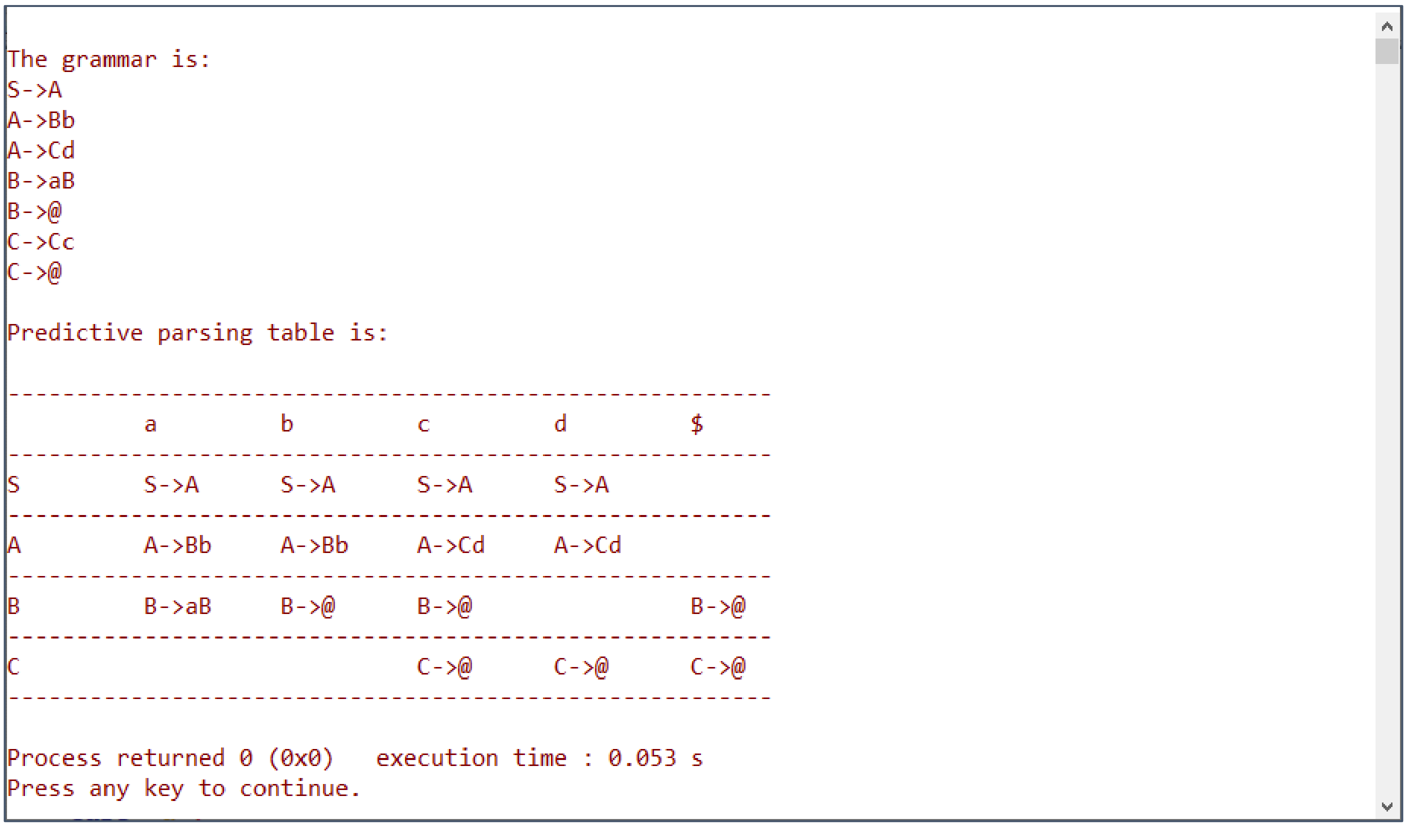
Experiment 10: Implement the predictive parser for the following grammar-

```
S->A
A->Bb
A->Cd
B->aB
B->@
C->Cc
C->@
```

Code:

```
#include<stdio.h>
#include<string.h>
char prol[7][10]= {"S","A","A","B","B","C","C"};
char pror[7][10]= {"A","Bb","Cd","aB","@","Cc","@"};
char prod[7][10]= {"S->A","A->Bb","A->Cd","B->aB","B->@","C->Cc","C->@"};
char first[7][10]= {"abcd","ab","cd","a@","@","c@","@"};
char follow[7][10]= {"$","$","$","a$","b$","c$","d$"};
char table[5][6][10];
numr(char c)
{
    switch(c)
    {
    case 'S':
        return 0;
    case 'A':
        return 1;
    case 'B':
        return 2;
    case 'C':
        return 3;
    case 'a':
        return 0;
    case 'b':
        return 1;
    case 'c':
        return 2;
    case 'd':
        return 3;
    case '$':
        return 4;
    }
    return(2);
}
int main()
{
    int i,j,k;
    for(i=0; i<5; i++)
        for(j=0; j<6; j++)
            strcpy(table[i][j]," ");
    printf("\nThe grammar is:\n");
    for(i=0; i<7; i++)
        printf("%s\n",prod[i]);
    printf("\nPredictive parsing table is:\n");
    fflush(stdin);
    for(i=0; i<7; i++)
    {
        k=strlen(first[i]);
        for(j=0; j<10; j++)
            if(first[i][j]!='@')
                strcpy(table[numr(prol[i][0])+1][numr(first[i][j])+1],prod[i]);
    }
    for(i=0; i<7; i++)
    {
        if(strlen(pror[i])==1)
        {
            if(pror[i][0]=='@')
            {
                k=strlen(follow[i]);
                for(j=0; j<k; j++)

strcpy(table[numr(prol[i][0])+1][numr(follow[i][j])+1],prod[i]);
            }
        }
    }
    strcpy(table[0][0]," ");
    strcpy(table[0][1],"a");
    strcpy(table[0][2],"b");
    strcpy(table[0][3],"c");
    strcpy(table[0][4],"d");
    strcpy(table[0][5],"$");
    strcpy(table[1][0],"S");
    strcpy(table[2][0],"A");
    strcpy(table[3][0],"B");
    strcpy(table[4][0],"C");
    printf("\n--------------------------------------------------------\n");
    for(i=0; i<5; i++)
    {
        for(j=0; j<6; j++)
        {
            printf("%-10s",table[i][j]);
            if(j==5)
                printf("\n--------------------------------------------------------
\n");
        }
    }
    return 0;
}
```

Output:



```
The grammar is:
S->A
A->Bb
A->Cd
B->aB
B->@
C->Cc
C->@

Predictive parsing table is:

--------------------------------------------------------
          a         b         c         d         $
--------------------------------------------------------
S         S->A      S->A      S->A      S->A
--------------------------------------------------------
A         A->Bb     A->Bb     A->Cd     A->Cd
--------------------------------------------------------
B         B->aB     B->@      B->@                B->@
--------------------------------------------------------
C                             C->@      C->@      C->@
--------------------------------------------------------

Process returned 0 (0x0)   execution time : 0.053 s
Press any key to continue.
```

------------------------------x------------------------------