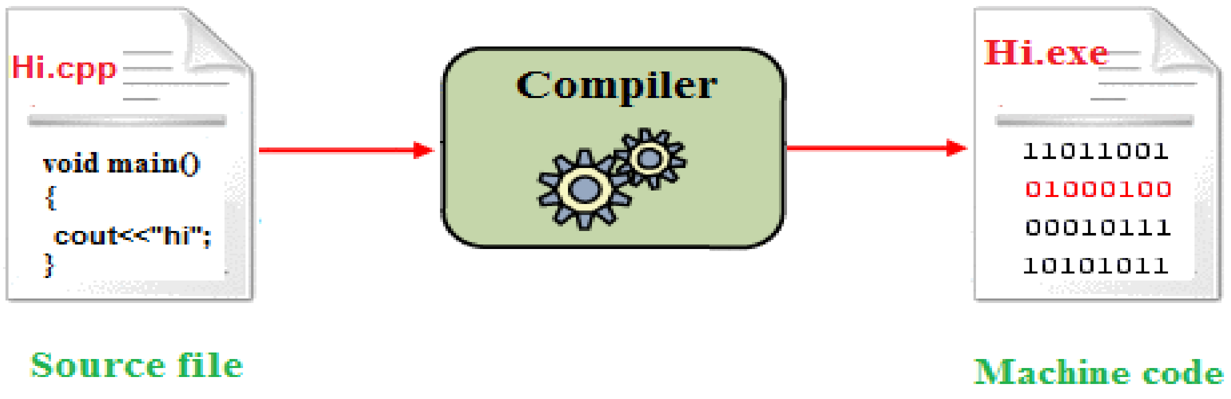
SAVEETHA SCHOOL OF ENGINEERING

SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES

DEPARTMENT
OF
COMPUTER SCIENCE & ENGINEERING

CSA14 - COMPILER DESIGN

LAB RECORD



```
void main()
{
 cout<<"hi";
}
```

```
11011001
01000100
00010111
10101011
```

SAVEETHA
INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES
(Declared as Deemed to be University under Section 3 of UGC Act 1956)

Saveetha Nagar, Thandalam, Chennai - 602 105, Tamil Nadu, India. Phone : +91 44 6672 6672
Website : www.saveetha.com,
Email : principal.sse@saveetha.com

CSE DEPARTMENT VISION-MISSION

Vision of the Department

To establish an environment to provide quality education and Inculcate research attributes among computer science engineering graduates through problem-solving skills and technological innovations.

Mission of the Department

M1

To create and sustain an academic environment to the highest level in teaching and research by enhancing the knowledge of the faculties and students in technological advancements to solve real-time problems.

M2

Providing a suitable environment for the students to develop professionalism with knowledge in Computer Science & Engineering to meet the contemporary industry needs and satisfy global standards.

M3

To facilitate the development of professional behavior and stronger ethical values so as to work with commitment for the progress of the nation and face challenges with ethical and social responsibility.

PO and PSO mapped in the Course:

PO 1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO 2. Problem analysis: Identify, formulate, review research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO 10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

Course Outcomes (CO)

CO	TITLE	PO
CO1	Discuss the major phases of compilers	PO1
CO2	Develop parsers and implement different parsers without automated tools	PO2
CO3	Apply syntax-directed translation method using synthesized and inherited attributes to generate intermediate code	PO3
CO4	Demonstrate intermediate code generation in the form of three address code representations	PO4
CO5	Interpret knowledge on back end of the compiler - Code Generator and DAG	PO5
CO6	Construct various code optimization techniques and a simple code generator	PSO1

INSTRUCTIONS FOR THE EXPERIMENTS

1. Students are advised to come to the laboratory on time; those who come after 5 minutes will not be allowed into the lab.

2. Plan your task properly before to the commencement, come prepared to the lab with the synopsis/program /
experiment details. Students should enter the laboratory with:

    - Laboratory observation notes with all the details (Problem statement, Aim, Algorithm, Procedure, Program, Expected Output, etc.,) filled in for the lab session.
    - Laboratory Records updated up to the last session experiments and other utensils (if any) needed in the lab.
    - Proper Dress code and identity card.
    - Sign in to the laboratory login register, write the TIME-IN, and occupy the computer system allotted to you by the faculty.
    - Execute your task in the laboratory, record the results/output in the lab observation notebook, and get certified by the concerned faculty.
    - All the students should be polite and cooperative with the laboratory staff and must main discipline and decency in the laboratory.
    - Computer labs are established with sophisticated and high-end branded systems, which should be utilized properly.
    - Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions. Misuse of the equipment, misbehaviors with the staff and systems, etc., will attract severe punishment.
    - Students must take the permission of the faculty in case of any urgency to go out; if anybody is found loitering outside the lab/class without permission during working hours will be treated seriously and punished appropriately.
    - Students should LOG OFF/ SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system/seat is kept properly.

DO'S AND DON'TS

- ✓ Be on time to lab and maintain silence.
- ✓ Inform the instructor /TA in case of any working environment problems.
- ✓ Be aware of all the safety devices. Even though the instructor and TA will take care of emergencies.
- ✓ Bring all the necessary things required to do laboratory experiments like an observation notebook, record notebook, and any others.
- ✓ Enter the system number, in-time, and out-time register number, name, and signature in the students' log register.
- ✓ Arrange the seating chairs and system accessories in place at the end of the session.
- ✓ Shut down the system properly and switch off the power switch at the end of the session.
- ✓ Keep your bags in front of the labs empty space.
- ✓ Do not eat, drink, chew gum, smoke or apply cosmetics in the lab.
- ✓ Do not unplug/plug any wires in the system connectivity.
- ✓ Do not use or charge mobile phones or any electronic gadgets inside the lab.
- ✓ Not to troubleshoot by yourself without knowledge of the instructor/TA.
- ✓ Do not open any unnecessary applications in the system.
- ✓ Mobile phones are prohibited inside the lab.
- ✓ Do not share a system to do experiments.

Students strictly follow all the above instructions

SAVEETHA SCHOOL OF ENGINEERING
SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
CSA14 COMPILER DESIGN

CONTENTS

13		Write a LEX Program for the Following (a) To Validate the Date of Birth. (b) To validate the Mobile number.			
14		Implement the following programs using LEX. (a)Write a LEX program to print all the constants in the given C source program file. (b)Write a LEX program that adds line numbers to the given C program file and displays the same in the standard output.			
15		Implement the following programs using LEX. (a)To count the number of Macros defined and header files included in the C program. (b)To print all HTML tags in the input.			
16		Write a LEX Program (a) To Count the Number of Vowels and Consonants in the given input. (b) To count substrings from Lowercase to Uppercase for the given input.			
17		Write a LEX program to count the number of comment lines in a given C program, eliminate them, and write them into another file.			
18		To implement Lexical Analyzer using LEX or FLEX (Fast Lexical Analyzer). The program should separate the tokens in the given C program and display them with the appropriate caption.			
19		Implement the following in LEX Program (a) To check whether the given Email ID is Valid or Not. (b) To check Whether the given URL is Valid or Not.			
20		Write a LEX program to identify and count positive and negative numbers.			
		ADDITIONAL EXPERIMENT			
21		Create YACC (or BISON) and LEX specification files to implement a basic calculator which accepts variables and constants of integer and float type.			

Exp. No.1	**SYMBOL TABLE OPERATIONS**
Date:	

Experiment 1: Implement a C program to perform symbol table operations.

<u>Aim</u>:

<u>Algorithm:</u>

Program:

```
#include<stdio.h>
#include<conio.h>
#include<malloc.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;
int 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();
}
}
```

Sample Output:

```
SYMBOL TABLE IMPLEMENTATION

INSERT
DISPLAY
DELETE
SEARCH
MODIFY
END

Enter your option : 1
Enter the label : star
Enter the symbol : *
Enter the address : 50
Label inserted

SYMBOL TABLE IMPLEMENTATION

INSERT
DISPLAY
DELETE
SEARCH
MODIFY
END
Enter your option : 2

LABEL        SYMBOL      ADDRESS
star            *            50

SYMBOL TABLE IMPLEMENTATION

INSERT
DISPLAY
DELETE
SEARCH
MODIFY
END

Enter your option : 3

Enter the label to be deleted: starAfter Deletion:

LABEL        SYMBOL      ADDRESS

SYMBOL TABLE IMPLEMENTATION
INSERT
DISPLAY
```

```
DELETE
SEARCH
MODIFY
END

Enter your option : 2

LABEL          SYMBOL      ADDRESS

SYMBOL TABLE IMPLEMENTATION

INSERT
DISPLAY
DELETE
SEARCH
MODIFY
END


Enter your option : 1 Enter the label : plus
Enter the symbol : - Enter the address : 100Label inserted

SYMBOL TABLE IMPLEMENTATION

INSERT
DISPLAY
DELETE
SEARCH
MODIFY
END

Enter your option : 4
Enter the label to be searched : plus
Search Result:
The label is present in the symbol table

SYMBOL TABLE IMPLEMENTATION

INSERT
DISPLAY
DELETE
SEARCH
MODIFY
END

Enter your option : 5
What do you want to modify?1.Only the label
Only the address
```

```
Both the label and addressEnter your choice : 1

Enter the old label : star

Enter the new label : star_new
After Modification:

LABEL          SYMBOL       ADDRESS
star_new            *                 300
SYMBOL TABLE IMPLEMENTATION
INSERT
DISPLAY
DELETE
SEARCH
MODIFY
END

Enter your option : 6
```

Result:

Exp. No.2	**IDENTIFY IDENTIFIERS, CONSTANTS, AND OPERATORS**
Date:	

Experiment 2 :The lexical analyzer should ignore redundant spaces, tabs, and new lines. It should also ignore comments. Although the syntax specification states that identifiers can be arbitrarily long, you may restrict the length to some reasonable value.

Aim:

Algorithm:

Program:

```
#include<string.h>
#include<ctype.h>
#include<stdio.h>
void keyword(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("st
atic",str)==0||strcmp("switch",str)==0||strcmp("case",str)==0)
printf("\n%s is a keyword",str);
else
printf("\n%s is an identifier",str);
}

main()
{
FILE *f1,*f2,*f3;
 char c, str[10], st1[10];
int num[100], lineno=0, tokenvalue=0,i=0,j=0,k=0;
printf("\n Enter the c program : ");
/*gets(st1);*/
 f1=fopen("input","w");
while((c=getchar())!=EOF)putc(c,f1);
fclose(f1);
f1=fopen("input","r");
f2=fopen("identifier","w");

f3=fopen("specialchar","w");
while((c=getc(f1))!=EOF)
{
if(isdigit(c))
{
tokenvalue=c-'0';c=getc(f1);
while(isdigit(c))
{
tokenvalue*=10+c-'0';
c=getc(f1);
}
num[i++]=tokenvalue;
ungetc(c,f1);
}
else if(isalpha(c))
{
putc(c,f2); c=getc(f1);
while(isdigit(c)||isalpha(c)||c=='_'||c=='$')
{
putc(c,f2);
c=getc(f1);
}
putc(' ',f2);
ungetc(c,f1);
}
else
if(c==' '||c=='\t')
printf(" ");
else if(c=='\n')lineno++;
else
putc(c,f3);
}
fclose(f2);
fclose(f3);
fclose(f1);
printf("\n The no's in the program are :");
for(j=0; j<i; j++)
printf("%d", num[j]);
printf("\n");
 f2=fopen("identifier", "r");
k=0;
printf("The keywords and identifiers are:");
while((c=getc(f2))!=EOF)
{
if(c!=' ') str[k++]=c;
else
{
str[k]='\0';
keyword(str);
 k=0;
}
}
fclose(f2);
f3=fopen("specialchar","r");
printf("\n Special characters are : ");
while((c=getc(f3))!=EOF)
printf("%c",c);
printf("\n");
fclose(f3);
printf("Total no. of lines are:%d", lineno);
}
```

Sample Input & Output:

```
Enter Program  Ctrl+D for termination:
{
int a[3],t1,t2;
t1=2;  a[0]=1; a[1]=2; a[t1]=3;
t2=-(a[2]+t1*6)/(a[2]-t1);
if t2>5 then print(t2); else {int t3; t3=99; t2=-25;
print(-t1+t2*t3);  /* this is a comment   on 2 lines */
}  endif
}

Output:
Variables : a[3] t1 t2 t3  Operator : - + * / >Constants :  2 1 3 6 5 99 -25
Keywords : int if then else endif  Special Symbols : , ;( ) { }
Comments :  this is a comment on 2 lines
```

Result:

Exp. No.3(a)	**LEFT RECURSION**
Date:	

Experiment 3(a) :Implement a C program to eliminate left recursion from a given CFG.

Aim:

Algorithm:

Program:

```
#include<stdio.h>
 #include<string.h>
#define SIZE 10
 int main () {
char non_terminal;
char  beta,alpha;
 int num;
char production[10][SIZE];
int index=3;
/* starting of the string following "->" */
printf("Enter Number of Production : ");
scanf("%d",&num);
printf("Enter the grammar as E->E-A :\n");
for(int i=0;i<num;i++)
{
scanf("%s",production[i]);
}
for(int i=0;i<num;i++){
printf("\nGRAMMAR : : : %s",production[i]);
non_terminal=production[i][0];
if(non_terminal==production[i][index]) {
alpha=production[i][index+1];
printf(" is left recursive.\n");
while(production[i][index]!=0 && production[i][index]!='|')index++;
if(production[i][index]!=0) { beta=production[i][index+1];
printf("Grammar without left recursion:\n");
printf("%c->%c%c\'",non_terminal,beta,non_terminal);
printf("\n%c\'->%c%c\'|E\n",non_terminal,alpha,non_terminal);
}
else
printf(" can't be reduced\n");
}
else
printf(" is not left recursive.\n");index=3;
}
}
```

Sample Output:

```
Enter Number of Production : 4Enter the grammar as E->E-T :E->ET|T
S->ST|aT->a

GRAMMAR : : : E->ET|T is left recursive.Grammar without left recursion:
E->TE' E'->TE'|E

GRAMMAR : : : S->ST|a is left recursive.Grammar without left recursion:
S->aS' S'->TS'|E

GRAMMAR : : : T->a is not left recursive.
```

Result:

Exp. No.3(b)	**LEFT FACTORING**
Date:	

Experiment 3 (b): Implement a C program to eliminate left factoring from a given CFG.

Aim:

Algorithm:

Program:

```
#include<stdio.h>
 #include<string.h>
int main()
{
char gram[20],part1[20],part2[20],modifiedGram[20],newGram[20],tempGram[20];
int i,j=0,k=0,l=0,pos;
printf("Enter Production : A->");
gets(gram);
for(i=0;gram[i]!='|';i++,j++)
part1[j]=gram[i];
part1[j]='\0';
for(j=++i,i=0;gram[j]!='\0';j++,i++)
part2[i]=gram[j];
part2[i]='\0';
 for(i=0;i<strlen(part1)||i<strlen(part2);i++)
{
if(part1[i]==part2[i])
{
modifiedGram[k]=part1[i];k++;
pos=i+1;
}
}
for(i=pos,j=0;part1[i]!='\0';i++,j++)
{newGram[j]=part1[i];
}
newGram[j++]='|';
for(i=pos;part2[i]!='\0';i++,j++){
newGram[j]=part2[i];
}
modifiedGram[k]='X';
modifiedGram[++k]='\0';
newGram[j]='\0';
printf("\n A->%s",modifiedGram);
printf("\n X->%s\n",newGram);
}
```

Sample Output:

```
Enter Production : A->abC|abD

A->abX X->C|D
```

Result:

Exp. No.4	**COUNT THE NUMBER OF CHARACTERS, WORDS, AND LINES**
Date:	

Experiment 4 : Write a C program for implementing a Lexical Analyzer to Scan and Count the number of characters, words, and lines in a file.

Aim:

Algorithm:

Program:

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

int main() {
    FILE *file;
    char path[100];
    char ch;
    int characters, words, lines;

    /* Input path of files to merge to third file */
    printf("Enter source file path: ");
    scanf("%s", path);

    /* Open source files in 'r' mode */
    file = fopen(path, "r");

    /* Check if file opened successfully */
    if (file == NULL) {
        printf("\nUnable to open file.\n");
        printf("Please check if file exists and you have read privilege.\n");
        exit(EXIT_FAILURE);
    }

    /*
     * Logic to count characters, words and lines.
     */
    characters = words = lines = 0;
    while ((ch = fgetc(file)) != EOF) {
        characters++;

        /* Check new line */
        if (ch == '\n' || ch == '\0')
            lines++;

        /* Check words */
        if (ch == ' ' || ch == '\t' || ch == '\n' || ch == '\0')
            words++;
    }

    /* Increment words and lines for last word */
    if (characters > 0) {
        words++;
        lines++;
    }
    /* Print file statistics */
    printf("\n");
    printf("Total characters = %d\n", characters);
    printf("Total words      = %d\n", words);
    printf("Total lines      = %d\n", lines);

    /* Close files to release resources */
    fclose(file);

    return 0;
}
```

Input file :

Compiler Design uses c.
Saveetha School of Engineering.

Sample Output:

```
Total characters =54
Total words =8
Total lines  =2
```

Result:

Exp. No.5(a)	**COMPUTATION OF FIRST**
Date:	

Experiment 5(a) : Write a C program to find FIRST for the predictiveparser.

Aim:

Algorithm:

Program:

```
#include<stdio.h>
#include<ctype.h>
void FIRST(char[],char );
void addToResultSet(char[],char);
int numOfProductions;
char productionSet[10][10];
int main()
{
int i;
char choice;
char c;
char result[20];
printf("How many number of productions ? :");
scanf(" %d",&numOfProductions);
for(i=0;i<numOfProductions;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 resultArray.
*/
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)))
{
addToResultSet(Result,c);return ;
}
//If X is non terminal
//Read each production
for(i=0;i<numOfProductions;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]=='$') addToResultSet(Result,'$');
//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++)
addToResultSet(Result,subResult[k]);
for(k=0;subResult[k]!='\0';k++)
if(subResult[k]=='$')
{
foundEpsilon=1;break;
}
//No ε found, no need to check next elementi
if(!foundEpsilon)
break;j++;
}
}
}
}
return ;
}
/* addToResultSet adds the computed
*element to resultset.
*This code avoids multiple inclusion of elements
*/
void addToResultSet(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';
}
```

Sample Output:

```
How many number of productions? :7
Enter productions Number 1 : E=TD
Enter productions Number 2 : D=+TD
Enter productions Number 3 : D=$
Enter productions Number 4 : T=FS
Enter productions Number 5 : S=*FS
Enter productions Number 6 : S=$
Enter productions Number 7 : F=a

Find the FIRST of:   E FIRST(E)= {    a  }
press 'y' to continue : y

Find the FIRST of  :DFIRST(D)= {  +  $ }
press 'y' to continue : y

Find the FIRST of  :TFIRST(T)= { a   }
press 'y' to continue : n
```

Result:

Exp. No.5(b)	**COMPUTATION OF FOLLOW**
Date:	

Experiment 5 (b): Write a C program to find FOLLOW set for predictive parser.

Aim:

Algorithm:

Program:

```
 #include<stdio.h>
 #include<ctype.h>
#include<string.h>
int limit, x = 0;
char production[10][10], array[10];

void find_first(char ch);
void find_follow(char ch);
void Array_Manipulation(char ch);

int main()
{
int count;
char option, ch;
printf("\nEnter Total Number of Productions:\t");
scanf("%d", &limit);
for(count = 0; count < limit; count++)
{
printf("\nValue of Production Number [%d]:\t", count + 1);
scanf("%s", production[count]);
}
do
{
x = 0;
printf("\nEnter production Value to Find Follow:\t");
scanf(" %c", &ch);
find_follow(ch);
printf("\nFollow Value of %c:\t{ ", ch);
for(count = 0; count < x; count++)
{
printf("%c ", array[count]);
}
printf("}\n");
printf("To Continue, Press Y:\t");
scanf(" %c", &option);
}while(option == 'y' || option == 'Y');return 0;
}

void find_follow(char ch)
{
int i, j;
int length = strlen(production[i]);i
f(production[0][0] == ch)
{
Array_Manipulation('$');
}
```

```
for(i = 0; i < limit; i++)
{
for(j = 2; j < length; j++)
{
if(production[i][j] == ch)
{
if(production[i][j + 1] != '\0')
{
find_first(production[i][j + 1]);
}
if(production[i][j + 1] == '\0' && ch != production[i][0])
{
find_follow(production[i][0]);
}
}
}
}
}

void find_first(char ch)
{
int i, k; if(!(isupper(ch)))
{
Array_Manipulation(ch);
}
for(k = 0; k < limit; k++)
{
if(production[k][0] == ch)
{
if(production[k][2] == '$')
{
find_follow(production[i][0]);
}
else if(islower(production[k][2]))
{
Array_Manipulation(production[k][2]);
}
else
{
find_first(production[k][2]);
}
}
}
}

void Array_Manipulation(char ch)
{
int count;
```

```
for(count = 0; count <= x; count++)
{
if(array[count] == ch)
{
return;
}
}
array[x++] = ch;
}
```

Sample Output:

```
Enter Total Number of Productions:  7
Value of Production Number [1]: E=TD
Value of Production Number [2]: D=+TD
Value of Production Number [3]: D=$
Value of Production Number [4]: T=FS
Value of Production Number [5]: S=*FS
Value of Production Number [6]: S=$
Value of Production Number [7]: F=(E)
Enter production Value to Find Follow:  E

Follow Value of E:     { $ ) }To Continue, Press Y: y

Enter production Value to Find Follow:  D
Follow Value of D:     { ) }
To Continue, Press Y:  n
```

Result:

Exp. No.6(a)	**COMMENT OR NOT**
Date:	

Experiment 6 (a): Write a C program to identify whether a given line is a comment or not

Aim:

Algorithm:

Program:

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

Sample input :

//compiler design

Sample Output:

It is a comment

Result:

Exp. No.6(b)	**IDENTIFIER VALID OR NOT**
Date:	

Experiment 6(b): Write a C program to test whether a given identifier is valid or not.

Aim:

Algorithm:

Program:
```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
void main()
{
char a[10];
int flag, i=1;
clrscr();
printf("\n Enter an identifier:");
gets(a);
if(isalpha(a[0]))
flag=1;else
printf("\n Not a valid identifier");
while(a[i]!='\0')
{
if(!isdigit(a[i])&&!isalpha(a[i]))
{
flag=0;break;
} i++;
}
if(flag==1)
printf("\n Valid identifier");
getch();
}
```

Sample Input & Output:

Input: Enter an identifier: first

Output:
```
Valid identifier
Enter an identifier:0xyz
Not a valid identifier
```

Result:

Exp. No.7	**PREDICTIVE PARSER**
Date:	

Experiment 7: Write a C program for constructing of LL (1) parsing.

Aim:

Algorithm:

Program:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
char s[20],stack[20];
int main()
{
char m[5][6][3]={"tb"," "," ","tb"," "," "," ","+tb"," "," ","n","n","fc"," "," ","fc"," "," ","","n","*fc"," a
","n","n","i"," "," ","(e)"," "," "};
int size[5][6]={2,0,0,2,0,0,0,3,0,0,1,1,2,0,0,2,0,0,0,1,3,0,1,1,1,0,0,3,0,0};
int i,j,k,n,str1,str2;
// clrscr();
printf("\n Enter the input string: ");
scanf("%s",s);
strcat(s,"$");
 n=strlen(s);
stack[0]='$';
            stack[1]='e';
i=1;
j=0;
printf("\nStack Input\n");
printf("\n");
while((stack[i]!='$')&&(s[j]!='$'))
{
if(stack[i]==s[j])
{
i--; j++;
}
switch(stack[i])
{
case 'e':
str1=0;
break;
 case 'b':
str1=1;
break;
 case 't':
str1=2;
break;
case 'c':
str1=3;
break;
case 'f':
str1=4;
break;

}
switch(s[j])
{
 case 'i':
str2=0;
 break;
case '+':
str2=1;
break;
case '*':
 str2=2;
break;
case '(':
str2=3;
break;
case ')':
str2=4;
break;
case '$':
str2=5;
break;

}
if(m[str1][str2][0]=='\0')
{
printf("\nERROR");
exit(0);
}
else if(m[str1][str2][0]=='n')
i--;
else if(m[str1][str2][0]=='i')
stack[i]='i';
else
{
for(k=size[str1][str2]-1;k>=0;k--)
{
stack[i]=m[str1][str2][k];
i++;
}
i--;
}
for(k=0;k<=i;k++)
printf(" %c",stack[k]);
printf(" ");
for(k=j;k<=n;k++)
printf("%c",s[k]);
printf(" \n ");
}
printf("\n SUCCESS");
getch();
```

}

Sample Output:

```
Enter the input string: a*b+c
Stack Input
        ____________________
$ b t     a*b+c$
$ b c f  a*b+c$
$ b c i  a*b+c$
$ b c f *           *b+c$
$ b c    b+c$
$ b       +c$
$ b t +  +$
$ b c f  c$
$ b c b c$
$ b $ SUCCESS
```

Result:

Exp. No.8	**CONSTRUCT RECURSIVE DESCENT PARSING**
Date:	

Experiment 8: Write a C program to construct recursive descent parsing.

Aim:

Algorithm:

Program:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
char  input[100];
 int i,l;
void main()
{
printf("\nRecursive descent parsing for the following grammar\n");
 printf("\nE->TE'\nE'->+TE'/@\nT->FT'\nT'->*FT'/@\nF->(E)/ID\n");
 printf("\nEnter the string to be checked:");
gets(input);
if(E())
{
if(input[i+1]=='\0')
 printf("\nString is accepted");
else
printf("\nString is not accepted");
}
else
printf("\nString not accepted");
getch();
} E()
{ if(T())
{ if(EP())
return(1);
else
return(0);
}
else
return(0);
}
EP()
{
if(input[i]=='+')
{
i++; if(T())
{ if(EP())
return(1);
else
return(0);
}
else
return(0);
}
else
return(1);
```

```
}
T()
{
 if(F())
{
if(TP())
return(1);
else
 return(0);
}
else
return(0);
} TP()
{
if(input[i]=='*')
{
 i++;
if(F())
{ if(TP())
return(1);
else
return(0);
}
else
return(0);
}
Else
 return(1);
} F()
{
if(input[i]=='(')
{
i++;
if(E())
{
if(input[i]==')')
{ i++;
return(1);
}
Else
 return(0);
}
Else
 return(0);
}
else
if(input[i]>='a'&&input[i]<='z'||input[i]>='A'&&input[i]<='Z')
{ i++;
return(1);
}
Else
 return(0);
}
```

Sample Output:

```
Recursive descent parsing for the following grammar

S->AS'
S'->+AS'/@
A->CA'
A'->*CA'/@
C->(S)/ID

Enter the string to be checked: (x+y)*zString is accepted
Enter the string to be checked: x/y+zString is not accepted
```

Result:

Exp. No.9	**STACK IMPLEMENTATION OF SHIFT REDUCE PARSER**
Date:	

Experiment 9: Write a C program for stack implementation of the Shift Reduce parser.

Aim:

Algorithm:

Program:

```
#include<stdio.h>
 #include<stdlib.h>
#include<conio.h>
#include<string.h>
char ip_sym[15],stack[15];
int ip_ptr=0,st_ptr=0,len,i;
char temp[2],temp2[2];
char act[15];
void check();
int main()
{
//clrscr();
printf("\n\t\t SHIFT REDUCE PARSER\n");
printf("\n GRAMMER\n");
printf("\n E->E+E\n E->E/E");
 printf("\n E->E*E\n E->a/b");
printf("\n enter the input symbol:\t");
gets(ip_sym);
printf("\n\t stack implementation table");
 printf("\n stack \t\t input symbol\t\t action");
printf("\n       _\t\t_____\t\t____\n");
printf("\n $\t\t%s$\t\t\t--",ip_sym);
strcpy(act,"shift ");
temp[0]=ip_sym[ip_ptr];
temp[1]='\0';
strcat(act,temp);
len=strlen(ip_sym);
for(i=0;i<=len-1;i++)
{
stack[st_ptr]=ip_sym[ip_ptr];
stack[st_ptr+1]='\0';
ip_sym[ip_ptr]=' ';
 ip_ptr++;
printf("\n $%s\t\t%s$\t\t\t%s",stack,ip_sym,act);
strcpy(act,"shift");
temp[0]=ip_sym[ip_ptr];
temp[1]='\0';
 strcat(act,temp);
check();
st_ptr++;
}
st_ptr++;
 check();
}
void check()
{
```

```
int flag=0; temp2[0]=stack[st_ptr];
temp2[1]='\0';
if((!strcmpi(temp2,"a"))||(!strcmpi(temp2,"b")))
{
stack[st_ptr]='E';
if(!strcmpi(temp2,"a"))
printf("\n $%s\t\t%s$\t\t\tE->a",stack,ip_sym);
else
printf("\n $%s\t\t%s$\t\t\tE->b",stack,ip_sym);
flag=1;
}
if((!strcmpi(temp2,"+"))||(strcmpi(temp2,"*"))||(!strcmpi(temp2,"/")))
{
flag=1;
}
if((!strcmpi(stack,"E+E"))||(!strcmpi(stack,"E\E"))||(!strcmpi(stack,"E*E")))
{
strcpy(stack,"E");
 st_ptr=0; if(!strcmpi(stack,"E+E"))
printf("\n $%s\t\t%s$\t\t\tE->E+E",stack,ip_sym);
else
if(!strcmpi(stack,"E\E"))
printf("\n $%s\t\t%s$\t\t\tE->E\E",stack,ip_sym);
else
if(!strcmpi(stack,"E*E"))
printf("\n $%s\t\t%s$\t\t\tE->E*E",stack,ip_sym);
else
printf("\n $%s\t\t%s$\t\t\tE->E+E",stack,ip_sym);flag=1;
}
if(!strcmpi(stack,"E")&&ip_ptr==len)
{
printf("\n $%s\t\t%s$\t\t\tACCEPT",stack,ip_sym);
getch();
exit(0);
}
if(flag==0)
{
printf("\n%s\t\t\t%s\t\t reject",stack,ip_sym);
exit(0);
}
return;
}
```

Sample Output:

```
SHIFT REDUCE PARSER

GRAMMERE->E+E
E->E/E->E*E
E->(E)
E->id

enter the input symbol:        id*id

stack implementation table

stack          input symbol          action
 _____        ____________        ____________

$              id*id$                 --
$a             *id$                   shift aid
$E             *id$                   E->id
$E*            id$                    shift *
$E*id          $                      shift id
$E*E           $                      E->id
$E             $                      E->E*E
$E             $                      ACCEPT
```

Result:

Exp. No.10	**OPERATOR PRECEDENCE PARSING**
Date:	

Experiment 10: Write a C program to implement operator precedence parsing.

Aim:

Algorithm:

Program:

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

char *input;int i=0;
char lasthandle[6],stack[50],handles[][5]={")E(","E*E","E+E","i","E^E"};
//(E) becomes )E( when pushed to stack

int top=0,l;
char prec[9][9]={

/*input*/

/*stack +       -   *  /   ^   i   (   )   $  */

/*  + */  '>', '>','<','<','<','<','<','>','>',

/*  - */  '>', '>','<','<','<','<','<','>','>',

/*  * */  '>', '>','>','>','<','<','<','>','>',

/*  / */  '>', '>','>','>','<','<','<','>','>',

/*  ^ */  '>', '>','>','>','<','<','<','>','>',

/*  i */  '>', '>','>','>','>','e','e','>','>',

/*  ( */  '<', '<','<','<','<','<','<','>','e',

/*  ) */  '>', '>','>','>','>','e','e','>','>',

/*  $ */  '<', '<','<','<','<','<','<','<','>',
};

int getindex(char c)
{
switch(c)
{
case '+':
return 0;
case '-':
return 1;
case '*':
return 2;
case '/':
return 3;
case '^':
```

```
return 4;
case 'i':
return 5;
case '(':
return 6;
case ')':
return 7;
case '$':
return 8;
}
}


int shift()
{
stack[++top]=*(input++);
stack[top+1]='\0';
}


int reduce()
{
int i,len,found,t;
for(i=0;i<5;i++)//selecting handles
{
len=strlen(handles[i]);
if(stack[top]==handles[i][0]&&top+1>=len)
{
found=1;
for(t=0;t<len;t++)
{
if(stack[top-t]!=handles[i][t])
{
found=0;
break;
}
}
if(found==1)
{
stack[top-t+1]='E';
top=top-t+1;
strcpy(lasthandle,handles[i]);
stack[top+1]='\0';
return 1;
//successful reduction
}
}
}
```

```
return 0;
}

void dispstack()
{
int j; for(j=0;j<=top;j++)
printf("%c",stack[j]);
}

void dispinput()
{
int j; for(j=i;j<l;j++)
printf("%c",*(input+j));
}

void main()
{
int j;

input=(char*)malloc(50*sizeof(char));
printf("\nEnter the string\n");
 scanf("%s",input);
input=strcat(input,"$");
 l=strlen(input);
strcpy(stack,"$");
printf("\nSTACK\tINPUT\tACTION");
while(i<=l)
{
shift();
printf("\n");
dispstack();
printf("\t");
dispinput();
printf("\tShift");
if(prec[getindex(stack[top])][getindex(input[i])]=='>')
{
while(reduce())
{
printf("\n");
dispstack();
printf("\t");
dispinput();
printf("\tReduced: E->%s",lasthandle);
}
}
}

if(strcmp(stack,"$E$")==0)
```

```
printf("\nAccepted;");
else
printf("\nNot Accepted;");
}
```

Sample Output:

```
Enter the stringi*(i+i)*i

STACK         INPUTACTION
$i       *(i+i)*i$       Shift
$E       *(i+i)*i$       Reduced: E->i
$E*      (i+i)*i$        Shift
$E*(     i+i)*i$ Shift
$E*(i    +i)*i$  Shift
$E*(E    +i)*i$  Reduced: E->i
$E*(E+           i)*i$   Shift
$E*(E+i          )*i$    Shift
$E*(E+E          )*i$    Reduced: E->i
$E*(E    )*i$    Reduced: E->E+E
$E*(E)   *i$     Shift
$E*E     *i$     Reduced: E->)E(
$E       *i$     Reduced: E->E*E
$E*      i$      Shift
$E*i     $       Shift
$E*E     $       Reduced: E->i
$E       $       Reduced: E->E*E
$E$      Shift
$E$      Shift
Accepted;
```

Result:

Exp. No.11	**INTERMEDIATE CODE GENERATOR**
Date:	

Experiment 11: Implement a simple intermediate code generator in C program, which producesthree address code statements for a given input expression.

Aim:

Algorithm:

Program:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>
struct three
{
char data[10],temp[7];
}s[30];
int main()
{
char d1[7],d2[7]="t";int i=0,j=1,len=0;
FILE *f1,*f2;
//clrscr();
 f1=fopen("sum.txt","r");
f2=fopen("out.txt","w");
while(fscanf(f1,"%s",s[len].data)!=EOF)
len++;
itoa(j,d1,7);
strcat(d2,d1); s
trcpy(s[j].temp,d2);
strcpy(d1,"");
strcpy(d2,"t");
if(!strcmp(s[3].data,"+"))
{
fprintf(f2,"%s=%s+%s",s[j].temp,s[i+2].data,s[i+4].data);
j++;
}
else if(!strcmp(s[3].data,"-"))
{
fprintf(f2,"%s=%s-%s",s[j].temp,s[i+2].data,s[i+4].data);
j++;
}
for(i=4;i<len-2;i+=2)
{
itoa(j,d1,7);
 strcat(d2,d1);
strcpy(s[j].temp,d2);
if(!strcmp(s[i+1].data,"+"))
fprintf(f2,"\n%s=%s+%s",s[j].temp,s[j-1].temp,s[i+2].data);
else if(!strcmp(s[i+1].data,"-"))
fprintf(f2,"\n%s=%s-%s",s[j].temp,s[j-1].temp,s[i+2].data);
 strcpy(d1,"");
strcpy(d2,"t");
j++;
}
fprintf(f2,"\n%s=%s",s[0].data,s[j-1].temp);f
close(f1);
fclose(f2);
getch();
}
```

Sample Output:

Input: sum.txt

out = in1 + in2 + in3 - in4

Output: out.txtt1=in1+in2 t2=t1+in3
t3=t2-in4out=t3

Result:

Exp. No.12	**BACK END OF THE COMPILER**
Date:	

Experiment 12: Write a C program to implement the back end of the compiler.

Aim:

Algorithm:

Program:

```
#include <stdio.h >
#include <stdio.h >
#include <string.h >
 int main() {
   char icode[10][30], str[20], opr[10];
   int i = 0;
   printf("\n Enter the set of intermediate code (terminated by exit):\n");
   do
   {
     scanf("%s", icode[i]);
   }
         while (strcmp(icode[i++], "exit") != 0);
   printf("\n target code generation");
   printf("\n**********");
   i = 0;
   do {
     strcpy(str, icode[i]);
     switch (str[3]) {
     case '+':
       strcpy(opr, "ADD ");
       break;
     case '-':
       strcpy(opr, "SUB ");
       break;
     case '*':
       strcpy(opr, "MUL ");
       break;
     case '/':
       strcpy(opr, "DIV ");
       break;
     }
     printf("\n\tMov %c,R%d", str[2], i);
     printf("\n\t%s%c,R%d", opr, str[4], i);
     printf("\n\tMov R%d,%c", i, str[0]);
   }
         while (strcmp(icode[++i], "exit") != 0);
   return 0;
 }
```

Sample Output:

```
Enter the set of intermediate code (terminated by exit):
x=y-z
c=y+k
y=y/x
exit

Mov y,R0
SUB z,R0
Mov R0,x
Mov y,R1
ADD k,R1
Mov y,R2
DIV x,R2
Mov R2,y
```

Result:

Exp. No.13 (a)	**VALIDATE THE DATE OF BIRTH**
Date:	

Experiment 13 (a): Write a LEX Program for the validation of Date of Birth.

Aim:

Algorithm:

Program:

```
%{
 #include <stdio.h>
 int month, day, year;
 %}


 %%


 [0-9]{2}\/[0-9]{2}\/[0-9]{4} {
     sscanf(yytext, "%d/%d/%d", &day, &month, &year);
     if (month < 1 || month > 12) {
         printf("Invalid date\n");
     } else {
         int max_day = 31;
         if (month == 4 || month == 6 || month == 9 || month == 11) {
             max_day = 30;
         } else if (month == 2) {
             max_day = (year % 4 == 0 && (year % 100 != 0 || year % 400 == 0)) ? 29 :28;
         }
         if (day < 1 || day > max_day) {
             printf("Invalid date\n");
         } else {
             printf("Valid date\n");
         }
     }
 }

 .|\n


 %%
 int yywrap(){}
 int main() {
     yylex();
     return 0;
 }
```

Sample Input:

```
C:\User>flex d.l.txt
C:\User>gcc lex.yy.c
C:\User>a
28/02/2015
```

Sample Output:

```
Valid DOB
```

Result:

Exp. No.13(b)	**VALIDATE THE MOBILE NUMBER**
Date:	

Experiment 13 (b): Write a LEX Program for the validation of Mobile Number.

Aim:

Algorithm:

Program:

```
%{
 %}
 %%
 [6-9][0-9]{9} {printf("\nMobile Number Valid\n");}

 .+ {printf("\nMobile Number Invalid\n");}
 %%
 int yywrap(){}
 int main()
 {
    printf("\nEnter Mobile Number : ");
    yylex();
    printf("\n");
    return 0;
 }
```

Sample Input:

```
C:\User>flex mobile.l.txt
C:\User>gcc lex.yy.c
C:\User>a
Enter Mobile Number: 9000012345
```

Sample Output:

```
Mobile Number Valid
```

Result:

Exp. No.14(a)	**PRINT THE CONSTANTS**
Date:	

Experiment 14 (a): Write a LEX program to print all the constants in the given C source program file.

Aim:

Algorithm:

Program:

```
digit [0-9]
%{
int cons=0;
%}
%%
{digit}+ { cons++; printf("%s is a constant\n", yytext);  }
.|\n { }
%%
int yywrap(void) {
return 1; }
int main(void)
{
FILE *f;
char file[10];
printf("Enter File Name : ");
scanf("%s",file);
f = fopen(file,"r");
yyin = f;
yylex();
printf("Number of Constants : %d\n", cons);
fclose(yyin);
}
```

Sample Input:

```
#include<stdio.h>
#include<conio.h>
 void main()
{
	int i;
	for(i=0;i<10;i++)
	printf("yes");
}
```

Sample Output:

```
C:\User >flex 22.l.txt
C:\User>gcc lex.yy.c
C:\Users>a.exe boring.c
Enter File Name : boring.c
0 is a constant
10 is a constant
Number of Constants : 2
```

Result:

Exp. No.14(b)	**ADDS LINE NUMBERS TO THE GIVEN C PROGRAM FILE**
Date:	

Experiment 14(b): Write a LEX program that adds line numbers to the given C program file and displays the same in the standard output.

Aim:

Algorithm:

Program:

```
%{
int yylineno;
%}
%%
^(.*)\n printf("%4d\t%s", ++yylineno, yytext);
%%
int yywrap(void) {
return 1;
}
int main(int argc, char *argv[]) {
yyin = fopen(argv[1], "r");
yylex();
fclose(yyin);
}
```

Sample Input:

```
#include<stdio.h>
 void main()
{
	printf("Value of pi=3.14")
}
```

Sample Output:

```
C:\User>flex 25.l.txt
C:\User>gcc lex.yy.c
C:\User>a.exe 22.c
   1
   2    #include<stdio.h>
   3     void main()
   4    {
   5            printf("Value of pi=3.14")
   6    }
   7
```

Result:

Exp. No.15(a)	COUNT THE NUMBER OF MACROS DEFINED AND HEADER FILES INCLUDED IN THE C PROGRAM
Date:	

Experiment 15(a): write a LEX program to count the number of Macros defined and header files included in the C program. File.

Aim:

Algorithm:

Program:

```
%{
int nmacro,nheader;
%}
%%
^#define {nmacro++; }
^#include {nheader++; }
.|\n { }
%%
int yywrap(void)
{
return 1;
}
int main(int argc, char *argv[]) {
yyin = fopen(argv[1], "r");
yylex();
printf("Number of macros defined = %d\n", nmacro);
printf("Number of header files included = %d\n", nheader);
fclose(yyin);
}
```

Sample Input:

```
#define MIN 10
#define MAX 20
#define PI 3.14
#include<stdio.h>
void main(){
int a,b;
printf("%d",MAX);
printf("%d",MIN);
}
```

Sample Output:

```
C:\Users>flex macrosheaders.l.txt
C:\Users>gcc lex.yy.c
C:\Users>a.exe sample.c
Number of macros defined = 3
Number of header files included = 1
```

Result:

Exp. No.15(b)	PRINT ALL HTML TAGS
Date:	

Experiment 15 (b): Write a LEX program to print all HTML tags in the input file.

Aim:

Algorithm:

Program:

```
%{
int tags;
%}
%%
"<"[^>]*>  { tags++;}
.|\n {}
%%
int yywrap(void)
{
return 1;
}
int main(int argc, char *argv[]) {
yyin = fopen(argv[1],"r");
yylex();
printf("\n Number of html tags: %d",tags);
fclose(yyin);
}
```

Sample Input:

```
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
</ul>
```

Sample Output:

```
C:\User>flex html.l.txt
C:\User>gcc lex.yy.c
C:\User>a.exe sample.c
Number of html tags: 8
```

Result:

Exp. No.16(a)	**COUNT THE NUMBER OF VOWELS AND CONSONANTS**
Date:	

Experiment 16(a): Write a LEX program to Count the Number of Vowels and Consonants in the given string

Aim:

Algorithm:

Program:

```
%{
#include<stdio.h>
int v=0;
int c=0;
%}
%%
[  \t\n]+;
[aeiouAEIOU]  {v++;}
[^aeiouAEIOU]  {c++;}
%%
int main( )
{
printf ("Enter the input String :\n");
yylex();
printf("no of vowels are %d\n",v);
printf("No of consonants are %d \n",c);
}
int yywrap( )
{
return 1;
}
```

Sample Output:

```
C:\User>flex consonentsvowels.l.txt
C:\User>gcc lex.yy.c
C:\User>a
Enter the input String :
divya
no of vowels are 2
No of consonants are 3
```

Result:

Exp. No.16(b)	**LOWERCASE TO UPPERCASE**
Date:	

Experiment 16(b): Write a LEX program to convert substrings from Lowercase to Uppercase for the given input

Aim:

Algorithm:

Program:

```
%{
%}
%%
[a-z]  {printf("%c",yytext[0]-32);}
.  {}
%%
int yywrap(void){}
int main()
{
printf("\nenter the string : ");
yylex();
}
```

Sample Output:

```
C:\User>flex smalltocap.l.txt
C:\User>gcc lex.yy.c
C:\User>a
enter the string : student
STUDENT
```

Result:

Exp. No.17	**REMOVING AND COUNTING THE COMMENT LINES**
Date:	

Experiment 17: Write a LEX program to count the number of comment lines in a given C program, eliminate them, and write them into another file.

Aim:

Algorithm:

Program:

```
%{
int com = 0;
%}
%%
"/*"[^\n]+"*/" {com++; fprintf(yyout," ");}
\/\/.* {; com++; fprintf(yyout," ");}
%%
void main(int argc, char *argv[])
{
yyin=fopen(argv[1],"r");
yyout=fopen(argv[2],"w");
yylex();
printf("\n number of comments are = %d\n",com);
}
int yywrap()
{
return 1;
}
```

<u>Sample Input:</u>

```
#include<stdio.h>
int main()
"{
int a,b,c;
printf("enter two numbers:");
a=b+c;
}
}
```

<u>Sample Output:</u>

```
C:\User>flex commentsline.l.txt
C:\User>gcc lex.yy.c
C:\User>a.exe input.c
number of comments are = 2
C:\User>a.exe output.c
#include<stdio.h>
int main(){
int a,b,c;
printf("Enter two numbers : ");
scanf("%d %d",&a,&b);
c = a+b;
printf("Sum is %d",c);
return 0;
}
 number of comments are = 0
```

<u>Result</u>:

Exp. No.18	TO SEPARATE THE TOKENS IN THE GIVEN C PROGRAM AND DISPLAY
Date:	

Experiment 18 Implement Lexical Analyzer using LEX or FLEX (FastLexical Analyzer). The program should separate the tokens in the given C program and display them with the appropriate caption.

Aim:

Algorithm:

Program:

```
%{
#include <stdio.h>
%}
DIGIT   [0-9]
LETTER  [a-zA-Z]
ID      {LETTER}({LETTER}|{DIGIT})*
INT     {DIGIT}+
OP      '+'|'-'|'*'|'/'|'='
PUNCT   '('|')'|'{'|'}'|';'
%%
[ \t\n]+    ;
{ID}        { printf("ID: %s\n", yytext); }
{INT}       { printf("INT: %s\n", yytext); }
{OP}        { printf("OP: %s\n", yytext); }
{PUNCT}     { printf("PUNCT: %s\n", yytext); }
{ printf("ERROR: unrecognized token '%s'\n", yytext); }
%%
int main(int argc, char** argv)
 {
    yylex();
    return 0;
}
```

Sample Input:

```
int fibonacci(int n) {
    if (n <= 1)
       return n;
    else
        return fibonacci(n - 1) + fibonacci(n - 2);
}
```

Sample Output:

```
ID: void
ID: swap
PUNCT: (
ID: int
OP: *
ID: a
PUNCT: ,
ID: int
OP: *
ID: b
PUNCT: )
PUNCT: {
ID: int
ID: temp
OP: =
OP: *
ID: a
PUNCT: ;
OP: *
ID: a
OP: =
OP: *
ID: b
PUNCT: ;
OP: *
ID: b
OP: =
ID: temp
PUNCT: ;
PUNCT: }
```

Result:

Exp. No.19 (a)	**EMAIL ID IS VALID (OR) NOT**
Date:	

Experiment 19 (a): Implement the following in LEX Program to check whether the given Email ID is Valid or Not

Aim:

Algorithm:

PROGRAM:

```
%{
#include<stdio.h>
%}
%%
[a-zA-Z0-9_]+@[a-z]+.[a-z]+ {printf("%s is a valid email", yytext);}
.+ {printf("It is not a valid email");}
%%
int main()
{
printf("\n Enter the email:");
yylex();
}
int yywrap()
{
return 1;
}
```

Sample Output:

```
C:\User>flex email.l.txt
C:\User>gcc lex.yy.c
C:\User>a
 Enter the email:pavani09.com
It is not a valid email
```

Result:

Exp. No.19(b)	**URL IS VALID (OR) NOT**
Date:	

Experiment 19 (b): To check Whether the given URL is Valid or not

Aim:

Algorithm:

PROGRAM:

```
%{
%}
%%
((http)|(ftp))s?:\/\/[a-zA-Z0-9]{2}(\.[a-z]{2})+(\/[a-zA-Z0-9+=?]*)*{printf("\nURL InValid\n");}
.+ {printf("\nURL valid\n");}
%%
int yywrap(){}
void main()
{
	printf("\nEnter URL : ");
	yylex();
	printf("\n");
}
```

Sample input :

```
C:\User>flex url.l.txt
C:\User>gcc lex.yy.c
C:\User>a
Enter URL :www.bing.in
```

Sample Output:

```
Valid URL
```

Result:

Exp. No.20	COUNT POSITIVE AND NEGATIVE NUMBERS
Date:	

Experiment 20: Write a LEX program to identify and count positive and negative numbers

Aim:

Algorithm:

PROGRAM:

```
%{
int positive_no = 0, negative_no = 0;
%}
%%
^[-][0-9]+ {negative_no++;
printf("negative number = %s\n",
yytext);} // negative number
[0-9]+ {positive_no++;
printf("positive number = %s\n",
yytext);} // positive number
%%
int yywrap(){}
int main()
{
yylex();
printf ("number of positive numbers = %d,"
"number of negative numbers = %d\n",
positive_no, negative_no);
return 0;
}
```

Sample Output:

```
C:\User>a
-12
negative number = -12
-12
negative number = -12
-34
negative number = -34
number of positive numbers = 0, number of negative numbers = 3
```

Result:

Exp. No.21	**CREATE BASIC CALCULATOR USING YACC**
Date:	

Experiment 21: Create YACC (or BISON) and LEX specification files to implement a basic calculator which accepts variables and constants of integer and float type.

Aim:

Algorithm:

Program:

```
calc. l   (flex source program)
%{
#include <stdlib.h>
 void yyerror(char *);
 #include "calc.tab.h"
%}
%%
/* variables */ [a-z] {
yylval = *yytext - 'a';
 return VARIABLE;
}
/* integers */ [0-9]+ {
yylval = atoi(yytext);
return INTEGER;
}
/* operators */
[-+()=/*\n]      { return *yytext; }
/* skip whitespace */ [ \t]       ;
/* anything else is an error */
. yyerror("invalid character");
%%
int yywrap(void) { return 1;
}
calc.y (bison source program)
%token INTEGER VARIABLE
%left '+' '-'
%left '*' '/'
%{
#include <stdio.h> void yyerror(char *); int yylex(void);
int sym[26];
%}
%%
program:

statement:

program statement '\n'
|
;
 expr:




%%
```

```
expr { printf("%d\n", $1); }
| VARIABLE '=' expr { sym[$1] = $3; }
;

INTEGER
| VARIABLE { $$ = sym[$1]; }
| expr '+' expr { $$ = $1 + $3; }
| expr '-' expr { $$ = $1 - $3; }
| expr '*' expr { $$ = $1 * $3; }
| expr '/' expr { $$ = $1 / $3; }
| '(' expr ')' { $$ = $2; }
;
void yyerror(char *s) { fprintf(stderr, "%s\n", s);
}
int main(void) { yyparse(); return 1;
}
```

Sample Output:

```
C:\User>flex calc.l
C:\User>bison calc.y
C:\User>gcc calc.tab.c lex.yy.c -o calc.exe

C:\User>calc.exe 10*(1+2)
30
x = 9 * (2+2)
y = 2
x
36
y
2
x + 2*y
40
```

Result: