

GURU TEGH BAHADUR INSTITUTE OF  
TECHNOLOGY

COMPILER DESIGNE LAB FILE

NAME: SIDDHIKA SHUKLA

ENROLLMENT NO.: 05113203121

BRANCH: IT-1, 5<sup>TH</sup> SEM

## TITLE TABLE

S.NO.	TITLE	PAGE NO.	SIGNATURE

## PRACTICAL – 1

### Introduction to Compiler Design.

The compiler is software that converts a program written in a high-level language (Source Language) to a low-level language (Object/Target/Machine Language/0, 1's).

A translator or language processor is a program that translates an input program written in a programming language into an equivalent program in another language. The compiler is a type of translator, which takes a program written in a high-level programming language as input and translates it into an equivalent program in low-level languages such as machine language or assembly language.

The program written in a high-level language is known as a source program, and the program converted into a low-level language is known as an object (or target) program. Without compilation, no program written in a high-level language can be executed. For every programming language, we have a different compiler; however, the basic tasks performed by every compiler are the same. The process of translating the source code into machine code involves several stages, including lexical analysis, syntax analysis, semantic analysis, code generation, and optimization.

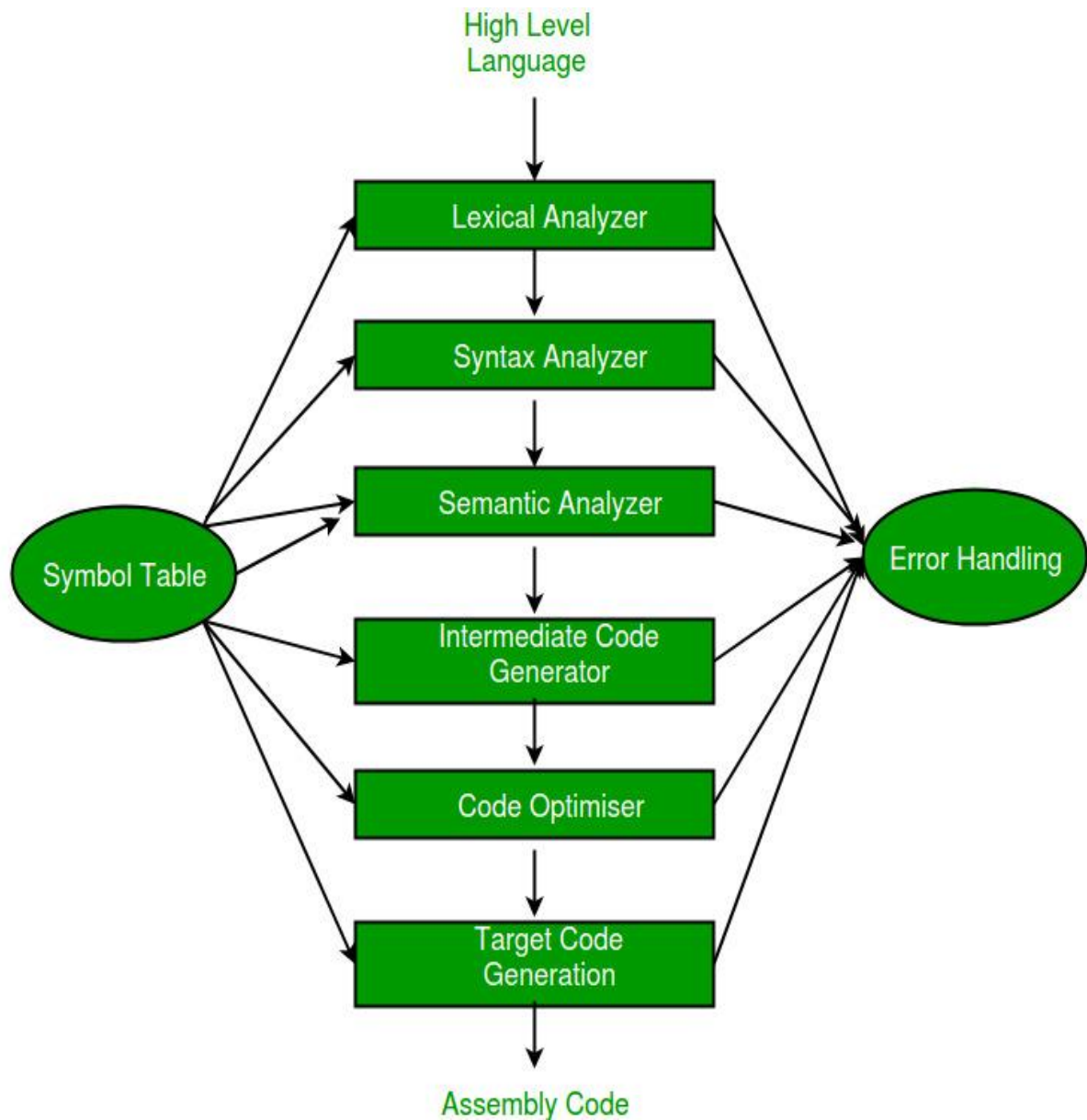
Compiler is an intelligent program as compare to an assembler. Compiler verifies all types of limits, ranges, errors, etc. Compiler program takes more time to run and it occupies huge amount of memory space. The speed of compiler is slower than other system software. It takes time because it enters through the program and then does translation of the full program. When compiler runs on same machine and produces machine code for the same machine on which it is running. Then it is called as self-compiler or resident compiler. Compiler may run on one machine and produces the machine codes for other computer then in that case it is called as cross compiler.

## Stages of Compiler Design

- **Lexical Analysis:** The first stage of compiler design is lexical analysis, also known as scanning. In this stage, the compiler reads the source code character by character and breaks it down into a series of tokens, such as keywords, identifiers, and operators. These tokens are then passed on to the next stage of the compilation process.
- **Syntax Analysis:** The second stage of compiler design is syntax analysis, also known as parsing. In this stage, the compiler checks the syntax of the source code to ensure that it conforms to the rules of the programming language. The compiler builds a parse tree, which is a hierarchical representation of the program's structure, and uses it to check for syntax errors.
- **Semantic Analysis:** The third stage of compiler design is semantic analysis. In this stage, the compiler checks the meaning of the source code to ensure that it makes sense. The compiler performs type checking, which ensures that variables are used correctly and that operations are performed on compatible data types. The compiler also checks for other semantic errors, such as undeclared variables and incorrect function calls.
- **Code Generation:** The fourth stage of compiler design is code generation. In this stage, the compiler translates the parse tree into machine code that can be executed by the computer. The code generated by the compiler must be efficient and optimized for the target platform.
- **Optimization:** The final stage of compiler design is optimization. In this stage, the compiler analyses the generated code and makes optimizations to improve its performance. The compiler may perform optimizations such as constant folding, loop unrolling, and function inlining.

Overall, compiler design is a complex process that involves multiple stages and requires a deep understanding of both the programming language and the target platform. A well-designed compiler can

greatly improve the efficiency and performance of software programs, making them more useful and valuable for users.



## PRACTICAL – 2

Write a program to check whether a string belong to the grammar or not.

### **Program:**

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main() {
    char string[50];
    int flag,count=0;
    printf("The grammar is:\n S->aS,\n S->Sb,\n S->ab\n");
    printf("Enter the string to be checked:\n");
    gets(string);
    if(string[0]=='a') {
        flag=0;
        for (count=1;string[count-1]!='\0';count++) {
            if(string[count]=='b') {
                flag=1;
                continue;
            }
            else if((flag==1)&&(string[count]=='a')) {
                printf("The string does not belong to the specified
grammar");
                break;
            }
            else if(string[count]=='a'){
```

```

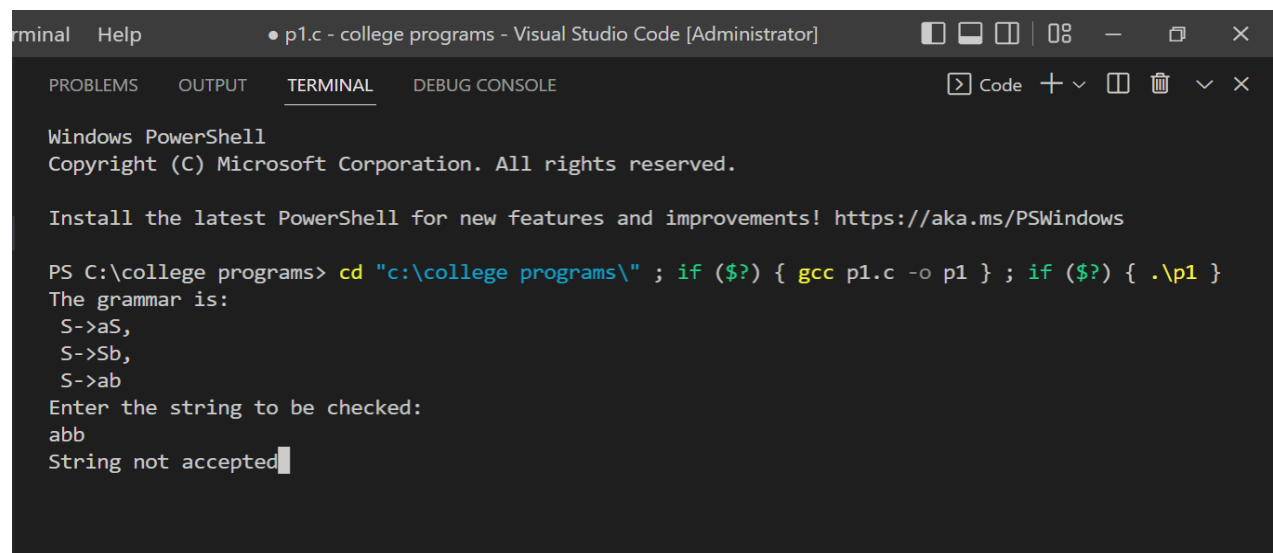
        continue;
    }
    else if((flag==1)&&(string[count]!='\0')) {
        printf("String accepted.....!!!!");
        break;
    }
    else {
        printf("String not accepted");
    }
}

}

getch();
}

```

## Output:



```

terminal  Help  • p1.c - college programs - Visual Studio Code [Administrator]
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\college programs> cd "c:\college programs\" ; if ($?) { gcc p1.c -o p1 } ; if ($?) { .\p1 }
The grammar is:
S->aS,
S->Sb,
S->ab
Enter the string to be checked:
abb
String not accepted

```

## PRACTICAL – 3

Write a program to check whether a string include keyword or not.

### **Program:**

```
#include <stdbool.h>
#include <stdio.h>
#include <string.h>
bool isKeyword(char* str)
{
    if (!strcmp(str, "auto") || !strcmp(str, "default")
        || !strcmp(str, "signed") || !strcmp(str, "enum")
        || !strcmp(str, "extern") || !strcmp(str, "for")
        || !strcmp(str, "register") || !strcmp(str, "if")
        || !strcmp(str, "else") || !strcmp(str, "int")
        || !strcmp(str, "while") || !strcmp(str, "do")
        || !strcmp(str, "break") || !strcmp(str, "continue")
        || !strcmp(str, "double") || !strcmp(str, "float")
        || !strcmp(str, "return") || !strcmp(str, "char")
        || !strcmp(str, "case") || !strcmp(str, "const")
        || !strcmp(str, "sizeof") || !strcmp(str, "long")
        || !strcmp(str, "short") || !strcmp(str, "typedef")
        || !strcmp(str, "switch")
        || !strcmp(str, "unsigned") || !strcmp(str, "void")
        || !strcmp(str, "static") || !strcmp(str, "struct")
        || !strcmp(str, "goto") || !strcmp(str, "union")
        || !strcmp(str, "volatile"))
```

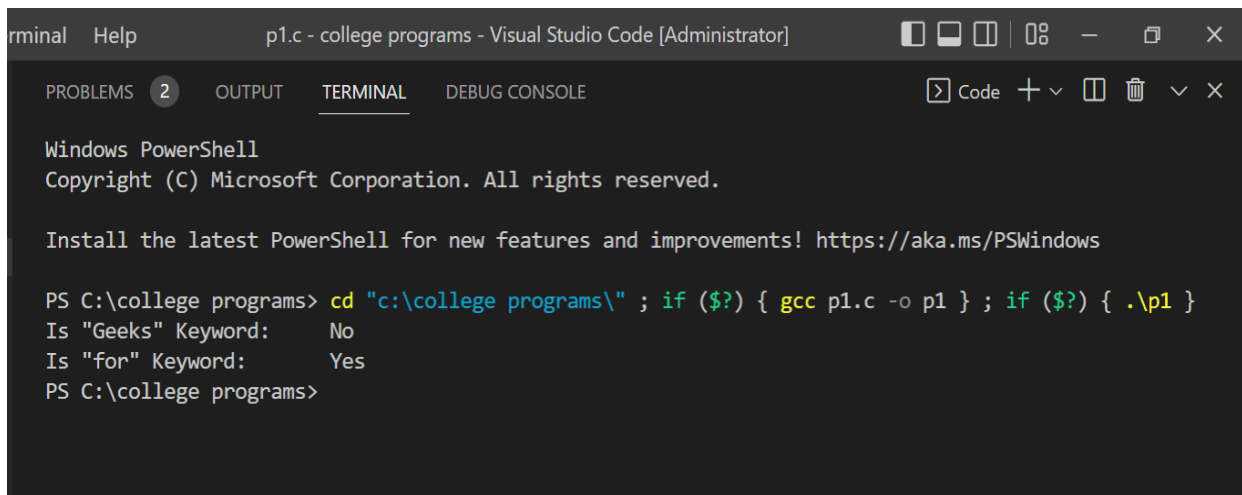


```

        return (true);
    return (false);
}
int main()
{
    printf("Is \"Geeks\" Keyword: \t");
    isKeyword("geeks") ? printf("Yes\n") : printf("No\n");
    printf("Is \"for\" Keyword: \t");
    isKeyword("for") ? printf("Yes\n") : printf("No\n");
    return 0;
}

```

## Output:



```

terminal  Help      p1.c - college programs - Visual Studio Code [Administrator]
PROBLEMS 2 OUTPUT  TERMINAL  DEBUG CONSOLE
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\college programs> cd "c:\college programs\" ; if ($?) { gcc p1.c -o p1 } ; if ($?) { .\p1 }
Is "Geeks" Keyword:      No
Is "for" Keyword:       Yes
PS C:\college programs>

```

## PRACTICAL – 4

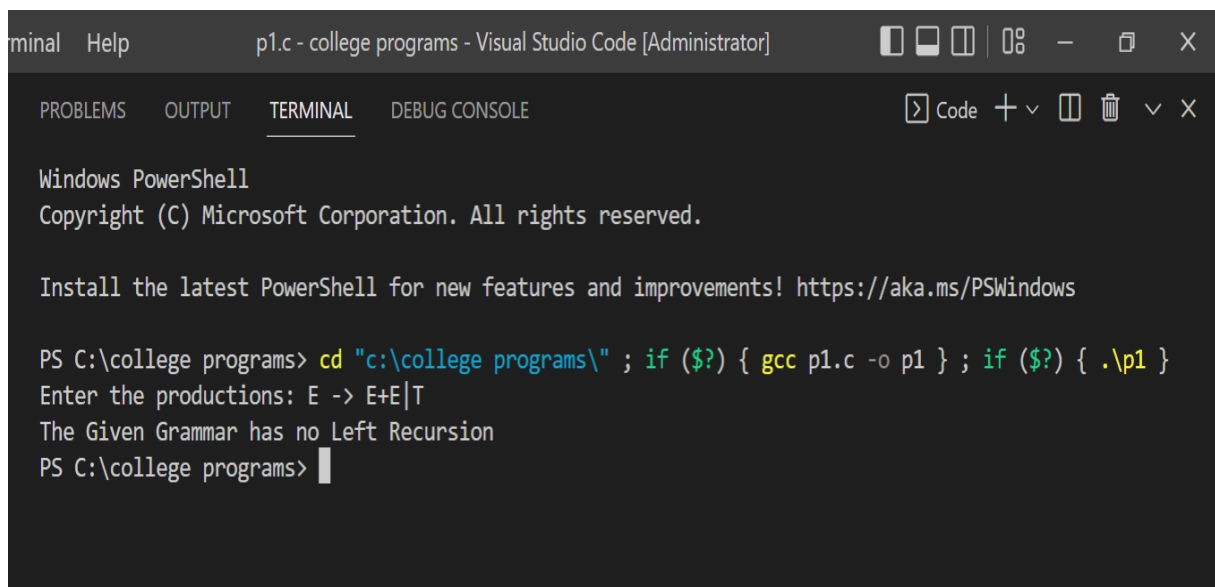
Write a program to remove left recursion from a grammar.

### **Program:**

```
#include<stdio.h>
#include<string.h>
void main() {
    char input[100],l[50],r[50],temp[10],tempprod[20],productions[25][50];
    int i=0,j=0,flag=0,consumed=0;
    printf("Enter the productions: ");
    scanf("%1s->%s",l,r);
    printf("%s",r);
    while(sscanf(r+consumed,"%^[ ]s",temp) == 1 && consumed <= strlen(r)) {
        if(temp[0] == l[0]) {
            flag = 1;
            sprintf(productions[i++],"%s->%s%s\0",l,temp+1,l);
        }
        else
            sprintf(productions[i++],"%s'->%s%s\0",l,temp,l);
        consumed += strlen(temp)+1;
    }
    if(flag == 1) {
        sprintf(productions[i++],"%s->\epsilon\0",l);
        printf("The productions after eliminating Left Recursion are:\n");
        for(j=0;j<i;j++)
            printf("%s\n",productions[j]);
    }
}
```

```
else
    printf("The Given Grammar has no Left Recursion");
}
```

## Output:



```
terminal Help p1.c - college programs - Visual Studio Code [Administrator]
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\college programs> cd "c:\college programs\" ; if ($?) { gcc p1.c -o p1 } ; if ($?) { .\p1 }
Enter the productions: E -> E+E|T
The Given Grammar has no Left Recursion
PS C:\college programs>
```

## PRACTICAL – 5

Write a program to perform left factoring on a grammar.

### **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]!='\0';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("\nGrammar Without Left Factoring : \n");
    printf(" A->%s",modifiedGram);
    printf("\n X->%s\n",newGram);
}

```

## Output:

```

terminal Help p1.c - college programs - Visual Studio Code [Administrator]
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\college programs> cd "c:\college programs\" ; if ($?) { gcc p1.c -o p1 } ; if ($?) { .\p1 }
Enter Production : A->ab|ac

Grammar Without Left Factoring :
A->aX
X->b|c
PS C:\college programs>

```

## PRACTICAL – 6

Write a program to show all the operations of a stack.

### **Program:**

```
#include <stdio.h>

int MAXSIZE = 8;

int stack[8];

int top = -1;

int isempty(){
    if(top == -1)
        return 1;
    else
        return 0;
}

int isfull(){
    if(top == MAXSIZE)
        return 1;
    else
        return 0;
}

int peek(){
    return stack[top];
}

int pop(){
    int data;
    if(!isempty()) {
        data = stack[top];
```

```

        top = top - 1;
        return data;
    } else {
        printf("Could not retrieve data, Stack is empty.\n");
    }
}

int push(int data){
    if(!isfull()) {
        top = top + 1;
        stack[top] = data;
    } else {
        printf("Could not insert data, Stack is full.\n");
    }
}

int main(){
    push(44);
    push(10);
    push(62);
    push(123);
    push(15);
    printf("Element at top of the stack: %d\n", peek());
    printf("Elements: \n");

    // print stack data
    while(!isempty()) {
        int data = pop();
        printf("%d\n", data);
    }
}

```

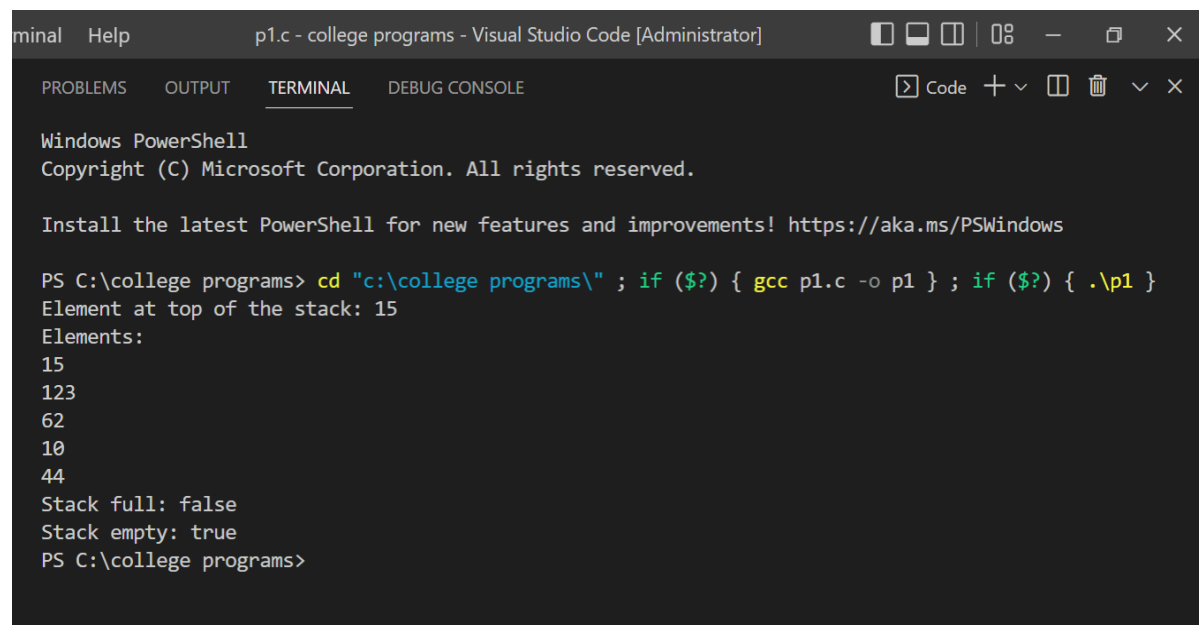
```

}

printf("Stack full: %s\n" , isfull()?"true":"false");
printf("Stack empty: %s\n" , isempty()?"true":"false");
return 0;
}

```

## Output:



```

minal  Help  p1.c - college programs - Visual Studio Code [Administrator]
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\college programs> cd "c:\college programs\" ; if ($?) { gcc p1.c -o p1 } ; if ($?) { .\p1 }
Element at top of the stack: 15
Elements:
15
123
62
10
44
Stack full: false
Stack empty: true
PS C:\college programs>

```



## PRACTICAL – 7

Write a program to find out the leading of the non-terminals in a grammar.

### **Program:**

```
#include<conio.h>

#include<stdio.h>

char arr[18][3]={{'E', '+', 'F'},{'E', '*', 'F'},{'E', '(', 'F'}, {'E', ')', 'F'},{'E', 'i', 'F'},{'E', '$', 'F'},

{'F', '+', 'F'},{'F', '*', 'F'},{'F', '(', 'F'},{'F', ')', 'F'},{'F', 'i', 'F'},{'F', '$', 'F'}, {'T', '+', 'F'},

{'T', '*', 'F'}, {'T', '(', 'F'},{'T', ')', 'F'},{'T', 'i', 'F'},{'T', '$', 'F'}};

char prod[6] = "EETTFF";

char res[6][3]={ {'E', '+', 'T'}, {'T', '\0'}, {'T', '*', 'F'}, {'F', '\0'}, {'(', 'E', ')'}, {'i', '\0'}};

char stack [5][2];

int top = -1;

void install(char pro, char re) {

    int i;

    for (i = 0; i < 18; ++i) {

        if (arr[i][0] == pro && arr[i][1] == re) {

            arr[i][2] = 'T';

            break;

        }

    }

    ++top;

    stack[top][0] = pro;
```

```

    stack[top][1] = re;
}
void main() {
    int i = 0, j;
    char pro, re, pri = ' ';
    for (i = 0; i < 6; ++i) {
        for (j = 0; j < 3 && res[i][j] != '\0'; ++j) {
            if (res[i][j] == '+' || res[i][j] == '*' || res[i][j] == '(' || res[i][j] == ')' ||
res[i][j] == 'i' || res[i][j] == '$') {
                install(prod[i], res[i][j]);
                break;
            }
        }
    }
    while (top >= 0) {
        pro = stack[top][0];
        re = stack[top][1];
        --top;
        for (i = 0; i < 6; ++i) {
            if (res[i][0] == pro && res[i][0] != prod[i]) {
                install(prod[i], re);
            }
        }
    }
    for (i = 0; i < 18; ++i) {
        printf("\n\t");
        for (j = 0; j < 3; ++j)
            printf("%c\t", arr[i][j]);
    }
}

```

```

    }
    getch();
    printf("\n\n");
    for (i = 0; i < 18; ++i) {
        if (pri != arr[i][0]) {
            pri = arr[i][0];
            printf("\n\t%c -> ", pri);
        }
        if (arr[i][2] == 'T')
            printf("%c ", arr[i][1]);
    }
    getch();
}

```

## Output:

```

PS C:\college programs> cd "c:\college programs\" ; if ($?) { gcc p1.c -o p1 } ; if ($?) { .\p1 }

E      +      T
E      *      T
E      (      T
E      )      F
E      i      T
E      $      F
F      +      F
F      *      F
F      (      T
F      )      F
F      i      T
F      $      F
T      +      F
T      *      T
T      (      T
T      )      F
T      i      T
T      $      F

E -> + * ( i
F -> ( i
T -> * ( i
PS C:\college programs>

```

## PRACTICAL – 8

Write a program to implement shift reduce parsing for a string.

### **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();
void main()
{
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: ");
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 --",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\tE->a",stack,ip_sym);
else
printf("\n $%s\t\t%s$\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\tE->E+E",stack,ip_sym);
else
if(!strcmpi(stack,"E\E"))
printf("\n $%s\t\t%s$\t\tE->E\E",stack,ip_sym);
else if(!strcmpi(stack,"E*E"))
printf("\n $%s\t\t%s$\t\tE->E*E",stack,ip_sym);
else
printf("\n $%s\t\t%s$\t\tE->E+E",stack,ip_sym);
flag=1;
}
if(!strcmpi(stack,"E")&&ip_ptr==len)
{

```

```

printf("\n $%s\t\t%s$\t\tACCEPT",stack,ip_sym);
getch();
exit(0);
}
if(flag==0)
{
printf("\n%s\t\t%s\t\t reject",stack,ip_sym);
exit(0);
}
return;
}

```

## Output:

```

PS C:\college programs> cd "c:\college programs\" ; if ($?) { gcc p1.c -o p1 } ; if ($?) { .\p1 }

SHIFT REDUCE PARSER

GRAMMER
E->E+E
E->E/E
E->E*E
E->a/b
enter the input symbol: a+b

stack implementation table
stack      input symbol      action
-----
$          a+b$          --
$a         +b$          shift a
$E         +b$          E->a
$E+        b$          shift +
$E+b       $          shift b
$E+E       $          E->b
$E         $          E->E+E
$E         $          ACCEPT
PS C:\college programs>

```

## PRACTICAL – 9

Write a program to find out the FIRST of the non-terminals in a grammar.

### **Program:**

```
#include<stdio.h>
#include<conio.h>
char array[10][20],temp[10];
int c,n;
void fun(int,int[]);
int fun2(int i,int j,int p[],int );
void main()
{
    int p[2],i,j;
    printf("Enter the no. of productions :");
    scanf("%d",&n);
    printf("Enter the productions :\n");
    for(i=0;i<n;i++)
        scanf("%s",array[i]);
    for(i=0;i<n;i++)
    {
        c=-1,p[0]=-1,p[1]=-1;
        fun(i,p);
        printf("First(%c) : [ ",array[i][0]);
        for(j=0;j<=c;j++)
            printf("%c,",temp[j]);
        printf("\b ].\n");
    }
}
```



```

    getch();
}
}
int fun2(int i,int j,int p[],int key)
{
    int k;
    if(!key)
    {
        for(k=0;k<n;k++)
            if(array[i][j]==array[k][0])
                break;
        p[0]=i;p[1]=j+1;
        fun(k,p);
        return 0;
    }
    else
    {
        for(k=0;k<=c;k++)
        {
            if(array[i][j]==temp[k])
                break;
        }
        if(k>c)return 1;
        else return 0;
    }
}
void fun(int i,int p[])

```

```

{
int j,k,key;
for(j=2;array[i][j] != NULL; j++)
{
if(array[i][j-1]=='/')
{
if(array[i][j]>= 'A' && array[i][j]<='Z')
{
key=0;
fun2(i,j,p,key);
}
else
{
key = 1;
if(fun2(i,j,p,key))
temp[++c] = array[i][j];
if(array[i][j]== '@'&& p[0]!=-1)
{
if(array[p[0]][p[1]]>='A' && array[p[0]][p[1]] <='Z')
{
key=0;
fun2(p[0],p[1],p,key);
}
else
if(array[p[0]][p[1]] != '/'&& array[p[0]][p[1]]!=NULL)
{
if(fun2(p[0],p[1],p,key))

```

```
temp[++c]=array[p[0]][p[1]];
}
}
}
}
}
}
```

## Output:

```
Enter the no. of productions :6
Enter the productions :
S/aBDh
B/cC
C/bC/e
E/g/e
D/E/F
F/f/e
First(S) : [ a ].
First(B) : [ c ].
First(C) : [ b,e ].
First(E) : [ g,e ].
First(D) : [ g,e,f ].
First(F) : [ f,e ].
PS C:\college programs>
```

## PRACTICAL – 10

Write a program to check whether a grammar is operator precedent.

### **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"};
int top=0,l;
char prec[9][9]={
    '>', '>', '<', '<', '<', '<', '<', '>', '>',
    '>', '>', '<', '<', '<', '<', '<', '>', '>',
    '>', '>', '>', '>', '<', '<', '<', '>', '>',
    '>', '>', '>', '>', '<', '<', '<', '>', '>',
    '>', '>', '>', '>', '>', '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+i++);
stack[top+1]='\0';
}
int reduce()
{
int i,len,found,t;
for(i=0;i<5;i++)
{
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;
    }
}

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;");
}

```

## Output:

```

Enter the string
i*(i+i)*i

STACK  INPUT  ACTION
$i    *(i+i)*i$  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;
PS C:\college programs>

```