1Q: Write a C program to simulate FOLLOW of given Context Free Grammar.

#include<stdio.h>

#include<ctype.h>

#include<string.h>

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

char a[10][10],f[10];

void follow(char c);

void first(char c);

int main()

{

int i,j,z ;

char c,ch;

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

scanf("%d",&n);

printf("Enter the productions (Epsilon = $):\n");

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

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

do{

m=0;

printf("Enter the element whose follos is to be Found: ");

scanf("%c",&c);

follow(c);

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

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

printf("%c",f[i]);

printf("}\n");

printf("Do you want to continue (0/1)?: ");

scanf("%d%c",&z,&ch);

}while (z==1);

}

void follow(char c)

{

if(a[0][0]==c)f[m++] = '$';

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

{

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

{

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

{

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

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

follow(a[i][0]);

}

}

}

}

void first(char c)

{

int k;

if(!(isupper(c)))f[m++]=c;

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

{

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

{

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

else if(islower(a[k][2]))f[m++]=a[k][2];

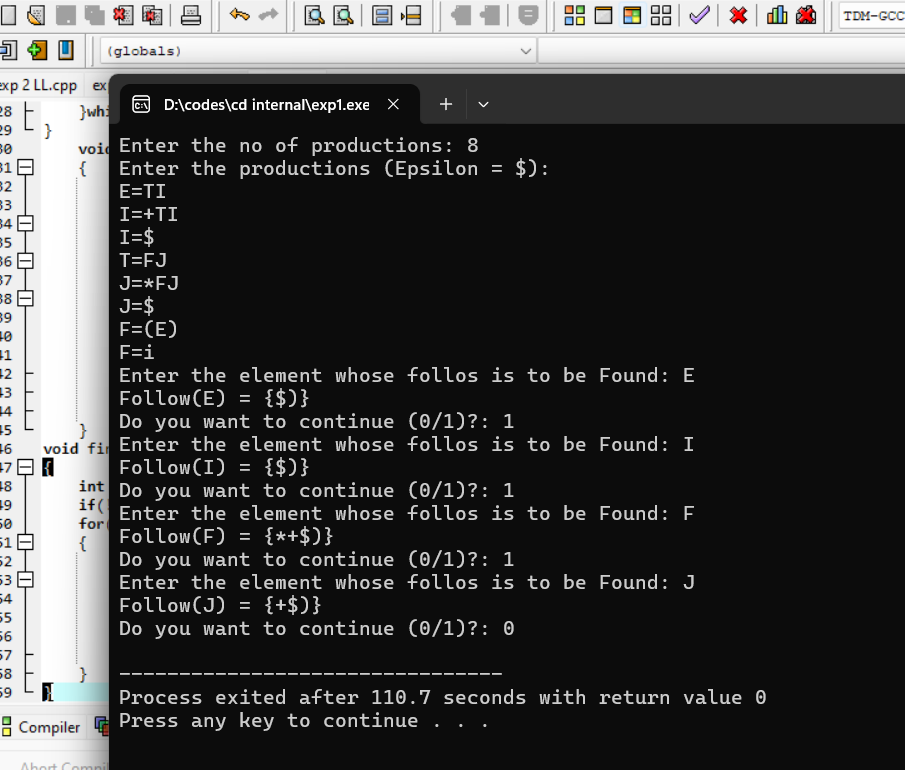
else first(a[k][2]);

}

}

}

OUTPUT:



2Q: Write a C program to construct a LL(1) parser for an expression

#include<stdio.h>

#include<string.h>

char str[25],st[25],\*temp,v,ch,ch1;

char t[5][6][10]={"$","$","TX","TX","$","$",

"+TX","$","$","$","e","e",

"$","$","FY","FY","$","$",

"e","\*FY","$","$","e","e",

"$","$","i","(E)","$","$"};

int i,k,n,top=-1,r,c,m,flag=0;

void push(char t)

{

top++;

st[top]=t;

}

char pop()

{

ch1=st[top];

top--;

return ch1;

}

main()

{

printf("enter the string:\n");

scanf("%s",str);

n=strlen(str);

str[n++]='$';

i=0;

push('$');

push('E');

printf("stack\tinput\toperation\n");

while(i<n)

{

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

printf("%c",st[k]);

printf("\t");

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

printf("%c",str[k]);

printf("\t");

if(flag==1)

printf("pop");

if(flag==2)

printf("%c->%s",ch,t[r][c]);

if(str[i]==st[top])

{

flag=1;

ch=pop();

i++;

}

else

{

flag=2;

if(st[top]=='E')

r=0;

else if(st[top]=='X')

r=1;

else if(st[top]=='T')

r=2;

else if(st[top]=='Y')

r=3;

else if(st[top]=='F')

r=4;

else

break;

if(str[i]=='+')

c=0;

else if(str[i]=='\*')

c=1;

else if(str[i]=='i')

c=2;

else if(str[i]=='(')

c=3;

else if(str[i]==')')

c=4;

else if(str[i]=='$')

c=5;

else

break;

if(strcmp(t[r][c],"$")==0)

break;

ch=pop();

temp=t[r][c];

m=strlen(temp);

if(strcmp(t[r][c],"e")!=0)

{

for(k=m-1;k>=0;k--)

push(temp[k]);

}

}

printf("\n");

}

if(i==n)

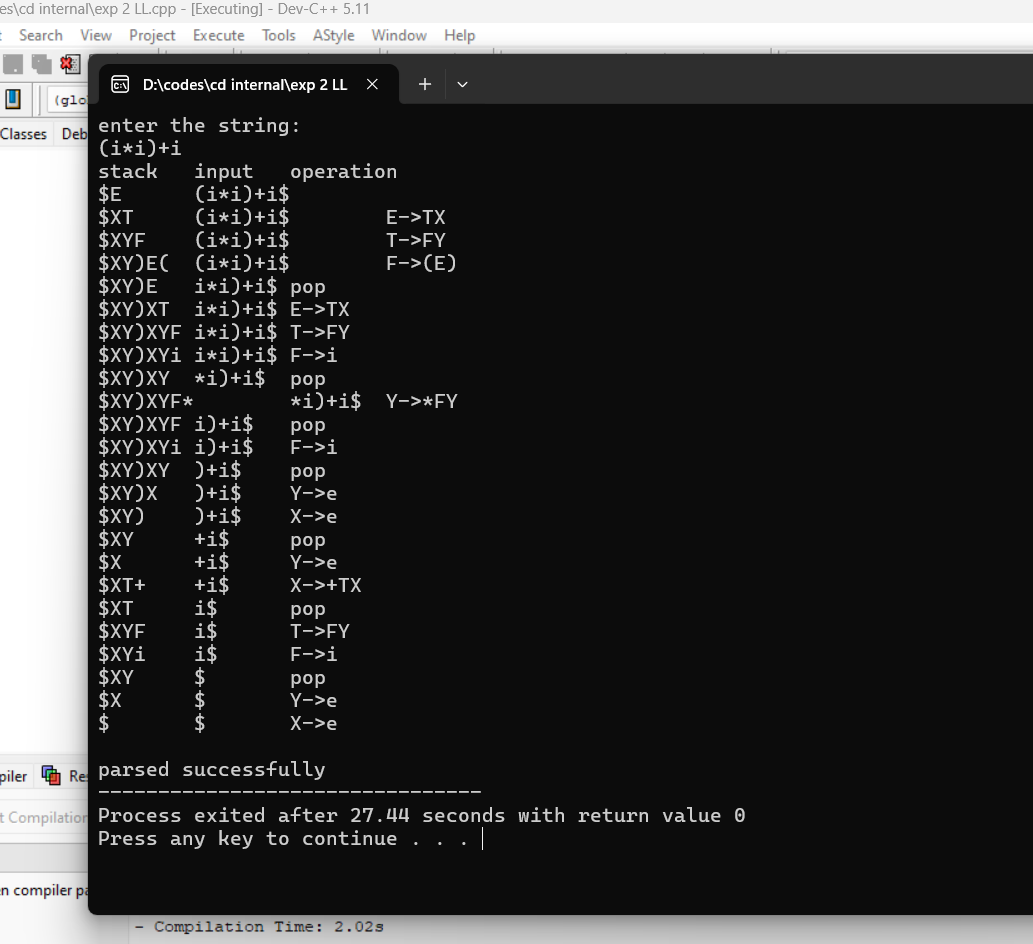
printf("\nparsed successfully");

else

printf("\nnot parsed");

}

Output:



3Q: Write a C program to design a lexical analyser for given language, which should ignore the

redundant spaces, tabs, new lines, find the tokens and also count the number of lines using

C program

Procedure:

 It is the first phase of the compiler. It gets input from the source program and

produces tokens as output.

 It reads the characters one by one, starting from left to right and forms the

tokens.

Token: It represents a logically cohesive sequence of characters such as keywords, operators,

identifiers, special symbols etc.

Example: a+b=20

Here, a,b,+,=,20 are all separate tokens.

Group of characters forming a token is called the Lexeme.

The lexical analyzer not only generates a token but also enters the lexeme into the symbol

table if it is not already there. Its main task is to read the input characters and produce as

output a sequence of tokens that the parser uses for syntax analysis. Upon receiving a "get

next token" command from the parser, the lexical analyzer reads input characters until it can

identify the next token.

Logic/Algorithm:

1. Read the C program as input and stores in a file.

2. Check all the characters from the file from left to right whether character is alphabet or

digit or special symbol.

3. If the input is operator prints as special symbol.

4. If the input is number prints as number.

5. If the input is identifier prints as identifier.

6. If the input is keyword prints as keyword.

7. Print no lines of code

Source Code/Program Code:

#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||strc

mp("float",str)==0||strcmp("char",str)==0||strcmp("double",str)==0||strcmp("static",str)==0||st

rcmp("switch",str)==0||strcmp("case",str)==0)

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

else

printf("\n%s is an identifier",str);

}

int main()

{

FILE \*f1,\*f2,\*f3;

char c, str[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);

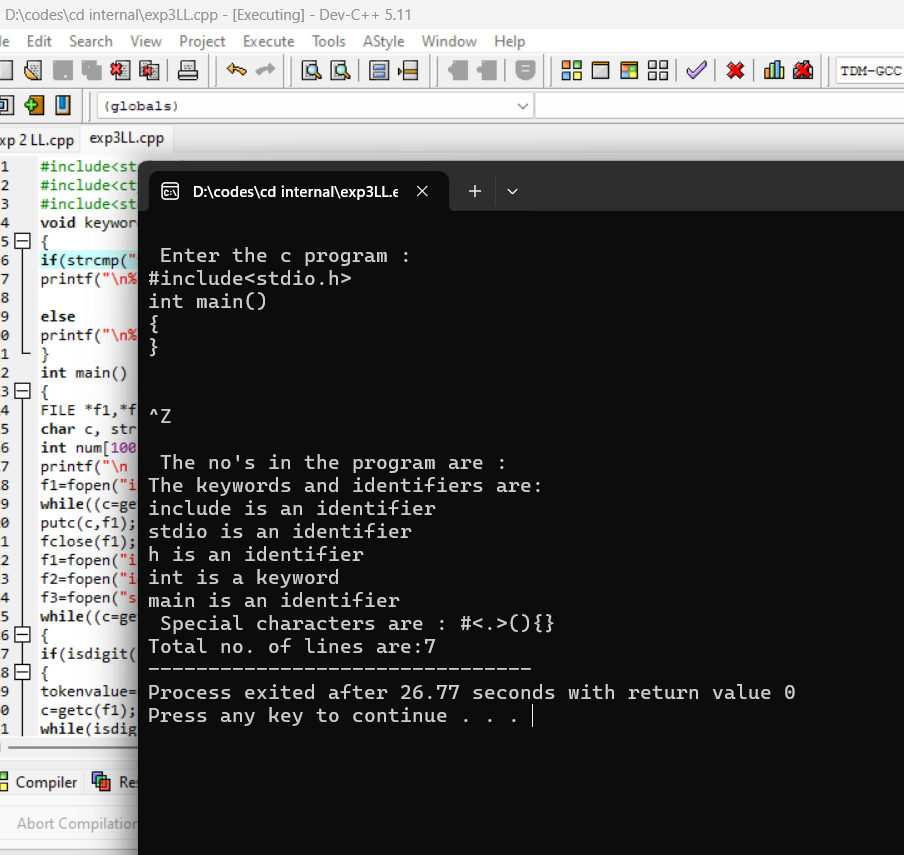
printf("\n");

fclose(f3);

printf("Total no. of lines are:%d", lineno);

}

Output://after write c program press ctrl+z and then press enter



4Q: Write a C program to construct a recursive descent parser for an expression.

#include<stdio.h>

#include<string.h>

#include<ctype.h>

char input[10];

int i,error;

void E();

void T();

void Eprime();

void Tprime();

void F();

int main()

{

i=0;

error=0;

printf("Enter an arithmetic expression : "); // Eg: a+a\*a

gets(input);

E();

if(strlen(input)==i&&error==0)

printf("\nAccepted..!!!\n");

else printf("\nRejected..!!!\n");

}

void E()

{

T();

Eprime();

}

void Eprime()

{

if(input[i]=='+')

{

i++;

T();

Eprime();

}

}

void T()

{

F();

Tprime();

}

void Tprime()

{

if(input[i]=='\*')

{

i++;

F();

Tprime();

}

}

void F()

{

if(isalnum(input[i]))i++;

else if(input[i]=='(')

{

i++;

E();

if(input[i]==')')

i++;

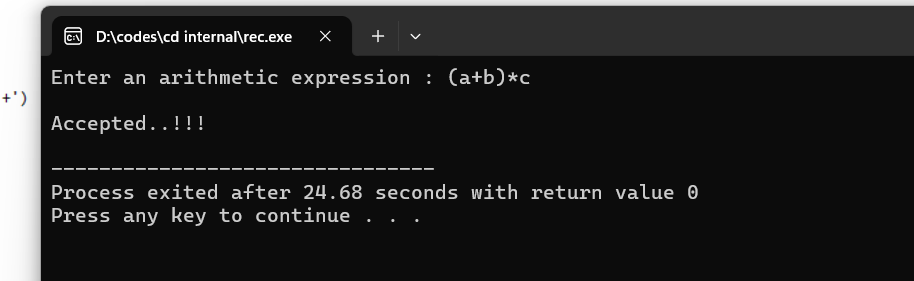
else error=1;

}

else error=1;

}

Output:



5Q:Write a Cprogram to simulate first of a context free grammar

#include<stdio.h>

#include<ctype.h>

void FIRST(char);

int count,n=0;

char prodn[10][10],first[10];

main()

{

int i,choice;

char c,ch;

printf("How many productions?:");

scanf("%d",&count);

printf("Enter %d productions epsilon=$:\n\n",count);

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

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

do

{

n=0;

printf("Element:");

scanf("%c",&c);

FIRST(c);

printf("\nFIRST(%c)={",c);

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

printf("%c",first[i]);

printf("}\n");

printf("press 1 to continue:");

scanf("%d%c",&choice,&ch);

}

while(choice==1);

}

void FIRST(char c)

{

int j;

if(!(isupper(c)))first[n++]=c;

for(j=0;j<count;j++)

{

if(prodn[j][0]==c)

{

if(prodn[j][2]=='$')first[n++]='$';

else if(islower(prodn[j][2]))first[n++]=prodn[j][2];

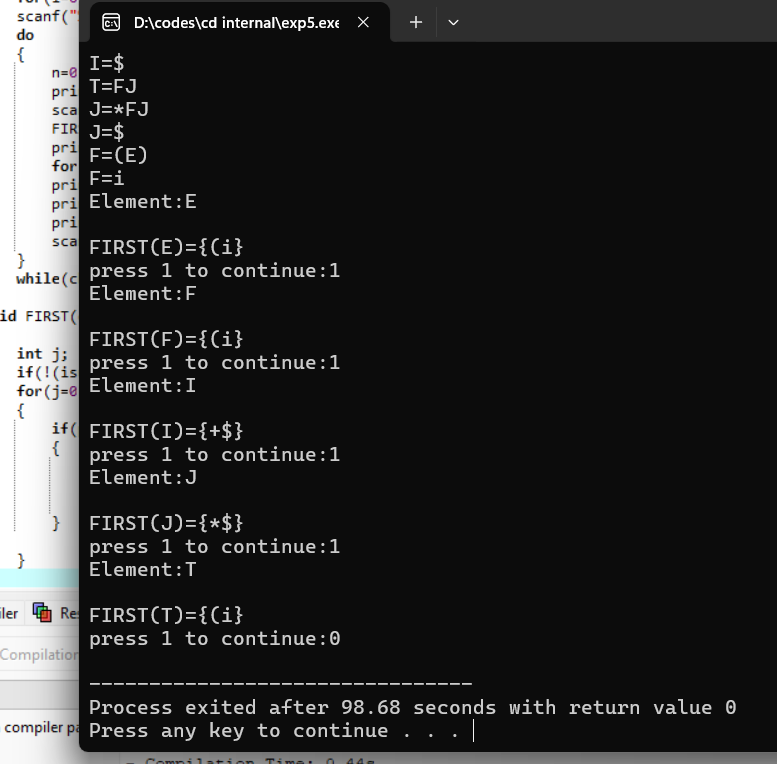
else FIRST(prodn[j][2]);

}

}

}

Output:



6Q: Write a C program to implement a shift-reduce parser.

#include<stdio.h>

#include<stdlib.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];

int check();

int 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:\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++;

}

check();

}

int check()

{

int flag=0;

temp2[0]=stack[st\_ptr];

temp2[1]='\0';

if((isalpha(temp2[0])))

{

stack[st\_ptr]='E';

printf("\n $%s\t\t%s$\t\t\tE->%s",stack,ip\_sym,temp2);

flag=1;

}

if((!strcmp(temp2,"+"))||(!strcmp(temp2,"\*"))||(!strcmp(temp2,"/")))

{

flag=1;

}

if((!strcmp(stack,"E+E"))||(!strcmp(stack,"E/E"))||(!strcmp(stack,"E\*E")))

{

if(!strcmp(stack,"E+E"))

{

strcpy(stack,"E");

printf("\n $%s\t\t%s$\t\t\tE->E+E",stack,ip\_sym);

}

else if(!strcmp(stack,"E/E"))

{

strcpy(stack,"E");

printf("\n $%s\t\t %s$\t\t\tE->E/E",stack,ip\_sym);

}

else

{

strcpy(stack,"E");

printf("\n $%s\t\t%s$\t\t\tE->E\*E",stack,ip\_sym);

}

flag=1;

st\_ptr=0;

}

if(!strcmp(stack,"E")&&ip\_ptr==len)

{

printf("\n $%s\t\t%s$\t\t\tACCEPT",stack,ip\_sym);

exit(0);

}

if(flag==0)

{

printf("\n $%s\t\t%s$\t\t\tReject",stack,ip\_sym);

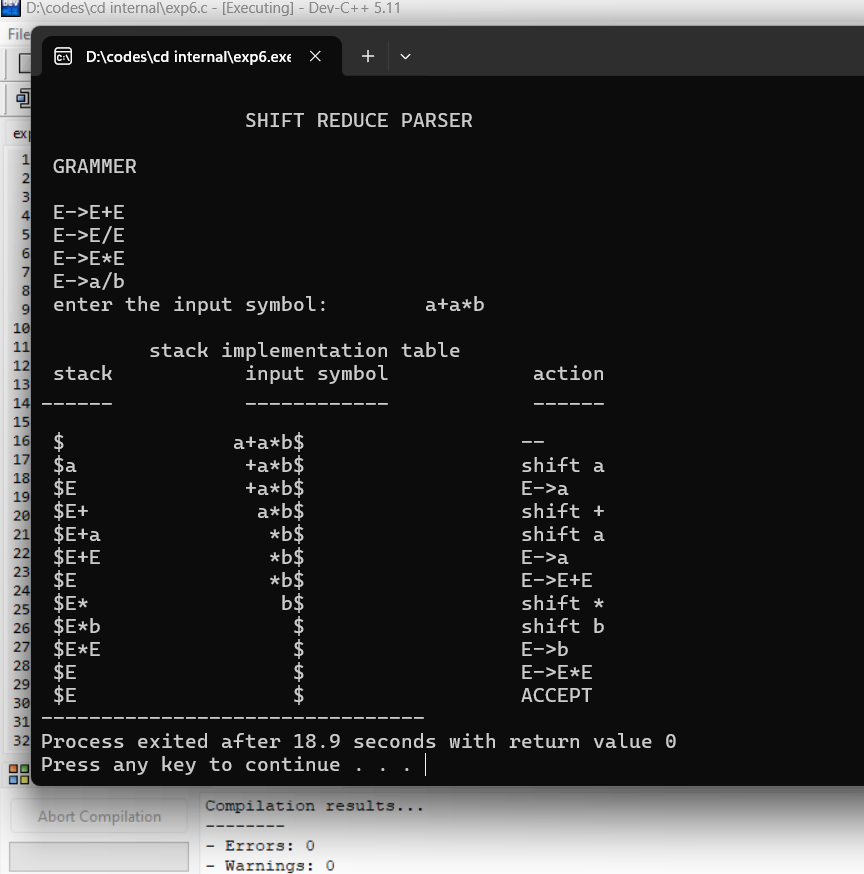
exit(0);

}

return 0;

}

output:



7Q: Write a C program to verify whether the given grammar is Operator precedence  
grammar or not?

#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+i++);

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;");

}

Output:

