Experiment :6

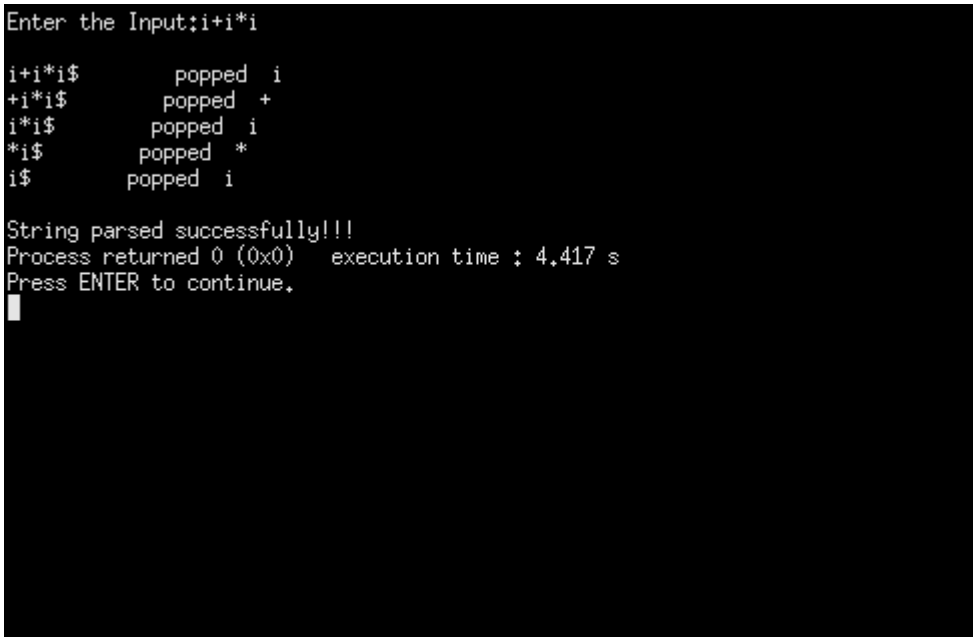
Aim: Design predictive parser for the given language.

Source code:

```
char input[20];int len,ln,err=0;
void E()
{
   T();
   E1();
}
void E1()
{
   if(*input=='+')
   {
      match('+');
      T();
      E1();
   }
  else return;
}
void T()
{
   F();
   T1();
}
void T1()
{
      if(*input=='*')
     {
      match('*');
      F();
      T1();
     }
  else return;
}
void F()
{
      if(*input=='(')
   {
      match('(');
      E();
      match(')');
   }
  else match('i');
}
void match(char topChar)
{
```

```
    if(*input==topChar)
    {
        printf("\n%s        popped  %c",input,topChar);
        ln++;
        strcpy(input,&input[1]); // pops matched i/p symbol from input
    }
    else  {printf("\nError   %c   didn't produced by any production at this
place",*input);err++;}
}
int main()
{
    printf("Enter the Input:");
    gets(input);
    len=strlen(input);
    input[len]='$';
    input[len+1]='\0';
    E();
    if(err==0&&ln==len)        printf("\n\nString parsed successfully!!!");
    else
          {
             printf("\n\n String is not parsed successfully\nErrors occurred or Input
contains invalid characters\n\n");
    }
    return 0;
}
```

Output:



Experiment:7

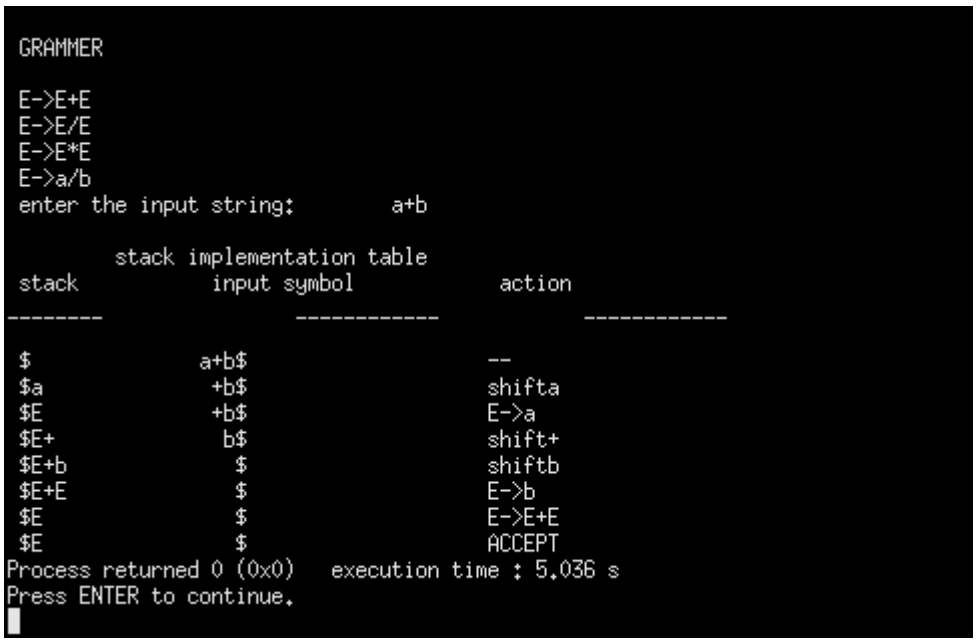
Aim: To implement shift reduce parsing algorithm.

Source code:

```
#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];
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 string:\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((!strcmp(temp2,"a"))||(!strcmp(temp2,"b")))
{
stack[st_ptr]='E';
if(!strcmp(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((!strcmp(temp2,"+"))||(strcmp(temp2,"*"))||(!strcmp(temp2,"/")))
{
   flag=1;
}
if((!strcmp(stack,"E+E"))||(!strcmp(stack,"E\E"))||(!strcmp(stack,"E*E")))
{
strcpy(stack,"E");
st_ptr=0;
if(!strcmp(stack,"E+E"))
printf("\n $%s\t\t%s$\t\t\tE->E+E",stack,ip_sym);
else
if(!strcmp(stack,"E\E"))
printf("\n $%s\t\t%s$\t\t\tE->E\E",stack,ip_sym);
else
if(!strcmp(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(!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\t%s\t\t reject",stack,ip_sym);
exit(0);
}
return;
}
```

Output:


GRAMMER
E->E+E
E->E/E
E->E*E
E->a/b
enter the input string: a+b
stack implementation table
stack input symbol action
$ a+b$ --
$a +b$ shifta
$E +b$ E->a
$E+ b$ shift+
$E+b $ shiftb
$E+E $ E->b
$E $ E->E+E
$E $ ACCEPT
Process returned 0 (0x0) execution time : 5.036 s
Press ENTER to continue.