Week 6: Recursive Descent Parser

6. Implement Recursive Descent Parser for the Expression Grammar given below.

E → TE’
E’→ +TE’ | ϵ
T → FT’
T’→ *FT’ | ϵ
F → (E) | i

Code:

```
#include"stdio.h"
#include"string.h"
#include"stdlib.h"
#include"ctype.h"

char ip_sym[15],ip_ptr=0,op[50],tmp[50];
void e_prime();
void e();
void t_prime();
void t();
void f();
void advance();
int n=0;
void e()
{
 strcpy(op,"TE'");
 printf("E=%-25s",op);
 printf("E->TE'\n");
 t();
 e_prime();
}
```

```
void e_prime()
{
int i,n=0,l;
for(i=0;i<=strlen(op);i++)
   if(op[i]!='e')
 tmp[n++]=op[i];
strcpy(op,tmp);
l=strlen(op);
for(n=0;n < l && op[n]!='E';n++);
if(ip_sym[ip_ptr]=='+')
 {
    i=n+2;
 do
 {
 op[i+2]=op[i];
 i++;
 }while(i<=l);
  op[n++]='+';
  op[n++]='T';
  op[n++]='E';
  op[n++]=39;
  printf("E=%-25s",op);
  printf("E'->+TE'\n");
  advance();
  t();
  e_prime();
 }
 else
 {
    op[n]='e';
  for(i=n+1;i<=strlen(op);i++)
 op[i]=op[i+1];
 printf("E=%-25s",op);
 printf("E'->e");
 }
```

```
}
void t()
{
 int i,n=0,l;
 for(i=0;i<=strlen(op);i++)
  if(op[i]!='e')
   tmp[n++]=op[i];
 strcpy(op,tmp);
 l=strlen(op);
for(n=0;n < l && op[n]!='T';n++);

 i=n+1;
 do
 {
  op[i+2]=op[i];
  i++;
 }while(i < l);
 op[n++]='F';
 op[n++]='T';
 op[n++]=39;
 printf("E=%-25s",op);
 printf("T->FT'\n");
 f();
 t_prime();
}

void t_prime()
{
int i,n=0,l;
for(i=0;i<=strlen(op);i++)
    if(op[i]!='e')
 tmp[n++]=op[i];
strcpy(op,tmp);
l=strlen(op);
for(n=0;n < l && op[n]!='T';n++);
```

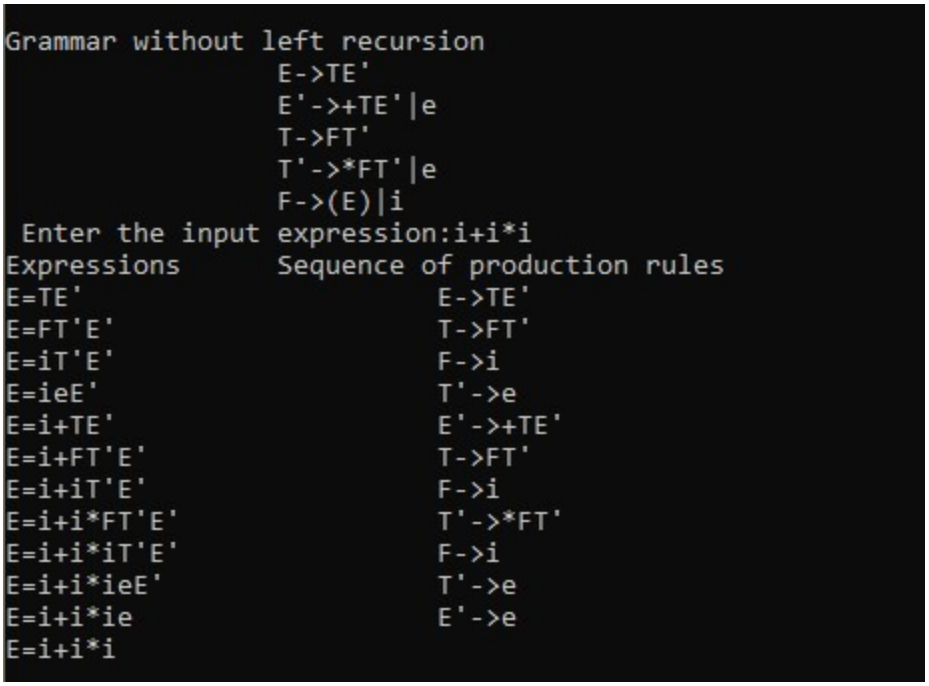
```
if(ip_sym[ip_ptr]=='*')
 {
    i=n+2;
 do
 {
 op[i+2]=op[i];
 i++;
 }while(i < l);
  op[n++]='*';
  op[n++]='F';
  op[n++]='T';
  op[n++]=39;
  printf("E=%-25s",op);
  printf("T'->*FT'\n");
  advance();
  f();
  t_prime();
 }
 else
 {
   op[n]='e';
  for(i=n+1;i<=strlen(op);i++)
 op[i]=op[i+1];
 printf("E=%-25s",op);
 printf("T'->e\n");
 }
}

void f()
{
int i,n=0,l;
for(i=0;i<=strlen(op);i++)
    if(op[i]!='e')
 tmp[n++]=op[i];
 strcpy(op,tmp);
```

```
l=strlen(op);
for(n=0;n < l && op[n]!='F';n++);
 if((ip_sym[ip_ptr]=='i')||(ip_sym[ip_ptr]=='I'))
 {
 op[n]='i';
 printf("E=%-25s",op);
 printf("F->i\n");
 advance();
 }
 else
 {
  if(ip_sym[ip_ptr]=='(')
  {
   advance();
   e();
   if(ip_sym[ip_ptr]==')')
   {
    advance();
     i=n+2;
 do
 {
 op[i+2]=op[i];
 i++;
 }while(i<=l);
  op[n++]='(';
  op[n++]='E';
  op[n++]=')';
  printf("E=%-25s",op);
  printf("F->(E)\n");
   }
  }
  else
  {
   printf("\n\t syntax error");
```

```
   exit(1);
  }
 }
}

void advance()
{
 ip_ptr++;
}

void main()
{
 int i;

 printf("\nGrammar without left recursion");
 printf("\n\t\t E->TE' \n\t\t E'->+TE'|e \n\t\t T->FT' ");
 printf("\n\t\t T'->*FT'|e \n\t\t F->(E)|i");
 printf("\n Enter the input expression:");
 scanf("%s",ip_sym);
 printf("Expressions");
 printf("\t Sequence of production rules\n");
  e();
  for(i=0;i < strlen(ip_sym);i++)
 {
  if(ip_sym[i]!='+'&&ip_sym[i]!='*'&&ip_sym[i]!='('&&
     ip_sym[i]!=')'&&ip_sym[i]!='i'&&ip_sym[i]!='I')
  {
   printf("\nSyntax error");
   break;
  }
  for(i=0;i<=strlen(op);i++)
    if(op[i]!='e')
 tmp[n++]=op[i];
    strcpy(op,tmp);
    printf("\nE=%-25s",op);
```

```
}
}
```

Output:



```
Grammar without left recursion
                E->TE'
                E'->+TE'|e
                T->FT'
                T'->*FT'|e
                F->(E)|i
 Enter the input expression:i+i*i
Expressions      Sequence of production rules
E=TE'                       E->TE'
E=FT'E'                     T->FT'
E=iT'E'                     F->i
E=ieE'                      T'->e
E=i+TE'                     E'->+TE'
E=i+FT'E'                   T->FT'
E=i+iT'E'                   F->i
E=i+i*FT'E'                 T'->*FT'
E=i+i*iT'E'                 F->i
E=i+i*ieE'                  T'->e
E=i+i*ie                    E'->e
E=i+i*i
```