CONSTANT PROPOGATION

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

void input();
void output();
void change(int p,char *res);
void constant();

struct expr
{
char op[2],op1[5],op2[5],res[5];
int flag;
}arr[10];
int n;
void main()
{

input();
constant();
output();

}
void input()
{
int i;
printf("\n\nEnter the maximum number of expressions : ");
scanf("%d",&n);
printf("\nEnter the input : \n");
for(i=0;i<n;i++)
{
scanf("%s",arr[i].op);
scanf("%s",arr[i].op1);
scanf("%s",arr[i].op2);
scanf("%s",arr[i].res);
arr[i].flag=0;
}
}
void constant()
{
int i;
int op1,op2,res;
```

```
char op,res1[5];
for(i=0;i<n;i++)
{
if(isdigit(arr[i].op1[0]) && isdigit(arr[i].op2[0]) || strcmp(arr[i].op,"=")==0)
/*if both digits, store them in variables*/
{
op1=atoi(arr[i].op1);
op2=atoi(arr[i].op2);
op=arr[i].op[0];
switch(op)
{
case '+':
res=op1+op2;
break;
case '-':
res=op1-op2;
break;
case '*':
res=op1*op2;
break;
case '/':
res=op1/op2;
break;
case '=':
res=op1;
break;
}
sprintf(res1,"%d",res);
arr[i].flag=1;
change(i,res1);
}
}
}
void output()
{
int i=0;
printf("\nOptimized code is : ");
for(i=0;i<n;i++)
{
if(!arr[i].flag)
{
printf("\n%s %s %s %s",arr[i].op,arr[i].op1,arr[i].op2,arr[i].res);
}
}
```

```
}
void change(int p,char *res)
{
int i;
for(i=p+1;i<n;i++)
{
if(strcmp(arr[p].res,arr[i].op1)==0)
strcpy(arr[i].op1,res);
else if(strcmp(arr[p].res,arr[i].op2)==0)
strcpy(arr[i].op2,res);
}
}
```

```
student@P51:~/Documents$ gcc constt.c
constt.c: In function ‘constant’:
constt.c:49:5: warning: implicit declaration of function ‘atoi’ [-Wimplicit-func
tion-declaration]
   49 | op1=atoi(arr[i].op1);
      |     ^~~~
student@P51:~/Documents$ ./a.out


Enter the maximum number of expressions : 4

Enter the input :
= 3 - a
+ a  b t1
+ a  c t2
+ t1 t2 t3

Optimized code is :
+ 3 b t1
+ 3 c t2
+ t1 t2 t3student@P51:~/Documents$
```

SHIFT REDUCE PRASER

```
#include<stdio.h>
#include<string.h>
int k=0,z=0,i=0,j=0,c=0;
char a[16],ac[20],stk[15],act[10];
void check();
int main()
  {
```

```
    puts("GRAMMAR is E->E+E \n E->E*E \n E->(E) \n E->id");
    puts("enter input string ");
    scanf("%s",a);
    c=strlen(a);
    strcpy(act,"SHIFT->");
    puts("stack \t input \t action");
    for(k=0,i=0; j<c; k++,i++,j++)
     {
       if(a[j]=='i' && a[j+1]=='d')
         {
            stk[i]=a[j];
            stk[i+1]=a[j+1];
            stk[i+2]='\0';
            a[j]=' ';
            a[j+1]=' ';
            printf("\n$%s\t%s$\t%sid",stk,a,act);
            check();
         }
       else
         {
            stk[i]=a[j];
            stk[i+1]='\0';
            a[j]=' ';
            printf("\n$%s\t%s$\t%ssymbols",stk,a,act);
            check();
         }
     }

  }
void check()
  {
    strcpy(ac,"REDUCE TO E");
    for(z=0; z<c; z++)
      if(stk[z]=='i' && stk[z+1]=='d')
        {
          stk[z]='E';
          stk[z+1]='\0';
          printf("\n$%s\t%s$\t%s",stk,a,ac);
          j++;
        }
    for(z=0; z<c; z++)
      if(stk[z]=='E' && stk[z+1]=='+' && stk[z+2]=='E')
        {
```

```
        stk[z]='E';
        stk[z+1]='\0';
        stk[z+2]='\0';
        printf("\n$%s\t%s$\t%s",stk,a,ac);
        i=i-2;
      }
   for(z=0; z<c; z++)
     if(stk[z]=='E' && stk[z+1]=='*' && stk[z+2]=='E')
      {
        stk[z]='E';
        stk[z+1]='\0';
        stk[z+1]='\0';
        printf("\n$%s\t%s$\t%s",stk,a,ac);
        i=i-2;
      }
   for(z=0; z<c; z++)
     if(stk[z]=='(' && stk[z+1]=='E' && stk[z+2]==')')
      {
        stk[z]='E';
        stk[z+1]='\0';
        stk[z+1]='\0';
        printf("\n$%s\t%s$\t%s",stk,a,ac);
        i=i-2;
      }
 }
```

```
student@P51:~/Documents$ gcc srp.c
student@P51:~/Documents$ ./a.out
GRAMMAR is E->E+E
 E->E*E
 E->(E)
 E->id
enter input string
id+id*id+id
stack    input   action

$id       +id*id+id$    SHIFT->id
$E        +id*id+id$    REDUCE TO E
$E+        id*id+id$    SHIFT->symbols
$E+id        *id+id$    SHIFT->id
$E+E         *id+id$    REDUCE TO E
$E           *id+id$    REDUCE TO E
$E*           id+id$    SHIFT->symbols
$E*id           +id$    SHIFT->id
$E*E            +id$    REDUCE TO E
$E              +id$    REDUCE TO E
$E+              id$    SHIFT->symbols
$E+id              $    SHIFT->id
$E+E               $    REDUCE TO E
$E                 $    REDUCE TO Estudent@P51:~/Documents$
```

RECURSIVE DESCENT PARSER

```
#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);
 }

}
```

```
student@P51:~/Documents$ gcc const.c
student@P51:~/Documents$ ./a.out

Grammar without left recursion
                E->TE'
                E'->+TE'|e
                T->FT'
                T'->*FT'|e
                F->(E)|i
 Enter the input expression: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+ieE'                    T'->e
E=i+ie                      E'->e
E=i+i                       student@P51:~/Documents$
```