**1. Implement Recursive Descent Parser for the Expression Grammar given below.**

**E → TE’**

**E’→ +TE’ | ͼ**

**T → FT’**

**T’→ \*FT’ | ͼ**

**F → (E) | i**

**Program:**

**C implementation of Recursive Descent Parser for the Expression Grammar is given below.**

#include<stdio.h>

#include<string.h>

int E(),Edash(),T(),Tdash(),F();

char \*ip;

char string[50];

int main()

{

printf("Enter the string\n");

scanf("%s",string);

ip=string;

printf("\n\nInput\tAction\n--------------------------------\n");

if(E() && \*ip==’\0’){

printf("\n--------------------------------\n");

printf("\n String is successfully parsed\n");

}

else{

printf("\n--------------------------------\n");

printf("Error in parsing String\n");

}

}

int E()

{

printf("%s\tE->TE' \n",ip);

if(T())

{

if(Edash())

{

return 1;

}

else

return 0;

}

else

return 0;

}

int Edash()

{

if(\*ip=='+')

{

printf("%s\tE'->+TE' \n",ip);

ip++;

if(T())

{

if(Edash())

{

return 1;

}

else

return 0;

}

else

return 0;

}

else

{

printf("%s\tE'->^ \n",ip);

return 1;

}

}

int T()

{

printf("%s\tT->FT' \n",ip);

if(F())

{

if(Tdash())

{

return 1;

}

else

return 0;

}

else

return 0;

}

int Tdash()

{

if(\*ip=='\*')

{

printf("%s\tT'->\*FT' \n",ip);

ip++;

if(F())

{

if(Tdash())

{

return 1;

}

else

return 0;

}

else

return 0;

}

else

{

printf("%s\tT'->^ \n",ip);

return 1;

}

}

int F()

{

if(\*ip=='(')

{

printf("%s\tF->(E) \n",ip);

ip++;

if(E())

{

if(\*ip==')')

{

ip++;

return 0;

}

else

return 0;

}

else

return 0;

}

else if(\*ip=='i')

{

ip++;

printf("%s\tF->id \n",ip);

return 1;

}

else

return 0;

}

**Test cases:**

|  |  |
| --- | --- |
| i+i\*i | String is successfully parsed |
| i+i | String is successfully parsed |
| i\*i | String is successfully parsed |
| i\*i+i\*i+i | String is successfully parsed |
| i+\*+i | Error in parsing String |
| i+i\* | Error in parsing String |

**2. Construct Recursive Descent Parser for the grammar**

**G = ({S, L}, {(, ), a, ,}, {S → (L) | a ; L→ L, S | S}, S) and verify the acceptability of**

**the following strings:**

**s-> (l) | a**

**l-> sl’**

**l’->,sl’ | eps**

**C program:**

#include<stdio.h>

char string[50],\*ip;

int s()

{

if(\*ip=='(')

{

printf("%s\ts->(l)\n",ip);

ip++;

if(l())

{

if(\*ip==')')

{

ip++;

return 1;

}

else

{

return 0;

}

}

else

{

return 0;

}

}

else if(\*ip=='a')

{

printf("%s\ts->a\n",ip);

ip++;

return 1;

}

else

{

return 0;

}

}

int l()

{

printf("%s\tl->sl'\n",ip);

if(s())

{

if(ldash())

{

return 1;

}

else

{

return 0;

}

}

else

{

return 0;

}

}

int ldash()

{

if(\*ip==',')

{

ip++;

printf("%s\tl'->sl'\n",ip);

if(s())

{

if(ldash())

{

return 1;

}

else

{

return 0;

}

}

else

{

return 0;

}

}

else

{

printf("%s\tldash->^\n",ip);

return 1;

}

}

int main()

{

printf("Enter string for parsing:\n");

scanf("%s",string);

ip=string;

printf("\n\nInput\tAction\n--------------------------------\n");

if(s() && \*ip=='\0'){

printf("\n--------------------------------\n");

printf("\n String is successfully parsed\n");

}

else{

printf("\n--------------------------------\n");

printf("Error in parsing String\n");

}

}

**Test cases:**

|  |  |
| --- | --- |
| a | String is successfully parsed |
| (a,a) | String is successfully parsed |
| (a,(a,a)) | String is successfully parsed |
| (a,((a,a),(a,a))) | String is successfully parsed |
| a,(a,a) | Error in parsing String |
| a,() | Error in parsing String |