

# Department of CSE

## SSN College of Engineering

Vishakan Subramanian - 18 5001 196 - Semester VI

26 February 2021

---

### UCS 1602 - Compiler Design

---

#### Exercise 4: Recursive Descent Parser Using C

##### Aim:

Write a program in C to construct **Recursive Descent Parser** for the following grammar which is for arithmetic expression involving + and \*. Check the Grammar for left recursion and convert into suitable for this parser. Write recursive functions for every non-terminal. Call the function for start symbol of the Grammar in main().

G1:

$$\begin{aligned} E &\rightarrow E + T \mid T \\ T &\rightarrow T * F \mid F \\ F &\rightarrow i \end{aligned}$$

Extend this parser to include division, subtraction and parenthesis operators.

G2:

$$\begin{aligned} E &\rightarrow E + T \mid E - T \mid T \\ T &\rightarrow T * F \mid T / F \mid F \\ F &\rightarrow (E) \mid i \end{aligned}$$

## Code - Grammar 1:

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<string.h>
4
5 /*Recursive Descent Parser*/
6
7 /*
8 Grammar: G: E->E+T|T
9           T->T*F|F
10          F->i
11 */
12
13 /*
14 Removed Left Recursion
15 Grammar G': E->TE'
16             E'->+TE'|e
17             T->FT'
18             T'->*FT'|e
19             F->i
20 */
21
22 struct parse_struct{
23     char str[100];
24     int pos;
25     int len;
26 };
27
28 typedef struct parse_struct parser;
29
30 parser E(parser p);
31 parser T(parser p);
32 parser EPrime(parser p);
33 parser F(parser p);
34 parser TPrime(parser p);
35 parser parse(parser p, char s);
36
37 int main(void){
38     parser p;
39
40     printf("\n\t\tRecursive Descent Parser\n");
41     printf("\nEnter a string to parse: ");
42     scanf("%s", p.str);
43
44     p.len = strlen(p.str);
45     p.pos = 0;
46
47     p = E(p);
```

```

48
49     if(p.pos == p.len){
50         //All characters have been parsed
51         printf("\nParse Success!\n");
52     }
53
54     else{
55         //Some characters haven't been parsed, but returned to main
56         printf("\nError parsing at Position %d!\n", p.pos);
57     }
58
59
60     return 0;
61 }
62
63 parser E(parser p){
64     //printf("\nAt E");
65     p = T(p);
66     p = EPrime(p);
67
68     return p;
69 }
70
71 parser T(parser p){
72     //printf("\nAt T");
73     p = F(p);
74     p = TPrime(p);
75
76     return p;
77 }
78
79 parser EPrime(parser p){
80     //printf("\nAt EPrime");
81     if(p.str[p.pos] == '+'){
82         p = parse(p, '+');
83         p = T(p);
84         p = EPrime(p);
85     }
86
87     return p;
88 }
89
90 parser TPrime(parser p){
91     //printf("\nAt TPrime");
92     if(p.str[p.pos] == '*'){
93         p = parse(p, '*');
94         p = F(p);
95         p = TPrime(p);
96     }
97
98     return p;

```

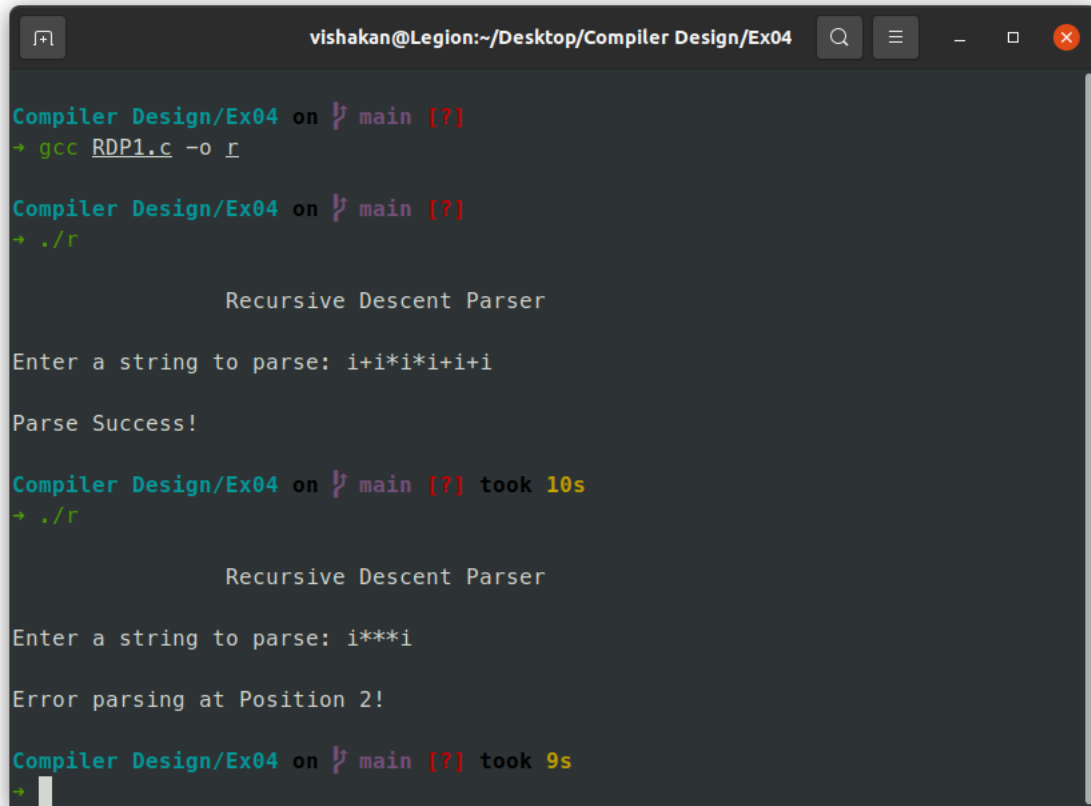
```

99 }
100
101 parser F(parser p){
102     //printf("\nAt F");
103     if(p.str[p.pos] == 'i'){
104         p = parse(p, 'i');
105     }
106     else{
107         printf("\nError parsing at Position %d!\n", p.pos);
108         exit(0);
109     }
110
111     return p;
112 }
113
114 parser parse(parser p, char s){
115     if(p.str[p.pos] != s){
116         printf("\nError parsing at Position %d!\n", p.pos);
117         exit(0);
118     }
119     else{
120         p.pos++;
121     }
122
123     return p;
124 }

```

## Output - Grammar 1:

Figure 1: Console Output for parsed strings of Grammar 1.



```
vishakan@Legion:~/Desktop/Compiler Design/Ex04

Compiler Design/Ex04 on  main [?]
→ gcc RDP1.c -o r

Compiler Design/Ex04 on  main [?]
→ ./r

Recursive Descent Parser

Enter a string to parse: i+i*i+i+i

Parse Success!

Compiler Design/Ex04 on  main [?] took 10s
→ ./r

Recursive Descent Parser

Enter a string to parse: i***i

Error parsing at Position 2!

Compiler Design/Ex04 on  main [?] took 9s
→
```

## Code - Grammar 2:

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<string.h>
4
5 /*Recursive Descent Parser*/
6
7 /*
8 Grammar: G: E->E+T|E-T|T
9           T->T*F|T/F|F
10          F->(E)|i
11 */
12
13 /*
14 Removed Left Recursion
15 Grammar G': E->TE'
16             E'->+TE'|-TE'|e
17             T->FT'
18             T'->*FT'|/FT'|e
19             F->(E)|i
20 */
21
22 struct parse_struct{
23     char str[100];
24     int pos;
25     int len;
26 };
27
28 typedef struct parse_struct parser;
29
30 parser E(parser p);
31 parser T(parser p);
32 parser EPrime(parser p);
33 parser F(parser p);
34 parser TPrime(parser p);
35 parser parse(parser p, char s);
36
37 int main(void){
38     parser p;
39
40     printf("\n\t\tRecursive Descent Parser\n");
41     printf("\nEnter a string to parse: ");
42     scanf("%s", p.str);
43
44     p.len = strlen(p.str);
45     p.pos = 0;
46
47     p = E(p);
```

```

48
49     if(p.pos == p.len){
50         //All characters have been parsed
51         printf("\nParse Success!\n");
52     }
53
54     else{
55         //Some characters haven't been parsed, but returned to main
56         printf("\nError parsing at Position %d!\n", p.pos);
57     }
58
59     return 0;
60 }
61
62 parser E(parser p){
63     //printf("\nAt E");
64     p = T(p);
65     p = EPrime(p);
66
67     return p;
68 }
69
70 parser T(parser p){
71     //printf("\nAt T");
72     p = F(p);
73     p = TPrime(p);
74
75     return p;
76 }
77
78 parser EPrime(parser p){
79     //printf("\nAt EPrime");
80     if(p.str[p.pos] == '+'){
81         p = parse(p, '+');
82         p = T(p);
83         p = EPrime(p);
84     }
85     else if(p.str[p.pos] == '-'){
86         p = parse(p, '-');
87         p = T(p);
88         p = EPrime(p);
89     }
90
91     return p;
92 }
93
94 parser TPrime(parser p){
95     //printf("\nAt TPrime");
96     if(p.str[p.pos] == '*'){
97         p = parse(p, '*');
98         p = F(p);

```

```

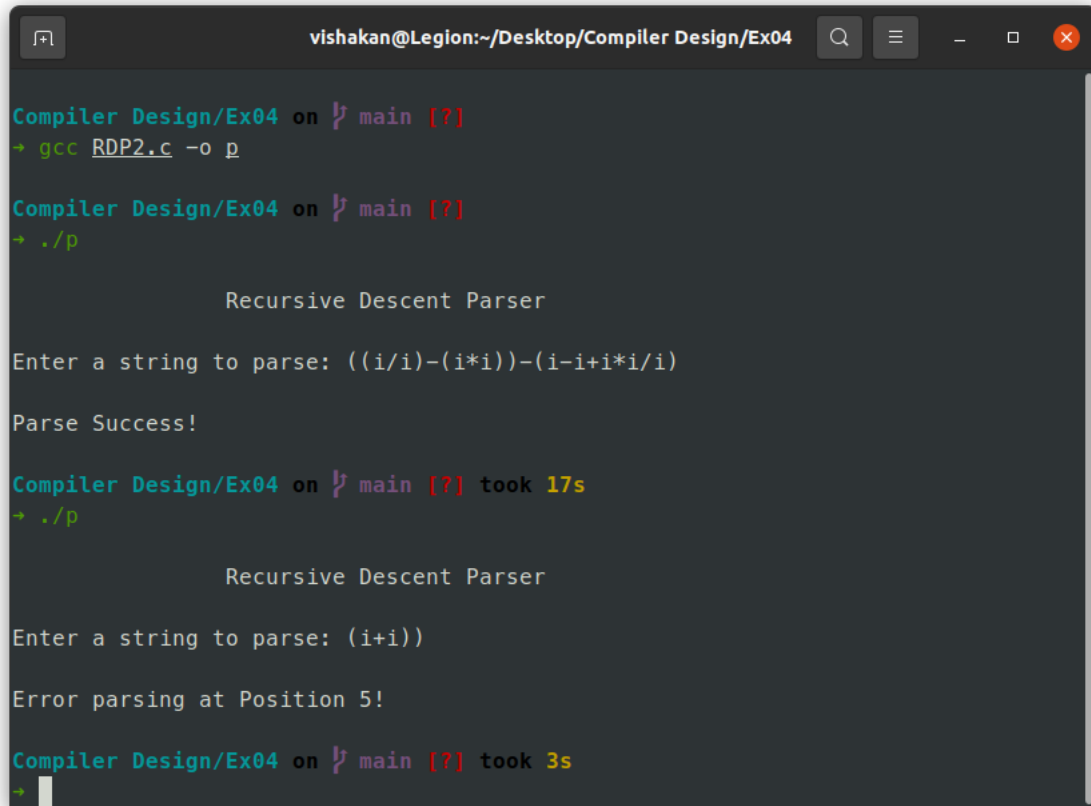
99         p = TPrime(p);
100     }
101     else if(p.str[p.pos] == '/'){
102         p = parse(p, '/');
103         p = F(p);
104         p = TPrime(p);
105     }
106
107     return p;
108 }
109
110 parser F(parser p){
111     //printf("\nAt F");
112     if(p.str[p.pos] == '('){
113         p = parse(p, '(');
114         p = E(p);
115         p = parse(p, ')');
116     }
117     else if(p.str[p.pos] == 'i'){
118         p = parse(p, 'i');
119     }
120     else{
121         printf("\nError parsing at Position %d!\n", p.pos);
122         exit(0);
123     }
124
125     return p;
126 }
127
128 parser parse(parser p, char s){
129     if(p.str[p.pos] != s){
130         printf("\nError parsing at Position %d!\n", p.pos);
131         exit(0);
132     }
133     else{
134         p.pos++;
135     }
136
137     return p;
138 }

```



## Output - Grammar 2:

Figure 2: Console Output for parsed strings of Grammar 2.



```
vishakan@Legion:~/Desktop/Compiler Design/Ex04
Compiler Design/Ex04 on 1 main [?]
→ gcc RDP2.c -o p

Compiler Design/Ex04 on 1 main [?]
→ ./p

Recursive Descent Parser
Enter a string to parse: ((i/i)-(i*i))-(i-i+i*i/i)
Parse Success!

Compiler Design/Ex04 on 1 main [?] took 17s
→ ./p

Recursive Descent Parser
Enter a string to parse: (i+i))
Error parsing at Position 5!

Compiler Design/Ex04 on 1 main [?] took 3s
→
```

## Learning Outcome:

- I understood about the working of a **Recursive Descent Parser**.
- I understood that Recursive Descent Parser, being a Top-Down Parser, does not work with Left-Recursive Grammars.
- I was able to implement a working Recursive Descent Parser for a simple grammar.
- I was able to extend the concept to implement a Recursive Descent Parser for a complicated grammar with more productions.
- I refreshed my concepts with recursion & return handling in functions with C.
- I understood how to manually perform the Recursive Descent Parsing Process.