
A4-Recursive Descent Parser using C

Sneha Sriram Kannan 185001157

26-02-2021

1 Code for input 1

```
1 #include "functions1.h"
2 #include <stdio.h>
3 #include <string.h>
4 #include <stdlib.h>
5
6 typedef struct parsestring
7 {
8     char instr[128];
9     int ptr;
10 } parsestring;
11
12 int substr(char str1[], char str2[]);
13 int checkFunction(char str[]);
14 int EliminateLeftRecursion();
15 int Eliminate(char production[], char newFile[10][128], int count, int pos);
16 int recursiveDescent(char newProduction[10][128], int count);
17 int checkterminal(char ch);
18 void E(parsestring *p, int tab);
19 void E_Dash(parsestring *p, int tab);
20 void T(parsestring *p, int tab);
21 void T_Dash(parsestring *p, int tab);
22 void F(parsestring *p, int tab);
23
24 int main()
25 {
26     EliminateLeftRecursion();
27     return 0;
28 }
29
30 int EliminateLeftRecursion()
31 {
32     char file[10][128];
33     char newProduction[10][128];
34     char functions[128][128];
35     int newProductionCount = 0;
36     int lrcount = 0;
37     char inputfile[30];
38     FILE *fd;
39     fd = fopen("input1.txt", "r");
40     int i = 0;
41     //reading code from a file and storing in an array
42     while (fgets(file[i], sizeof(file[i]), fd))
43         i++;
44
45     printf("=====\n");
46     printf("Input Productions:\n");
```

```

47 printf("=====\n");
48 for (int j = 0; j < i; j++)
49 {
50     printf("%s", file[j]);
51 }
52 printf("\n\n");
53 printf("=====\n");
54 printf("Result of checking for Left Recursion:\n");
55 printf("=====\n");
56 for (int j = 0; j < i; j++)
57 {
58     char lhs = file[j][0];
59     int noLeft = 1;
60     for (int k = 3; k < strlen(file[j]); k++)
61     {
62         if (lhs == file[j][k]) //checking if left recursion occurs in the productions
63         {
64             noLeft = 0;
65             Eliminate(file[j], newProduction, newProductionCount, k);
66             newProductionCount += 2;
67             lrcount++;
68             break;
69         }
70     }
71     if (noLeft == 1) //no LR so no change in the production
72     {
73         strcpy(newProduction[newProductionCount++], file[j]);
74     }
75 }
76 if (lrcount == 0)
77     printf("NO LEFT RECURSION\n");
78 else
79 {
80     for (int j = 0; j < newProductionCount; j++)
81     {
82         printf("%s", newProduction[j]);
83     }
84 }
85 printf("\n\n");
86 recursiveDescent(newProduction, newProductionCount);
87 }
88
89 int Eliminate(char production[], char newFile[10][128], int count, int pos)
90 {
91     char new[3];
92     new[0] = production[0];
93     new[1] = '\';
94     new[2] = '\0';
95     char alpha[20];
96     int j = 0;
97     int betacount = 0;
98     int betapointer = 0;
99     char betaproduction[30];
100     char newProduction[50];
101     sprintf(newProduction, "%c-> ", production[0]);
102     int k = 0;
103     int newFlag = 0;
104     //FINDING A-> beta A
105     for (int i = 3; i < strlen(production); i++)
106     {
107         if (production[i] == '|' || production[i] == '\n') //end of a production
108         {
109             strcat(newProduction, new);
110             betacount++;
111             newFlag = 1;
112         }
113         else if (production[i] != new[0])
114         {
115             if (newFlag == 1) //must concatenate |
116             {
117                 strcat(newProduction, "|");
118                 newFlag = 0;

```

```

119     }
120     char temp[2];
121     temp[0] = production[i];
122     temp[1] = '\0';
123     strcat(newProduction, temp); //Adding character of beta
124 }
125 else
126 { //left recursion position so not beta
127     while (production[i] != '|')
128         i++;
129 }
130 }
131 if (strlen(newProduction) != 4)
132 {
133     strcat(newProduction, "\n");
134     strcpy(newFile[count++], newProduction);
135 }
136 else
137 {
138     strcat(newProduction, new);
139     strcat(newProduction, "\n");
140     strcpy(newFile[count++], newProduction);
141 }
142 //FINDING A'->epsilon|alphaA
143 //finding alpha if there is more than one
144 int alphapos[5];
145 int alphacount = 0;
146 for (int i = pos; i < strlen(production); i++)
147 {
148     if (production[i] == new[0] && (production[i - 1] == '|' || production[i - 1] == '>'))
149         alphapos[alphacount++] = i;
150 }
151 for (int i = 0; i < alphacount; i++)
152     j = 0;
153 char alphaproduction[100];
154 memset(alphaproduction, 0, 100);
155 alphaproduction[0] = new[0];
156 alphaproduction[1] = new[1];
157 alphaproduction[2] = '-';
158 alphaproduction[3] = '>';
159 alphaproduction[4] = ' ';
160 int pointer = 5;
161 for (int i = 0; i < alphacount; i++)
162 {
163     j = alphapos[i] + 1;
164     while (production[j] != '|' && production[j] != '\n')
165     {
166         alphaproduction[pointer] = production[j];
167         pointer++;
168         j++;
169     }
170     strcat(alphaproduction, new);
171     pointer += 2;
172     alphaproduction[pointer++] = '|';
173 }
174 strcat(alphaproduction, "e\n\0");
175 sprintf(newFile[count++], "%s", alphaproduction);
176 return count;
177 }
178
179 int recursiveDescent(char newProduction[10][128], int count)
180 {
181     char instr[128];
182     printf("=====\n");
183     printf("Function Calls and Parsing:\n");
184     printf("=====\n\n");
185     printf("Enter the string to parse:");
186     scanf("%s", instr);
187     parsestring *p = malloc(sizeof(parsestring));
188     strcpy(p->instr, instr);
189     p->ptr = 0;

```

```

190     printf("INSIDE MAIN\n");
191     printf("ENTERED E\n");
192     E(p,1);
193     printf("EXITED E\n");
194     if (p->instr[ptr] == '$')
195     {
196         printf("\nPARSING SUCCESSFUL!!!");
197     }
198     else
199     {
200         printf("ERROR!!");
201         exit(0);
202     }
203 }

```

```

204 void E(parsestring *p,int tab)

```

```

205 {
206     for(int i=0;i<tab;i++)
207         printf("\t");
208     printf("ENTERED T\n");
209     T(p,tab+1);
210     for(int i=0;i<tab;i++)
211         printf("\t");
212     printf("EXITED T\n");
213     for(int i=0;i<tab;i++)
214         printf("\t");
215     printf("ENTERED E'\n");
216     E_Dash(p,tab+1);
217     for(int i=0;i<tab;i++)
218         printf("\t");
219     printf("EXITED E'\n");
220 }

```

```

221 void E_Dash(parsestring *p,int tab)

```

```

222 {
223     if (p->instr[ptr] == '+')
224     {
225         //match +
226         p->ptr += 1;
227         for(int i=0;i<tab;i++)
228             printf("\t");
229         printf("MATCHED +\n");
230         for(int i=0;i<tab;i++)
231             printf("\t");
232         printf("ENTERED T\n");
233         T(p,tab+1);
234         for(int i=0;i<tab;i++)
235             printf("\t");
236         printf("EXITED T\n");
237         for(int i=0;i<tab;i++)
238             printf("\t");
239         printf("ENTERED E'\n");
240         E_Dash(p,tab+1);
241         for(int i=0;i<tab;i++)
242             printf("\t");
243         printf("EXITED E'\n");
244     }
245     else
246         return;
247 }

```

```

248 void T(parsestring *p,int tab)

```

```

249 {
250     for(int i=0;i<tab;i++)
251         printf("\t");
252     printf("ENTERED F\n");
253     F(p,tab+1);
254     for(int i=0;i<tab;i++)
255         printf("\t");
256     printf("EXITED F\n");
257     for(int i=0;i<tab;i++)
258         printf("\t");
259     printf("ENTERED T'\n");
260     T_Dash(p,tab+1);
261 }

```

```

262     for(int i=0;i<tab;i++)
263         printf("\t");
264     printf("EXITED T'\n");
265 }
266 void T_Dash(parsestring *p,int tab)
267 {
268     if (p->instring[p->ptr] == '*')
269     {
270         //match *
271         p->ptr += 1;
272         for(int i=0;i<tab;i++)
273             printf("\t");
274         printf("MATCHED *\n");
275         for(int i=0;i<tab;i++)
276             printf("\t");
277         printf("ENTERED F'\n");
278         F(p,tab+1);
279         for(int i=0;i<tab;i++)
280             printf("\t");
281         printf("EXITED F'\n");
282         for(int i=0;i<tab;i++)
283             printf("\t");
284         printf("ENTERED E'\n");
285         E_Dash(p,tab+1);
286         for(int i=0;i<tab;i++)
287             printf("\t");
288         printf("EXITED E'\n");
289     }
290     else
291         return;
292 }
293 void F(parsestring *p,int tab)
294 {
295     if (p->instring[p->ptr] == 'i')
296     {
297         //match i
298         for(int i=0;i<tab;i++){
299             printf("\t");
300         }
301         printf("MATCHED i\n");
302         p->ptr += 1;
303     }
304     else
305     {
306         printf("ERROR!!");
307         exit(0);
308     }
309 }

```

2 Code for input 2

```

1 #include "functions1.h"
2 #include <stdio.h>
3 #include <string.h>
4 #include <stdlib.h>
5
6 typedef struct parsestring
7 {
8     char instring[128];
9     int ptr;
10 } parsestring;
11
12 int substr(char str1[], char str2[]);
13 int checkFunction(char str[]);
14 int EliminateLeftRecursion();
15 int Eliminate(char production[], char newFile[10][128], int count, int pos);
16 int recursiveDescent(char newProduction[10][128], int count);
17 int checkterminal(char ch);
18 void E(parsestring *p, int tab);
19 void E_Dash(parsestring *p, int tab);
20 void T(parsestring *p, int tab);

```

```

21 void T_Dash(parsestring *p, int tab);
22 void F(parsestring *p, int tab);
23
24 int main()
25 {
26     EliminateLeftRecursion();
27     return 0;
28 }
29
30 int EliminateLeftRecursion()
31 {
32     char file[10][128];
33     char newProduction[10][128];
34     char functions[128][128];
35     int newProductionCount = 0;
36     int lrcount = 0;
37     char inputfile[30];
38     FILE *fd;
39     fd = fopen("input2.txt", "r");
40     int i = 0;
41     //reading code from a file and storing in an array
42     while (fgets(file[i], sizeof(file[i]), fd))
43         i++;
44
45     printf("=====\n");
46     printf("Input Productions:\n");
47     printf("=====\n");
48     for (int j = 0; j < i; j++)
49     {
50         printf("%s", file[j]);
51     }
52     printf("\n\n");
53     printf("=====\n");
54     printf("Result of checking for Left Recursion:\n");
55     printf("=====\n");
56     for (int j = 0; j < i; j++)
57     {
58         char lhs = file[j][0];
59         int noLeft = 1;
60         for (int k = 3; k < strlen(file[j]); k++)
61         {
62             if (lhs == file[j][k]) //checking if left recursion occurs in the productions
63             {
64                 noLeft = 0;
65                 Eliminate(file[j], newProduction, newProductionCount, k);
66                 newProductionCount += 2;
67                 lrcount++;
68                 break;
69             }
70         }
71         if (noLeft == 1) //no LR so no change in the production
72         {
73             strcpy(newProduction[newProductionCount++], file[j]);
74         }
75     }
76     if (lrcount == 0)
77         printf("NO LEFT RECURSION\n");
78     else
79     {
80         for (int j = 0; j < newProductionCount; j++)
81         {
82             printf("%s", newProduction[j]);
83         }
84     }
85     printf("\n\n");
86     recursiveDescent(newProduction, newProductionCount);
87 }
88
89 int Eliminate(char production[], char newFile[10][128], int count, int pos)
90 {
91     char new[3];
92     new[0] = production[0];

```

```

93 new[1] = '\';
94 new[2] = '\0';
95 char alpha[20];
96 int j = 0;
97 int betacount = 0;
98 int betapointer = 0;
99 char betaproduction[30];
100 char newProduction[50];
101 sprintf(newProduction, "%c-> ", production[0]);
102 int k = 0;
103 int newFlag = 0;
104 //FINDING A->beta A
105 for (int i = 3; i < strlen(production); i++)
106 {
107     if (production[i] == '|' || production[i] == '\n') //end of a production
108     {
109         strcat(newProduction, new);
110         betacount++;
111         newFlag = 1;
112     }
113     else if (production[i] != new[0])
114     {
115         if (newFlag == 1) //must concatenate |
116         {
117             strcat(newProduction, "|");
118             newFlag = 0;
119         }
120         char temp[2];
121         temp[0] = production[i];
122         temp[1] = '\0';
123         strcat(newProduction, temp); //Adding character of beta
124     }
125     else
126     { //left recursion position so not beta
127         while (production[i] != '|')
128             i++;
129     }
130 }
131 if (strlen(newProduction) != 4)
132 {
133     strcat(newProduction, "\n");
134     strcpy(newFile[count++], newProduction);
135 }
136 else
137 {
138     strcat(newProduction, new);
139     strcat(newProduction, "\n");
140     strcpy(newFile[count++], newProduction);
141 }
142 //FINDING A'->epsilon|alpha A
143 //finding alpha if there is more than one
144 int alphapos[5];
145 int alphacount = 0;
146 for (int i = pos; i < strlen(production); i++)
147 {
148     if (production[i] == new[0] && (production[i - 1] == '|' || production[i - 1] == '>'))
149         alphapos[alphacount++] = i;
150 }
151 for (int i = 0; i < alphacount; i++)
152     j = 0;
153 char alphaproduction[100];
154 memset(alphaproduction, 0, 100);
155 alphaproduction[0] = new[0];
156 alphaproduction[1] = new[1];
157 alphaproduction[2] = '-';
158 alphaproduction[3] = '>';
159 alphaproduction[4] = ' ';
160 int pointer = 5;
161 for (int i = 0; i < alphacount; i++)
162 {
163     j = alphapos[i] + 1;

```

```

164     while (production[j] != '|' && production[j] != '\n')
165     {
166         alphaproduction[pointer] = production[j];
167         pointer++;
168         j++;
169     }
170     strcat(alphaproduction, new);
171     pointer += 2;
172     alphaproduction[pointer++] = '|';
173 }
174 strcat(alphaproduction, "e\n\0");
175 sprintf(newFile[count++], "%s", alphaproduction);
176 return count;
177 }
178
179 int recursiveDescent(char newProduction[10][128], int count)
180 {
181     char instr[128];
182     printf("=====\n");
183     printf("Function Calls and Parsing:\n");
184     printf("=====\n\n");
185     printf("Enter the string to parse:");
186     scanf("%s", instr);
187     parsestring *p = malloc(sizeof(parsestring));
188     strcpy(p->instr, instr);
189     p->ptr = 0;
190     printf("INSIDE MAIN\n");
191     printf("ENTERED E\n");
192     E(p, 1);
193     printf("EXITED E\n");
194     if (p->instr[p->ptr] == '$')
195     {
196         printf("\nPARSING SUCCESSFUL!!!");
197     }
198     else
199     {
200         printf("ERROR!!");
201         exit(0);
202     }
203 }
204
205 void E(parsestring *p, int tab)
206 {
207     for(int i=0; i<tab; i++)
208         printf("\t");
209     printf("ENTERED T\n");
210     T(p, tab+1);
211     for(int i=0; i<tab; i++)
212         printf("\t");
213     printf("EXITED T\n");
214     for(int i=0; i<tab; i++)
215         printf("\t");
216     printf("ENTERED E'\n");
217     E_Dash(p, tab+1);
218     for(int i=0; i<tab; i++)
219         printf("\t");
220     printf("EXITED E'\n");
221 }
222 void E_Dash(parsestring *p, int tab)
223 {
224     if (p->instr[p->ptr] == '+')
225     {
226         //match +
227         p->ptr += 1;
228         for(int i=0; i<tab; i++)
229             printf("\t");
230         printf("MATCHED +\n");
231         for(int i=0; i<tab; i++)
232             printf("\t");
233         printf("ENTERED T\n");
234         T(p, tab+1);
235         for(int i=0; i<tab; i++)

```



```

236         printf("\t");
237     printf("EXITED T\n");
238     for(int i=0;i<tab;i++)
239         printf("\t");
240     printf("ENTERED E'\n");
241     E_Dash(p,tab+1);
242     for(int i=0;i<tab;i++)
243         printf("\t");
244     printf("EXITED E'\n");
245 }
246 else if (p->instring[p->ptr] == '-')
247 {
248     //match +
249     p->ptr += 1;
250     for(int i=0;i<tab;i++)
251         printf("\t");
252     printf("MATCHED -\n");
253     for(int i=0;i<tab;i++)
254         printf("\t");
255     printf("ENTERED T\n");
256     T(p,tab+1);
257     for(int i=0;i<tab;i++)
258         printf("\t");
259     printf("EXITED T\n");
260     for(int i=0;i<tab;i++)
261         printf("\t");
262     printf("ENTERED E'\n");
263     E_Dash(p,tab+1);
264     for(int i=0;i<tab;i++)
265         printf("\t");
266     printf("EXITED E'\n");
267 }
268 else
269     return;
270 }
271 void T(parsestring *p,int tab)
272 {
273     for(int i=0;i<tab;i++)
274         printf("\t");
275     printf("ENTERED F\n");
276     F(p,tab+1);
277     for(int i=0;i<tab;i++)
278         printf("\t");
279     printf("EXITED F\n");
280     for(int i=0;i<tab;i++)
281         printf("\t");
282     printf("ENTERED T'\n");
283     T_Dash(p,tab+1);
284     for(int i=0;i<tab;i++)
285         printf("\t");
286     printf("EXITED T'\n");
287 }
288 void T_Dash(parsestring *p,int tab)
289 {
290     if (p->instring[p->ptr] == '*')
291     {
292         //match *
293         p->ptr += 1;
294         for(int i=0;i<tab;i++)
295             printf("\t");
296         printf("MATCHED *\n");
297         for(int i=0;i<tab;i++)
298             printf("\t");
299         printf("ENTERED F\n");
300         F(p,tab+1);
301         for(int i=0;i<tab;i++)
302             printf("\t");
303         printf("EXITED F\n");
304         for(int i=0;i<tab;i++)
305             printf("\t");
306         printf("ENTERED T'\n");
307         T_Dash(p,tab+1);

```

```

308     for(int i=0;i<tab;i++)
309         printf("\t");
310     printf("EXITED T'\n");
311 }
312 else if (p->instring[p->ptr] == '/')
313 {
314     //match *
315     p->ptr += 1;
316     for(int i=0;i<tab;i++)
317         printf("\t");
318     printf("MATCHED /\n");
319     for(int i=0;i<tab;i++)
320         printf("\t");
321     printf("ENTERED F\n");
322     F(p,tab+1);
323     for(int i=0;i<tab;i++)
324         printf("\t");
325     printf("EXITED F\n");
326     for(int i=0;i<tab;i++)
327         printf("\t");
328     printf("ENTERED T'\n");
329     T_Dash(p,tab+1);
330     for(int i=0;i<tab;i++)
331         printf("\t");
332     printf("EXITED T'\n");
333 }
334 else
335     return;
336 }
337 void F(parsestring *p,int tab)
338 {
339     if (p->instring[p->ptr] == '(')
340     {
341         //match *
342         p->ptr += 1;
343         for(int i=0;i<tab;i++)
344             printf("\t");
345         printf("MATCHED (\n");
346         for(int i=0;i<tab;i++)
347             printf("\t");
348         printf("ENTERED E\n");
349         E(p,tab+1);
350         for(int i=0;i<tab;i++)
351             printf("\t");
352         printf("EXITED E\n");
353         if (p->instring[p->ptr] == ')')
354         {
355             for(int i=0;i<tab;i++)
356                 printf("\t");
357             printf("MATCHED )\n");
358         }
359         else{
360             printf("ERROR!!");
361             exit(0);
362         }
363     }
364     else if (p->instring[p->ptr] == 'i')
365     {
366         //match i
367         for(int i=0;i<tab;i++){
368             printf("\t");
369         }
370         printf("MATCHED i\n");
371         p->ptr += 1;
372     }
373     else
374     {
375         printf("ERROR!!");
376         exit(0);
377     }
378 }
379 }

```

3 Output Screenshots

```
snehakannan@pop-os:~/Sneha/Semester 6/Compiler Design/Lab/A4$ ./a
=====
Input Productions:
=====
E->E+T|T
T->T*F|F
F->i

=====
Result of checking for Left Recursion:
=====
E-> TE'
E'-> +TE'|e
T-> FT'
T'-> *FT'|e
F->i

=====
Function Calls and Parsing:
=====

Enter the string to parse:i+i*i$
INSIDE MAIN
ENTERED E
    ENTERED T
        ENTERED F
            MATCHED i
        EXITED F
        ENTERED T'
        EXITED T'
    EXITED T
    ENTERED E'
        MATCHED +
        ENTERED T
            ENTERED F
                MATCHED i
            EXITED F
            ENTERED T'
                MATCHED *
                ENTERED F
                    MATCHED i
                EXITED F
                ENTERED E'
                EXITED E'
            EXITED T'
        EXITED T
        ENTERED E'
        EXITED E'
    EXITED E'
EXITED E

snehakannan@pop-os:~/Sneha/Semester 6/Compiler Design/Lab/A4$
```

Figure 1: Output for grammar 1 success case

```

snehakannan@pop-os:~/Sneha/Semester 6/Compiler Design/Lab/A4$ ./a
=====
Input Productions:
=====
E->E+T|T
T->T*F|F
F->i

=====
Result of checking for Left Recursion:
=====
E-> TE'
E'-> +TE'|e
T-> FT'
T'-> *FT'|e
F->i

=====
Function Calls and Parsing:
=====

Enter the string to parse:i+i*i$
INSIDE MAIN
ENTERED E
    ENTERED T
        ENTERED F
            MATCHED i
        EXITED F
        ENTERED T'
        EXITED T'
    EXITED T
    ENTERED E'
        MATCHED +
        ENTERED T
            ENTERED F
                MATCHED i
            EXITED F
            ENTERED T'
                MATCHED *
                ENTERED F
                    MATCHED i
                EXITED F
                ENTERED E'
                EXITED E'
            EXITED T'
        EXITED T
        ENTERED E'
        EXITED E'
    EXITED E'
EXITED E

snehakannan@pop-os:~/Sneha/Semester 6/Compiler Design/Lab/A4$

```

Figure 2: Output for grammar 1 failure case

```

=====
Input Productions:
=====
E->E+T|E-T|T
T->T*F|T/F|F
F->(E)|i

=====
Result of checking for Left Recursion:
=====
E-> TE'
E'-> +TE'|-TE'|e
T-> FT'
T'-> *FT'|/FT'|e
F->(E)|i

=====
Function Calls and Parsing:
=====

Enter the string to parse:i/i+i-i$
INSIDE MAIN
ENTERED E
    ENTERED T
        ENTERED F
            MATCHED i
        EXITED F
        ENTERED T'
            MATCHED /
            ENTERED F
                MATCHED i
            EXITED F
            ENTERED T'
                EXITED T'
        EXITED T'
    EXITED T
    ENTERED E'
        MATCHED +
        ENTERED T
            ENTERED F
                MATCHED i
            EXITED F
            ENTERED T'
                EXITED T'
        EXITED T
        ENTERED E'
            MATCHED -
            ENTERED T
                ENTERED F
                    MATCHED i
                EXITED F
                ENTERED T'
                    EXITED T'
            EXITED T
            ENTERED E'
                EXITED E'
        EXITED E'
    EXITED E'
EXITED E

snehakannan@pop-os:~/Sneha/Semester 6/Compiler Design/Lab/A4$

```

Figure 3: Output for grammar 2 success case

```

snehakannan@pop-os:~/Sneha/Semester 6/Compiler Design/Lab/A4$ ./b
=====
Input Productions:
=====
E->E+T|E-T|T
T->T*F|T/F|F
F->(E)|i

=====
Result of checking for Left Recursion:
=====
E-> TE'
E'-> +TE'|-TE'|e
T-> FT'
T'-> *FT'|/FT'|e
F->(E)|i

=====
Function Calls and Parsing:
=====

Enter the string to parse:i++i/i
INSIDE MAIN
ENTERED E
    ENTERED T
        ENTERED F
            MATCHED i
            EXITED F
        ENTERED T'
        EXITED T'
    EXITED T
    ENTERED E'
        MATCHED +
        ENTERED T
            ENTERED F
            ERROR!! snehakannan@pop-os:~/Sneha/Semester 6/Compiler Design/Lab/A4$

```

Figure 4: Output for grammar 2 failure case