Practical 5: Write a program in Lex /Yacc to create LR(1) Parser

Gahan Saraiya (18MCEC10)

18mcec10@nirmauni.ac.in

I. AIM

Write a program in Lex /Yacc to create LR(1) Parser

II. Implementation

I. C file

```c
#include <stdio.h>
#include <stdlib.h>
#define t_shift 0
#define t_reduce 1
#define t_accept 2
#define t_blank 3

struct action{
        int type;
        int value;
};

struct goto_table{
        char *symbols;
        int **table;
};

struct action_table{
        char *symbols;
        struct action **table;
};

struct lr_table{
        int num_states;
        int num_nonterm;
        int num_term;
```

```
27          struct action_table at;
28          struct goto_table gt;
29  };
30
31  int char_to_col(char c, char* ca, int len){
32          for(int i=0; i<len; ++i){
33                  if(ca[i]==c) return i;
34          }
35          return -1;
36  }
37
38  void PrintTable(struct lr_table* lrt){
39          printf("action table:\n");
40          for(int i=0; i<lrt->num_term; ++i){
41                  printf(" %c ", lrt->at.symbols[i]);
42          }
43          printf("\n");
44          for(int i=0; i<lrt->num_states; ++i){
45                  for(int j=0; j<lrt->num_term; ++j){
46                          int type = lrt->at.table[i][j].type;
47                          if(type==t_shift) printf("s%d ",
                             ↪  lrt->at.table[i][j].value);
48                          else if(type == t_reduce) printf("r%d ",
                             ↪  lrt->at.table[i][j].value);
49                          else if(type == t_accept) printf(" a");
50                          else printf("   ");
51                  }
52                  printf("\n");
53          }
54          printf("goto table:\n");
55          for(int i=0; i<lrt->num_nonterm; ++i){
56                  printf("%c ", lrt->gt.symbols[i]);
57          }
58          printf("\n");
59          for(int i=0; i<lrt->num_states; ++i){
60                  for(int j=0; j<lrt->num_nonterm; ++j){
61                          int val = lrt->gt.table[i][j];
62                          if(val==-1) printf("  ");
63                          else printf("%d ", val);
64                  }
65                  printf("\n");
66          }
```

```
}

void PrintTableNice(struct lr_table* lrt){
        printf("\nTable:\n");
        printf("|       |");
        for(int i=0; i<lrt->num_term; ++i){
                printf("%c\t", lrt->at.symbols[i]);
        }
        printf("|");
        for(int i=0; i<lrt->num_nonterm; ++i){
                printf(" %c\t", lrt->gt.symbols[i]);
        }
        printf("|\n");
        int type;
        for(int i=0; i<lrt->num_states; ++i){
                printf("|   %2d  |", i);
                for(int j=0; j<lrt->num_term; ++j){
                        type = lrt->at.table[i][j].type;
                        if(type==t_shift) printf("s%d\t",
                         ↪  lrt->at.table[i][j].value);
                        else if(type == t_reduce) printf("r%d\t",
                         ↪  lrt->at.table[i][j].value);
                        else if(type == t_accept) printf(" a\t");
                        else printf("\t");
                }
                printf("|");
                for(int j=0; j<lrt->num_nonterm; ++j){
                        int val = lrt->gt.table[i][j];
                        if(val==-1) printf("  \t");
                        else printf("%2d\t", val);
                }
                printf("|\n");
        }
        printf("\n");
}

char* appendToCharArray(char c, char* array, int len){
        if(array==NULL){
                char *cp = (char*)malloc(sizeof(char));
                *cp = c;
                return cp;
        }
```

```
107         char *cp = (char *)malloc(len+1);
108         for(int i=0; i<len; ++i){
109                 *(cp+i) + *(array+i);
110         }
111         *(cp+len) = c;
112         free(array);
113         return cp;
114 }
115
116 int discardable(char c){
117         if(c=='\t' || c=='\n' || c==' ') return 1;
118         return 0;
119 }
120
121 struct lr_table* CreateTable(){
122         int k;
123         struct lr_table* lrt = (struct lr_table*)malloc(sizeof(struct lr_table));
124         printf("How many non-terms are there ?: ");
125         scanf("\n%d", &lrt->num_nonterm);
126         printf("How many terminals are there ?: ");
127         scanf("\n%d", &lrt->num_term);
128         printf("How many states are there ?: ");
129         scanf("\n%d", &lrt->num_states);
130
131         // Enter non terminals
132         lrt->gt.symbols = (char*)malloc(lrt->num_nonterm * sizeof(char));
133         printf("Enter non terminals: ");
134         char c;
135         for(int i=0; i<lrt->num_nonterm; ++i){
136                 scanf("%c", &c);
137                 if(discardable(c)){
138                         i--;
139                         continue;
140                 }
141                 lrt->gt.symbols[i] = c;
142         }
143
144         // Enter terminals
145         lrt->at.symbols = (char*)malloc(lrt->num_term * sizeof(char));
146         printf("Enter terminals: ");
147         for(int i=0; i<lrt->num_term; ++i){
148                 scanf("%c", &c);
```

```
149                 if(discardable(c)){
150                         i--;
151                         continue;
152                 }
153                 lrt->at.symbols[i] = c;
154         }
155 
156         // Enter action table
157         printf("Enter action table in matrix form: 00=blank, si=shift i,
            ↪  ri=reduce i, a0=accept\n");
158         lrt->at.table = (struct action**)malloc(lrt->num_states * sizeof(struct
            ↪  action*));
159         int type;
160         for(int i=0; i<lrt->num_states; ++i){
161                 lrt->at.table[i] = (struct action*)malloc(lrt->num_term *
                    ↪  sizeof(struct action));
162                 for(int j=0; j<lrt->num_term; ++j){
163                         scanf(" %c%d", &c, &k);
164                         if(c=='s') type = t_shift;
165                         else if(c=='r') type = t_reduce;
166                         else if(c=='a') type = t_accept;
167                         else type = t_blank;
168                         lrt->at.table[i][j].type = type;
169                         lrt->at.table[i][j].value = k;
170                 }
171         }
172 
173         // Enter goto table
174         printf("Enter goto table in matrix form: -1=blank\n");
175         lrt->gt.table = (int **)malloc(lrt->num_states * sizeof(int *));
176         for(int i=0; i<lrt->num_states; ++i){
177                 lrt->gt.table[i] = (int*)malloc(lrt->num_nonterm * sizeof(int));
178                 for(int j=0; j<lrt->num_nonterm; ++j){
179                         scanf(" %d", &k);
180                         lrt->gt.table[i][j] = k;
181                 }
182         }
183 
184         return lrt;
185 }
186 #define explen 100
187 
```

```
188 struct rule{
189         char c;
190         int n;
191 };
192
193 struct rule* rules;
194
195 int* append_int(int n, int *arr, int *p);
196 int getReduction(int k);
197 char getRedChar(int k);
198 void printStack(int* stack, int n);
199 struct rule* appendRule(struct rule r, struct rule* _rules, int p);
200 void printBuffer(int buffer[], int buf_pos);
201 void reduceBuffer(int buffer[], int* buf_pos, int rule);
202
203 int main(){
204
205         int num_rules;
206         printf("How many rules are there ?: ");
207         scanf("\n%d", &num_rules);
208         printf("Enter rules properties: left(symbol) right(count). Eg.
          ↪ {A->Aa}=>{A 2}");
209         rules = (struct rule*)malloc(5*sizeof(struct rule));
210         char lhs; int rhs;
211         struct rule r;
212         for(int i=0; i<num_rules; ++i){
213                 scanf("\n%c %d", &lhs, &rhs);
214                 r.c = lhs;
215                 r.n = rhs;
216                 rules = appendRule(r, rules, i);
217         }
218         struct lr_table* lrt = CreateTable();
219         // PrintTable(lrt);
220         PrintTableNice(lrt);
221
222         // Scan expression
223         char expr[explen];
224         scanf("%s", expr);
225         printf("Expression: %s\n", expr);
226
227         char c;
228         int i,j;
```

```
229         int state = 0;
230         struct action act;
231         int* stack = (int*)malloc(5*sizeof(int));
232         int stack_ptr = 1;
233         int red;
234         stack[0] = state;
235         int buffer[100];
236         int buf_pos=-1;
237         printf("stack: ");
238         printStack(stack, stack_ptr);
239         for(i=0; expr[i]!='\0'; ++i){
240                 c = expr[i];
241                 j = char_to_col(c, lrt->at.symbols, lrt->num_term);
242                 act = lrt->at.table[state][j];
243                 switch (act.type)
244                 {
245                         case t_accept:
246                                 reduceBuffer(buffer, &buf_pos, 1);
247                                 printf("Accepted\n");
248                                 return 0;
249                                 break;
250                         case t_shift:
251                                 printf("shift: s%d\n", act.value);
252                                 state = act.value;
253                                 stack = append_int(c, stack, &stack_ptr);
254                                 stack = append_int(state, stack, &stack_ptr);
255                                 int k = c - '0';
256                                 if(k==0 || k==1){
257                                         buf_pos++;
258                                         buffer[buf_pos] = k;
259                                 }
260                                 printf("stack: ");
261                                 printStack(stack, stack_ptr);
262                                 printBuffer(buffer, buf_pos);
263                                 break;
264                         case t_reduce:
265                                 printf("Rduce: r%d\n", act.value);
266                                 red = getReduction(act.value);
267                                 c = getRedChar(act.value);
268                                 stack_ptr -= red*2;
269                                 if(stack_ptr<0){
270                                         printf("Error!\n");
```

```
271                                     return 0;
272                                 }
273                                 reduceBuffer(buffer, &buf_pos, act.value);
274                                 printBuffer(buffer, buf_pos);
275                                 stack = append_int(c, stack, &stack_ptr);
276                                 j = char_to_col(c, lrt->gt.symbols,
                                     ↪ lrt->num_nonterm);
277                                 state = stack[stack_ptr-2];
278                                 state = lrt->gt.table[state][j];
279                                 stack = append_int(state, stack, &stack_ptr);
280                                 printf("stack: ");
281                                 printStack(stack, stack_ptr);
282                                 i--;
283                                 break;
284                         default:
285                                 printf("Error!\n");
286                                 return 0;
287                                 break;
288                 }
289         }
290         return 0;
291 }
292
293 int* append_int(int n, int *arr, int *p){
294         if(*p>0 && *p%5==0){
295                 int* a = (int*)malloc((*p+5)*sizeof(int));
296                 for(int i=0; i<*p; ++i){
297                         a[i] = arr[i];
298                 }
299                 a[*p] = n;
300                 free(arr);
301                 *p = *p + 1;
302                 return a;
303         }
304         arr[*p] = n;
305         *p = *p + 1;
306         return arr;
307 }
308
309 int getReduction(int k){
310         return rules[k-1].n;
311 }
```

```
312
313 char getRedChar(int k){
314         return rules[k-1].c;
315 }
316
317 void printStack(int* stack, int n){
318         for(int i=0; i<n; ++i){
319                 if((i&1)==0){
320                         printf("%d ", stack[i]);
321                 }
322                 else printf("%c ", stack[i]);
323         }
324         printf("\n");
325 }
326
327 struct rule* appendRule(struct rule r, struct rule* _rules, int p){
328         if(p>0 && p%5==0){
329                 struct rule* array = (struct rule*)malloc((p+5)*sizeof(struct
                    ↪  rule));
330                 for(int i=0; i<p; ++i){
331                         array[i] = _rules[i];
332                 }
333                 array[p] = r;
334                 return array;
335         }
336         _rules[p] = r;
337         return _rules;
338 }
339
340 void printBuffer(int buffer[], int buf_pos){
341         printf("\t\t\t\tbuffer: ");
342         for(int i=0; i<=buf_pos; ++i){
343                 printf("%d ", buffer[i]);
344         }
345         printf("\n");
346 }
347
348 void reduceBuffer(int buffer[], int* buf_pos, int rule){
349         int pos = *buf_pos;
350         switch (rule)
351         {
352                 case 1:
```

```
353                         printf("S = %d\n", buffer[pos]);
354                         break;
355                 case 2:
356                         buffer[pos-1] = buffer[pos-1] + buffer[pos];
357                         *buf_pos = *buf_pos - 1;
358                         printf("E = %d\n", buffer[pos-1]);
359                         break;
360                 case 3:
361                         buffer[pos-1] = buffer[pos-1] * buffer[pos];
362                         *buf_pos = *buf_pos - 1;
363                         printf("E = %d\n", buffer[pos-1]);
364                         break;
365                 case 4:
366                         printf("E = %d\n", buffer[pos]);
367                         break;
368                 case 5:
369                         printf("B = 0\n");
370                         break;
371                 case 6:
372                         printf("B = 1\n");
373                         break;
374                 default:
375                         break;
376         }
377 }
```

II. Input

```
6
S 1
E 3
E 3
E 1
B 1
B 1
3
5
9
S E B
+ * 0 1 $
00 00 s3 s4 00
```

```
s5 s6 00 00 a0
r4 r4 00 00 r4
r5 r5 00 00 r5
r6 r6 00 00 r6
00 00 s3 s4 00
00 00 s3 s4 00
r2 r2 00 00 r2
r3 r3 00 00 r3
-1 1 2
-1 -1 -1
-1 -1 -1
-1 -1 -1
-1 -1 -1
-1 -1 7
-1 -1 8
-1 -1 -1
-1 -1 -1
1*1+1+1$
```

II.1 Output

```
How many rules are there ?: Enter rules properties: left(symbol) right(count).
 ↪  Eg. {A->Aa}=>{A 2}How many non-terms are there ?: How many terminals are
 ↪  there ?: How many states are there ?: Enter non terminals: Enter terminals:
 ↪  Enter action table in matrix form: 00=blank, si=shift i, ri=reduce i,
 ↪  a0=accept
Enter goto table in matrix form: -1=blank

Table:
|       |+        *        0        1        $        | S         E
 ↪  B        |
|    0  |               s3        s4               |           1
 ↪  2        |
|    1  |s5        s6                         a        |
 ↪          |
|    2  |r4        r4                         r4        |
 ↪          |
|    3  |r5        r5                         r5        |
 ↪          |
```

```
|    4  |r6        r6                          r6        |
 ↪          |
|    5  |                s3        s4                |
 ↪  7        |
|    6  |                s3        s4                |
 ↪  8        |
|    7  |r2        r2                          r2        |
 ↪          |
|    8  |r3        r3                          r3        |
 ↪          |

Expression: 1*1+1+1$
stack: 0
shift: s4
stack: 0 1 4
                                buffer: 1
Rduce: r6
B = 1
                                buffer: 1
stack: 0 B 2
Rduce: r4
E = 1
                                buffer: 1
stack: 0 E 1
shift: s6
stack: 0 E 1 * 6
                                buffer: 1
shift: s4
stack: 0 E 1 * 6 1 4
                                buffer: 1 1
Rduce: r6
B = 1
                                buffer: 1 1
stack: 0 E 1 * 6 B 8
Rduce: r3
E = 1
                                buffer: 1
stack: 0 E 1
shift: s5
stack: 0 E 1 + 5
                                buffer: 1
```

```
shift: s4
stack: 0 E 1 + 5 1 4
                                buffer: 1 1
Rduce: r6
B = 1
                                buffer: 1 1
stack: 0 E 1 + 5 B 7
Rduce: r2
E = 2
                                buffer: 2
stack: 0 E 1
shift: s5
stack: 0 E 1 + 5
                                buffer: 2
shift: s4
stack: 0 E 1 + 5 1 4
                                buffer: 2 1
Rduce: r6
B = 1
                                buffer: 2 1
stack: 0 E 1 + 5 B 7
Rduce: r2
E = 3
                                buffer: 3
stack: 0 E 1
S = 3
Accepted
```