

1.

Output:

```
C:\Users\vangj\source\repos\332hwk7\x64\Debug\332hwk7.exe
Please enter a string with $ at the end: (i+i)*i$
$QR)E
$QR)QR
$QR)QT
$QR)QR
$QR
$QRF
$QR
***Input string is ACCEPTED by the grammar***
```

```
C:\Users\vangj\source\repos\332hwk7\x64\Debug\332hwk7.exe
Please enter a string with $ at the end: i*(i-i)$
$QR
$QRF
$QR)E
$QR)QR
$QR)QT
$QR)QR
$QR
***Input string is ACCEPTED by the grammar***
Would you like to continue(y/n): _
```

```
C:\Users\vangj\source\repos\332hwk7\x64\Debug\332hwk7.exe
Please enter a string with $ at the end: i(i+i)$
$QR

***Input string is REJECTED by the grammar***

Would you like to continue(y/n): _
```

Code:

```
1 // =====
2 // Group Names: Jay Vang, Jesse Shaihor, Cristian Salinas
3 // Homework 7
4 // Due Date: Mar 23,2023
5 // Purpose:
6 // =====
7
8 #include <iostream>
9 #include <vector>
10 #include <string>
11 using namespace std;
12
13 //function prototypes
14 void readCFG(string expression);
15 string translation(char token, char match);
16
17 // === main =====
18 // Purpose: to ask the user for the input string and then see if it is traceable
19 // =====
20 int main()
21 {
22     string input;
23     string choice = "y";
24
25     //loop until user says no
26     while (choice[0] == 'y')
27     {
28         //get the input string
29         cout << "Please enter a string with $ at the end: ";
30         getline(&_Istr, cin, &_Str, input);
31
32         //call readCFG to trace the input string
33         readCFG(expression: input);
34
35         //ask if your would like to continue
36         cout << "\nWould you like to continue(y/n): ";
37         getline(&_Istr, cin, &_Str, choice);
38         cout << endl;
```

```

39     }
40     return 0;
41 }
42 }
43
44
45
46 // === readCFG =====
47 // Purpose: will try to trace the input string if possible and output if it is rejected or accepted
48 // to the user.
49 // =====
50
51 void readCFG(string expression)
52 {
53     int counter = 0;
54     char state; //tells use what row of the parsing tree we will be in
55     char cfgChange = expression[counter]; //holds the variable we need to match from the user
56     vector<char> stackTrace; //a vector to hold the stack of the trace
57     string temp = "$E"; //holds the change needed to be made to trace also initializes trace
58
59     //loop for duration of the input size
60     while (counter < expression.length())
61     {
62         //if we do not get 1(lambda) then we wrtie the change to trace
63         if (temp != "1")
64         {
65             //will write the stack change to trace
66             for (int i = 0; i < temp.length(); i++)
67             {
68                 stackTrace.push_back(temp[i]);
69             }
70         }
71
72         //the state we need to be in to make the change then delete it
73         state = stackTrace.back();
74         stackTrace.pop_back();

```

```

75
76 //when a match occurs
77 if (state == cfgChange)
78 {
79     //read the next element needed to be trace
80     counter = counter + 1;
81
82     //display the trace tree
83     for (int j = 0; j < stackTrace.size(); j++)
84     {
85         cout << stackTrace[j];
86     }
87     cout << "\n" << endl;
88
89     //set the next item needed to be match
90     cfgChange = expression[counter];
91
92     if (!stackTrace.empty())
93     {
94         state = stackTrace.back();
95         stackTrace.pop_back();
96     }
97 }
98
99 //a switch to decide what we will translate
100 switch (state)
101 {
102     case 'E':
103         temp = translation(token: state, match: cfgChange);
104         break;
105     case 'Q':
106         temp = translation(token: state, match: cfgChange);
107         break;
108     case 'T':
109         temp = translation(token: state, match: cfgChange);
110         break;

```

```

111     case 'R':
112         temp = translation(token: state, match: cfgChange);
113         break;
114     case 'F':
115         temp = translation(token: state, match: cfgChange);
116         break;
117     }
118
119     //if temp is ever '0' then we will break out of the loop and let the user know it is rejected
120     if (temp == "0")
121     {
122         cout << "***Input string is REJECTED by the grammar***\n";
123         break;
124     }
125
126 }
127
128 //if valid is true then let the user know
129 if (temp != "0")
130 {
131     cout << "***Input string is ACCEPTED by the grammar***\n";
132 }
133
134 }
135
136 // === translation =====
137 // Purpose: this function will see if the variable can be translated into the correct string and if
138 // it is traceable.
139 // =====
140
141 string translation(char token, char match)
142 {
143     string change;    //the translation for the trace to perform
144
145     //enter this block if 'E' is the token
146     if (token == 'E')

```

```
147 {
148     //the possible translations for 'E'
149     switch (match)
150     {
151     case 'i':
152         change = "QT";
153         break;
154     case '+':
155         change = "0";
156         break;
157     case '-':
158         change = "0";
159         break;
160     case '*':
161         change = "0";
162         break;
163     case '/':
164         change = "0";
165         break;
166     case '(':
167         change = "QT";
168         break;
169     case ')':
170         change = "0";
171         break;
172     case '$':
173         change = "0";
174         break;
175     default:
176         change = "0";
177         break;
178     }
179 }
180
181 //enter this block if 'Q' is the token
182 if (token == 'Q')
```

```
183     {
184         //the possible translations for 'Q'
185         switch (match)
186         {
187             case 'i':
188                 change = "0";
189                 break;
190             case '+':
191                 change = "QT+";
192                 break;
193             case '-':
194                 change = "QT-";
195                 break;
196             case '*':
197                 change = "0";
198                 break;
199             case '/':
200                 change = "0";
201                 break;
202             case '(':
203                 change = "0";
204                 break;
205             case ')':
206                 change = "1";
207                 break;
208             case '$':
209                 change = "1";
210                 break;
211             default:
212                 change = "0";
213                 break;
214         }
215     }
216
217     //enter this block if 'T' is the token
218     if (token == 'T')
```

```
219 {
220     //the possible translations for 'T'
221     switch (match)
222     {
223     case 'i':
224         change = "RF";
225         break;
226     case '+':
227         change = "0";
228         break;
229     case '-':
230         change = "0";
231         break;
232     case '*':
233         change = "0";
234         break;
235     case '/':
236         change = "0";
237         break;
238     case '(':
239         change = "RF";
240         break;
241     case ')':
242         change = "0";
243         break;
244     case '$':
245         change = "0";
246         break;
247     default:
248         change = "0";
249         break;
250     }
251 }
252
253 //enter this block if 'R' is the token
254 if (token == 'R')
```



```
255     {
256         //the possible translations for 'R'
257         switch (match)
258         {
259             case 'i':
260                 change = "0";
261                 break;
262             case '+':
263                 change = "1";
264                 break;
265             case '-':
266                 change = "1";
267                 break;
268             case '*':
269                 change = "RF*";
270                 break;
271             case '/':
272                 change = "RF/";
273                 break;
274             case '(':
275                 change = "0";
276                 break;
277             case ')':
278                 change = "1";
279                 break;
280             case '$':
281                 change = "1";
282                 break;
283             default:
284                 change = "0";
285                 break;
286         }
287     }
288
289     //enter this block if 'F' is the token
290     if (token == 'F')
```

```
290     if (token == 'F')
291     {
292         //the possible translations for 'F'
293         switch (match)
294         {
295             case 'i':
296                 change = "i";
297                 break;
298             case '+':
299                 change = "0";
300                 break;
301             case '-':
302                 change = "0";
303                 break;
304             case '*':
305                 change = "0";
306                 break;
307             case '/':
308                 change = "0";
309                 break;
310             case '(':
311                 change = ")E(";
312                 break;
313             case ')':
314                 change = "0";
315                 break;
316             case '$':
317                 change = "0";
318                 break;
319             default:
320                 change = "0";
321                 break;
322         }
323     }
324
325     return change;
326
327 }
```

2.

Output:

```
C:\Users\vangj\source\repos\332hwk7\x64\Debug\332hwk7.exe
Please enter a string with $ at the end: (a + a)*a$
$QR)E
$QR)QR
***Input string is REJECTED by the grammar***
Would you like to continue(y/n):
```

```
Please enter a string with $ at the end: a*(a-a)$
$QR
$QRF
$QR)E
$QR)QR
$QR)QT
$QR)QR
$QR
***Input string is ACCEPTED by the grammar***
Would you like to continue(y/n): _
```

```
Please enter a string with $ at the end: (a+a)a$
$QR)E
$QR)QR
$QR)QT
$QR)QR
$QR
***Input string is REJECTED by the grammar***
Would you like to continue(y/n): _
```

Code:

```
1 // =====
2 // Group Names: Jay Vang, Jesse Shaihor, Cristian Salinas
3 // Homework 7
4 // Due Date: Mar 23,2023
5 // Purpose:
6 // =====
7
8 #include <iostream>
9 #include <vector>
10 #include<string>
11 using namespace std;
12
13 //function prototypes
14 void readCFG(string expression);
15 string translation(char token, char match);
16
17 // == main =====
18 // Purpose: to ask the user for the input string and then see if it is traceable
19 // =====
20 int main()
21 {
22     string input;
23     string choice = "y";
24
25     //loop until user says no
26     while (choice[0] == 'y')
27     {
28         //get the input string
29         cout << "Please enter a string with $ at the end: ";
30         getline(&_Istr, cin, &_Str, input);
31
32         //call readCFG to trace the input string
33         readCFG(expression: input);
34
35         //ask if you would like to continue
36         cout << "\nWould you like to continue(y/n): ";
37         getline(&_Istr, cin, &_Str, choice);
38         cout << endl;
39     }
40     return 0;
41 }
42
```

```

43
44
45
46 // === readCFG =====
47 // Purpose: will try to trace the input string if possible and output if it is rejected or accepted
48 // to the user.
49 // =====
50
51 void readCFG(string expression)
52 {
53     int counter = 0;
54     char state; //tells use what row of the parsing tree we will be in
55     char cfgChange = expression[counter]; //holds the variable we need to match from the user
56     vector<char> stackTrace; //a vector to hold the stack of the trace
57     string temp = "$E"; //holds the change needed to be made to trace also initializes trace
58
59     //loop for duration of the input size
60     while (counter < expression.length())
61     {
62         //if we do not get 1(lambda) then we write the change to trace
63         if (temp != "1")
64         {
65             //will write the stack change to trace
66             for (int i = 0; i < temp.length(); i++)
67             {
68                 stackTrace.push_back(_Val: temp[i]);
69             }
70         }
71
72         //the state we need to be in to make the change then delete it
73         state = stackTrace.back();
74         stackTrace.pop_back();
75
76         //when a match occurs
77         if (state == cfgChange)
78         {
79             //read the next element needed to be trace
80             counter = counter + 1;
81

```

```

82 //display the trace tree
83 for (int j = 0; j < stackTrace.size(); j++)
84 {
85     cout << stackTrace[j];
86 }
87 cout << "\n" << endl;
88
89 //set the next item needed to be match
90 cfgChange = expression[counter];
91
92 if (!stackTrace.empty())
93 {
94     state = stackTrace.back();
95     stackTrace.pop_back();
96 }
97
98
99 //a switch to decide what we will translate
100 switch (state)
101 {
102     case 'S':
103         temp = translation(token: state, match: cfgChange);
104         break;
105     case 'W':
106         temp = translation(token: state, match: cfgChange);
107         break;
108     case 'E':
109         temp = translation(token: state, match: cfgChange);
110         break;
111     case 'Q':
112         temp = translation(token: state, match: cfgChange);
113         break;
114     case 'T':
115         temp = translation(token: state, match: cfgChange);
116         break;
117     case 'R':
118         temp = translation(token: state, match: cfgChange);
119         break;
120     case 'F':
121         temp = translation(token: state, match: cfgChange);
122         break;
123 }

```

```

124
125 //if temp is ever '0' then we will break out of the loop and let the user know it is rejected
126 if (temp == "0")
127 {
128     cout << "***Input string is REJECTED by the grammar***\n";
129     break;
130 }
131
132 }
133
134 //if valid is true then let the user know
135 if (temp != "0")
136 {
137     cout << "***Input string is ACCEPTED by the grammar***\n";
138 }
139
140 }
141
142 // === translation =====
143 // Purpose: this function will see if the varaible can be translated into the correct string and if
144 // it is tracable.
145 // =====
146
147 string translation(char token, char match)
148 {
149     string change; //the translation for the trace to perform
150
151     //enter this block if 'S' is the token
152     if (token == 'S')
153     {
154         //the possible translations for 'S'
155         switch (match)
156         {
157             case 'a':
158                 change = "Wa";
159                 break;
160             case '+':
161                 change = "0";
162                 break;
163             case '-':
164                 change = "0";
165                 break;

```

```

165         break;
166     case '*':
167         change = "0";
168         break;
169     case '/':
170         change = "0";
171         break;
172     case '(':
173         change = "0";
174         break;
175     case ')':
176         change = "0";
177         break;
178     case '$':
179         change = "0";
180         break;
181     case '=':
182         change = "0";
183         break;
184     default:
185         change = "0";
186         break;
187     }
188 }
189
190 //enter this block if 'W' is the token
191 if (token == 'W')
192 {
193     //the possible translations for 'W'
194     switch (match)
195     {
196     case 'a':
197         change = "0";
198         break;
199     case '+':
200         change = "0";
201         break;
202     case '-':
203         change = "0";
204         break;

```



```
204         break;
205     case '*':
206         change = "0";
207         break;
208     case '/':
209         change = "0";
210         break;
211     case '(':
212         change = "0";
213         break;
214     case ')':
215         change = "0";
216         break;
217     case '$':
218         change = "0";
219         break;
220     case '=':
221         change = "E=";
222         break;
223     default:
224         change = "0";
225         break;
226     }
227 }
228
229 //enter this block if 'E' is the token
230 if (token == 'E')
231 {
232     //the possible translations for 'E'
233     switch (match)
234     {
235     case 'a':
236         change = "QT";
237         break;
238     case '+':
239         change = "0";
240         break;
241     case '-':
242         change = "0";
243         break;
```

```

244         case '*':
245             change = "0";
246             break;
247         case '/':
248             change = "0";
249             break;
250         case '(':
251             change = "QT";
252             break;
253         case ')':
254             change = "0";
255             break;
256         case '$':
257             change = "0";
258             break;
259         case '=':
260             change = "0";
261             break;
262         default:
263             change = "0";
264             break;
265     }
266 }
267
268 //enter this block if 'Q' is the token
269 if (token == 'Q')
270 {
271     //the possible translations for 'Q'
272     switch (match)
273     {
274         case 'a':
275             change = "0";
276             break;
277         case '+':
278             change = "QT+";
279             break;
280         case '-':
281             change = "QT-";
282             break;

```

```

283     case '*':
284         change = "0";
285         break;
286     case '/':
287         change = "0";
288         break;
289     case '(':
290         change = "0";
291         break;
292     case ')':
293         change = "1";
294         break;
295     case '$':
296         change = "1";
297         break;
298     case '=':
299         change = "0";
300         break;
301     default:
302         change = "0";
303         break;
304     }
305 }
306
307 //enter this block if 'T' is the token
308 if (token == 'T')
309 {
310     //the possible translations for 'T'
311     switch (match)
312     {
313     case 'a':
314         change = "RF";
315         break;
316     case '+':
317         change = "0";
318         break;
319     case '-':
320         change = "0";
321         break;

```

```
322     case '*':
323         change = "0";
324         break;
325     case '/':
326         change = "0";
327         break;
328     case '(':
329         change = "RF";
330         break;
331     case ')':
332         change = "0";
333         break;
334     case '$':
335         change = "0";
336         break;
337     case '=':
338         change = "0";
339         break;
340     default:
341         change = "0";
342         break;
343     }
344 }
345
346 //enter this block if 'R' is the token
347 if (token == 'R')
348 {
349     //the possible translations for 'R'
350     switch (match)
351     {
352     case 'a':
353         change = "0";
354         break;
355     case '+':
356         change = "1";
357         break;
358     case '-':
359         change = "1";
360         break;
```

```

361     case '*':
362         change = "RF*";
363         break;
364     case '/':
365         change = "RF/";
366         break;
367     case '(':
368         change = "0";
369         break;
370     case ')':
371         change = "1";
372         break;
373     case '$':
374         change = "1";
375         break;
376     case '=':
377         change = "0";
378         break;
379     default:
380         change = "0";
381         break;
382     }
383 }
384
385 //enter this block if 'F' is the token
386 if (token == 'F')
387 {
388     //the possible translations for 'F'
389     switch (match)
390     {
391     case 'a':
392         change = "a";
393         break;
394     case '+':
395         change = "0";
396         break;
397     case '-':
398         change = "0";
399         break;
400     case '*':
401         change = "0";
402         break;

```

```
403     case '/':
404         change = "0";
405         break;
406     case '(':
407         change = ")E(";
408         break;
409     case ')':
410         change = "0";
411         break;
412     case '$':
413         change = "0";
414         break;
415     case '=':
416         change = "0";
417         break;
418     default:
419         change = "0";
420         break;
421     }
422 }
423
424 return change;
425
426 }
```