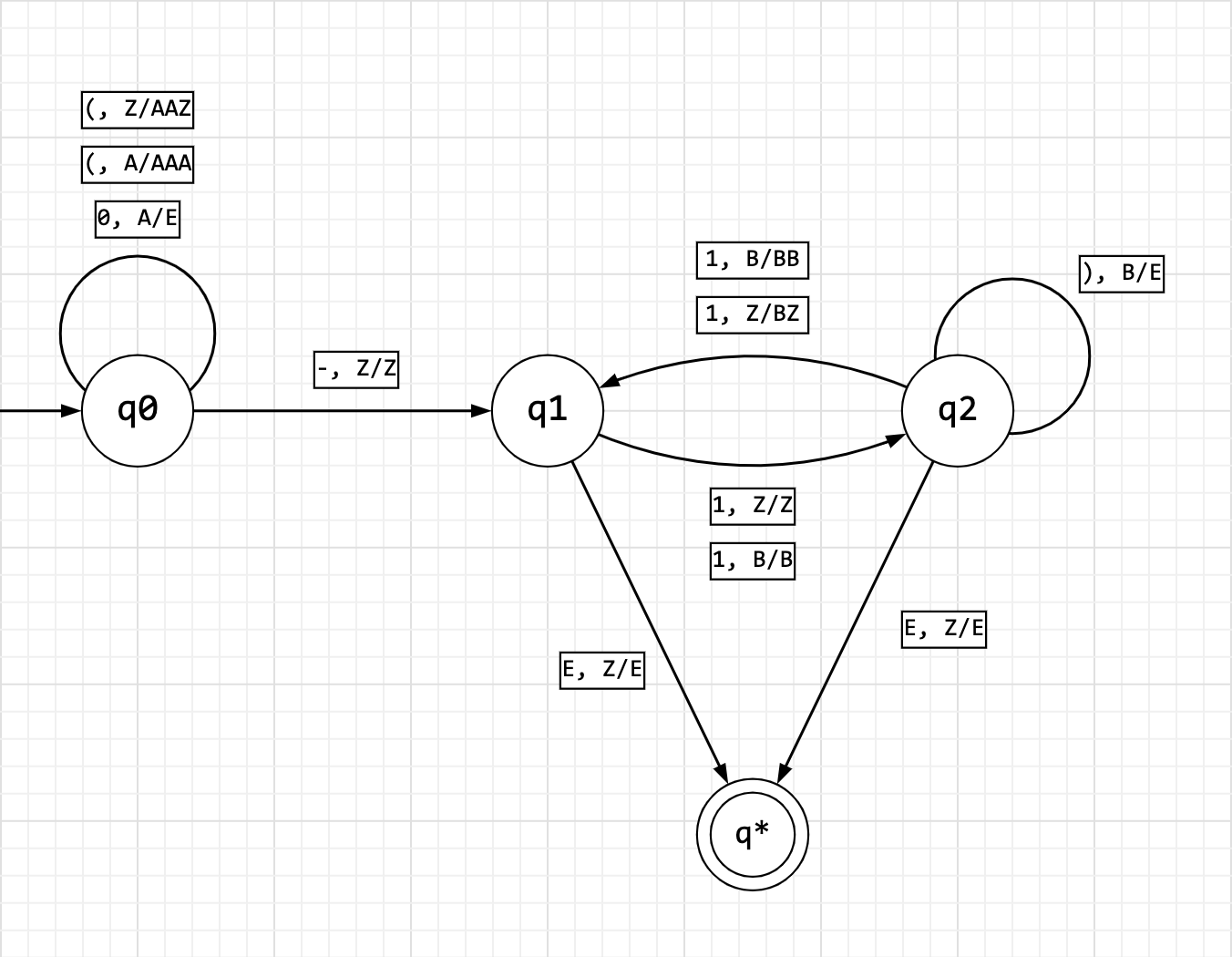
Реализация автомата с магазинной памятью  
  
  
  
  
  
  
  
  
   
 Тотиков Сармат  
 Группа: 21.Б-13

**1. Задание:**Реализовать распознающий МП-автомат для цепочек контекстно свободного языка L  
  
Описание языка L:  
L = { (n 02n - 12m )m | n,m ≥ 0 }, алфавит {-, 0, 1, (, )}  
  
**2. КС грамматика:**

S  A - B  
A  ( A 00 |   
B  11 B ) |   
 **3. Формальное задание МП автомата:**P = ({q0, q1, q2, q\*}, { -, 0, 1, (, ) }, {A, B, Z}, , q0, Z, {q\*})Далее будут представлены функции переходов  для правильной цепочки. Любая другая комбинация аргументов  функции будет считаться неверной и вести в ошибочное состояние q\*\*

1. ( q0, (, Z )  ( q0, AAZ ) - считываю все символы ‘**(‘**, при этом в стеке делаю количество букв в два раза больше, чем считанных символов ‘**(‘**
2. ( q0, (, A )  ( q0, AAA ) - аналогично 1.
3. ( q0, 0, A )  ( q0,  ) - считываю ‘**0**’, при этом удаляю из стека по одному А. Как итог, получаем, что к концу считывания символов ‘**0**’ из правильной цепочки в стеке ничего не будет находиться.
4. ( q0, -, Z )  ( q1, Z ) - считываю символ ‘ **-** ’. В стек ничего не заношу. Этот шаг необходим для дальнейшей работы программы, поэтому осуществим переход в новое состояние **q1**
5. ( q1, , Z )  ( q\*,  ) - спонтанный переход в   
   финальное состояние перед прочтением символа ‘**1**’
6. ( q1, 1, Z )  ( q2, Z ) - считываю символ **1**. Находясь в состоянии **q1** ничего не изменяю, чтобы количество символов на стеке было в 2 раза меньше количества считанных ‘**1**’
7. ( q1, 1, B )  ( q2, B ) - аналогично 6. Пункты 6 и 7 ничего не меняют на стеке и считывают ‘1’ из строки. Созданы для того, чтобы количество единиц было в 2 больше количества символов на стеке . Это не может быть аналогично 5, там переход в финальное состояние. И более, в состоянии q1 на вершине стека символ В не может появиться, так как в q1 попадаем по шагу 4, и в стеке будет только Z. Дальше не смотрел.
8. ( q2, 1, Z )  ( q1, BZ ) - в этом состоянии считываю ‘**1**’ и добавляю в стек символ ‘**B**’
9. ( q2, 1, B )  ( q1, BB ) - аналогично 8.  
     
   В правильной цепочке после 8 и 9 мы должны оказаться в состоянии **q2**
10. ( q2, ), B )  ( q2,  ) - в этом состоянии считываю символ ‘**)**’ и убираю из стека один символ B.
11. d( q2, , Z)  ( q\*,  ) - спонтанный переход в финальное состояние после прочтения всех символов и отчистки стека.

Диаграмма:  
  
Не отображенные переходы ведут в состояние ошибки **q\*\*  
  
4. Листинги программ:**

1. #include <iostream>
2. #include <string>
3. bool MP(std::string in)
4. {
5. char c;                                     *// Бегунок по строчке*
7. int counter\_one = 0;                        *// Счетчик для n и m символов*
8. int counter\_two = 0;                        *// Счетчик для 2n и 2m символов*
10. bool open\_zero = false;                     *// Проверка на то, что уже были '(' и '0'*
11. bool close\_one = false;                     *// Проверка на то, что уже были ')' и '1'*
12. bool mid\_minus = false;                     *// Проверка на то, что уже был  '-'*
14. int first\_after\_pos = 0;                    *// Запоминаем позицию элемента после последнего '0'*
15. char first\_after\_val = in[0];               *// Запоминаем элемент после последнего '0'*
17. *// Цикл для прохода по строке*
18. for(int idx = 0; idx < in.length(); ++idx)
19. {
20. c = in[idx];                            *// Обновление текста*
21. if(c == '(')                            *// Начало считывания конструкции (^n 0^(2n)*
22. {
23. if( open\_zero == false && close\_one == false && mid\_minus == false) *// проверка на то, что такой конструкции еще не было*
24. {
25. open\_zero = true;               *// Показываем, что началась конструкция (^n 0^(2n)*
27. while(c == '(')                 *// Считываем все '('*
28. {
29. ++counter\_one;
30. c = in[++idx];
31. }
33. if(c == '0')                    *// Проверяем символ после последнего '('*
34. {
35. while (c == '0')            *// Считываем все '0'*
36. {
37. ++counter\_two;
38. c = in[++idx];
39. }
40. } else {
41. std::cout << "Ошибка: в позиции " << idx + 1 << " неожидаемый символ " << c << std::endl;
42. return false;
43. }
45. if((counter\_two - (2\*counter\_one)) != 0)    *//проверка на то, что ( в 2 раза меньше 0*
46. {
47. if((counter\_two - (2\*counter\_one)) >= 0)
48. std::cout << "Ошибка: в позиции " << idx << " неожидаемый символ " << in[idx-1] << std::endl;
50. if((counter\_two - (2\*counter\_one)) < 0)
51. std::cout << "Ошибка: в позиции " << idx+1 << " неожидаемый символ " << in[idx] << std::endl;
52. return false;
53. }
55. first\_after\_pos = idx + 1;      *// Запоминаем позицию последнего символа и сам символ*
56. first\_after\_val = c;            *// На случай множетва нежданных символов после 0*
57. counter\_one = 0;
58. counter\_two = 0;
60. } else {
61. std::cout << "Ошибка: в позиции " << idx + 1 << " неожидаемый символ " << c << std::endl;
62. return false;
63. }
64. }
66. if(c == '-' && mid\_minus == false)      *// Проверяем на наличие '-'*
67. {
68. mid\_minus = true;                   *// Показываем, что '-' считался*
70. *// Обработка ряда потенциальных ошибок последовательной расстановки конструкции (^n 0^(2n), - и конструкции 1^(2n) )^n*
71. if(in[idx + 1] == '0' || in[idx + 1] == '(' || in[idx + 1] == '-')
72. {
73. std::cout << "Ошибка: в позиции " << idx + 2 << " неожидаемый символ " << in[idx + 1] << std::endl;
74. return false;
75. }
77. if((in[idx - 1] == ')' || in[idx - 1] == '1' ) && close\_one == false)
78. {
79. std::cout << "Ошибка: в позиции " << first\_after\_pos << " неожидаемый символ " << first\_after\_val << std::endl;
80. return false;
81. }
83. if((in[idx - 1] == ')' || in[idx - 1] == '1' ) && close\_one == true)
84. {
85. std::cout << "Ошибка: в позиции " << idx + 1 << " неожидаемый символ " << c << std::endl;
86. return false;
87. }
88. c = in[++idx];
89. }
91. if(c == '1' && mid\_minus == true)       *// Проверка на наличие минуса перед конструкцией 1^(2n) )^n*
92. {
93. close\_one = true;                   *// Показываем, что такая конструкция начала считываться*
95. while (c == '1')                    *// Считываем все единицы*
96. {
97. ++counter\_two;
98. c = in[++idx];
99. }
101. if (c == ')')                       *// Проверка на наличие ')' после '1'*
102. {
103. while(c == ')')                 *// Считывание ')'*
104. {
105. ++counter\_one;
106. c = in[++idx];
107. }
108. } else {
109. std::cout << "Ошибка: в позиции " << idx + 1 << " неожидаемый символ " << c << std::endl;
110. return false;
111. }
113. if ((counter\_two - 2\*counter\_one) != 0) *// Проверка на то, что '1' в два раза больше чем ')'*
114. {
115. if((counter\_two - (2\*counter\_one)) >= 0)
116. std::cout << "Ошибка: в позиции " << idx+1 << " неожидаемый символ \\0" << std::endl;
118. if((counter\_two - (2\*counter\_one)) < 0)
119. std::cout << "Ошибка: в позиции " << idx << " неожидаемый символ " << in[idx-1] << std::endl;
120. return false;
121. }
122. --idx;
123. } else if (mid\_minus == false)
124. {
125. std::cout << "Ошибка: в позиции " << idx + 1 << " неожидаемый символ " << in[idx] << std::endl;
126. return false;
127. }
129. *// Проверка на сторонние символы*
130. if(c != ')' && c != '1' && c != '-' && c != '0' && c != '(' && c != '\0')
131. {
132. std::cout << "Ошибка: в позиции " << idx + 2 << " неожидаемый символ " << in[++idx] << std::endl;
133. return 0;
134. }
136. *// Дополнительная проверка на то, что символов ')' слишком много*
137. if(c == ')')
138. {
139. std::cout << "Ошибка: в позиции " << idx + 1 << " неожидаемый символ " << in[idx] << std::endl;
140. return 0;
141. }
142. }
143. Std::cout << “Цепочка принадлежит языку” << std::endl;
144. return true;
145. }
146. int main(int argc, const char \* argv[]) {
148. *//    std::string input = "((((((((((000000000000000000000-111111)))";*
149. *//    bool avtomat = MP(input);*
151. *//---------------------------------------------------------*
152. *// Тест на верную строку:*
153. std::string test\_01 = "(00-11)";
154. std::cout << "=== Test 01: ===" << std::endl;
155. MP(test\_01);
156. std::cout << std::endl;
157. *// Вывод:*
158. *//      === Test 01: ===*
159. *//---------------------------------------------------------*
160. *// Тест на строку из -:*
161. std::string test\_02 = "-";
162. std::cout << "=== Test 02: ===" << std::endl;
163. MP(test\_02);
164. std::cout << std::endl;
166. *// Вывод:*
167. *//      === Test 02: ===*
168. *//*
170. *//---------------------------------------------------------*
171. *// Тест строку c n=0:*
172. std::string test\_03 = "-11)";
173. std::cout << "=== Test 03: ===" << std::endl;
174. MP(test\_03);
175. std::cout << std::endl;
177. *// Вывод:*
178. *//      === Test 03: ===*
179. *//*
181. *//---------------------------------------------------------*
182. *// Тест на длинную строку n = 26, m = 3:*
183. std::string test\_04 = "((((((((((((((((((((((((((0000000000000000000000000000000000000000000000000000-111111)))";
184. std::cout << "=== Test 04: ===" << std::endl;
185. MP(test\_04);
186. std::cout << std::endl;
188. *// Вывод:*
189. *//      === Test 04: ===*
190. *//*
192. *//---------------------------------------------------------*
193. *// Тест с ошибкой в количестве '(' и '0':*
194. std::string test\_05 = "((00000-11)";
195. std::cout << "=== Test 05: ===" << std::endl;
196. MP(test\_05);
197. std::cout << std::endl;
198. *// Вывод:*
199. *//      === Test 05: ===*
200. *// Ошибка: в позиции 7 неожидаемый символ 0*
202. *//---------------------------------------------------------*
203. *// Тест с ошибкой в количестве '1' и ')':*
204. std::string test\_06 = "((0000-1111)";
205. std::cout << "=== Test 06: ===" << std::endl;
206. MP(test\_06);
207. std::cout << std::endl;
208. *// Вывод:*
209. *//      === Test 06: ===*
210. *// Ошибка: в позиции 13 неожидаемый символ \0*
212. *//---------------------------------------------------------*
213. *// Тест c ошибкой последовательности №1:*
214. std::string test\_07 = "-11)(00";
215. std::cout << "=== Test 07: ===" << std::endl;
216. MP(test\_07);
217. std::cout << std::endl;
218. *// Вывод:*
219. *//      === Test 07: ===*
220. *// Ошибка: в позиции 5 неожидаемый символ (*
222. *//---------------------------------------------------------*
223. *// Тест c ошибкой последовательности №2:*
224. std::string test\_08 = "11)-";
225. std::cout << "=== Test 08: ===" << std::endl;
226. MP(test\_08);
227. std::cout << std::endl;
229. *// Вывод:*
230. *//      === Test 08: ===*
231. *// Ошибка: в позиции 1 неожидаемый символ 1*
232. *//---------------------------------------------------------*
233. *// Тест с ошибкой на наверное расположение '-':*
234. std::string test\_09 = "(0-011)";
235. std::cout << "=== Test 09: ===" << std::endl;
236. MP(test\_09);
237. std::cout << std::endl;
239. *// Вывод:*
240. *//      === Test 09: ===*
241. *// Ошибка: в позиции 3 неожидаемый символ -*
242. *//---------------------------------------------------------*
243. *// Тест на ошибочную последоватнльность '(' и '0'*
244. std::string test\_10 = "00(-11)";
245. std::cout << "=== Test 10: ===" << std::endl;
246. MP(test\_10);
247. std::cout << std::endl;
249. *// Вывод:*
250. *//      === Test 10: ===*
251. *// Ошибка: в позиции 1 неожидаемый символ 0*
252. *//---------------------------------------------------------*
253. *// Тест на ошибочную последоватнльность ')' и '1'*
254. std::string test\_11 = "(00-)11";
255. std::cout << "=== Test 11: ===" << std::endl;
256. MP(test\_11);
257. std::cout << std::endl;
258. *// Вывод:*
259. *//      === Test 11: ===*
260. *// Ошибка: в позиции 5 неожидаемый символ )*
261. *//---------------------------------------------------------*
262. *// Тест с ошибкой на сторонние символы в начале:*
263. std::string test\_12 = "+(00-11)";
264. std::cout << "=== Test 12: ===" << std::endl;
265. MP(test\_12);
266. std::cout << std::endl;
268. *// Вывод:*
269. *//      === Test 12: ===*
270. *// Ошибка: в позиции 1 неожидаемый символ +*
271. *//---------------------------------------------------------*
272. *// Тест с ошибкой на сторонние символы в середине:*
273. std::string test\_13 = "(00+-11)";
274. std::cout << "=== Test 13: ===" << std::endl;
275. MP(test\_13);
276. std::cout << std::endl;
278. *// Вывод:*
279. *//      === Test 13: ===*
280. *// Ошибка: в позиции 4 неожидаемый символ +*
281. *//---------------------------------------------------------*
282. *// Тест с ошибкой на сторонние символы в конце:*
283. std::string test\_14 = "(00-11)+";
284. std::cout << "=== Test 14: ===" << std::endl;
285. MP(test\_14);
286. std::cout << std::endl;
288. *// Вывод:*
289. *//      === Test 14: ===*
290. *// Ошибка: в позиции 8 неожидаемый символ +*
291. *//---------------------------------------------------------*
292. *// Тест с ошибкой на двойной минус:*
293. std::string test\_15 = "(00--11)";
294. std::cout << "=== Test 15: ===" << std::endl;
295. MP(test\_15);
296. std::cout << std::endl;
298. *// Вывод:*
299. *//      === Test 15 ===*
300. *// Ошибка: в позиции 5 неожидаемый символ -*
301. return 0;
302. }

**5. Краткое описание работы подпрограммы, реализующей МП автомат, и программы распознавания цепочек языка.**Разбиваем программу на три блока.  
  
Первый блок отвечает за считывание (n 02n  (1)  
Второй блок отвечает за считывание ‘-‘ (2)

Третий блок отвечает за считывание 12n )n (3)  
Между каждым блоком настроены связи:  
Если 1 активирован, то далее 1 быть не может, но может быть активирован 2  
Если 2 активирован, то далее 1 и 2 быть не может, но может быть активирован 3  
Если 3 активирован, то уже был активирован 2, и далее ни 1, ни 2, ни 3 не могут быть активированными.  
  
  
  
1. На вход поступает строка

2. Начинаем проходиться по ней в цикле

3. Ищем ‘(’, если такой есть, то активируем первый блок

3.1. Если мы такой нашли, то считываем все символы ‘(’ и считаем их.

3.2. Далее проверяем, что следующий символ – ‘0’

3.3. Если всё верно, то считываем и считаем все ‘0’

3.4. Если всё сходится и ‘0’ в два раза больше, чем ‘(’ и они стоят в нужной последовательности, то показываем, что блок завершен  
  
4. Ищем ‘-’, активируем второй блок

4.1. Считываем минус  
 4.2. Если третьего блока программы нет, то завершаем программу.

5. Ищем ‘1’, если уже был активирован второй блок, то активируем третий

5.1. Считываем все символы ‘1’, считаем их.

5.2. Далее проверяем, что следующий символ – ‘)’

5.3. Если всё сходится и ‘1’ в два раза больше, чем ‘)’, то показываем, что блок завершен и завершаем программу.

Таким образом, программа работает в случаях, когда:  
1. Есть первый блок, второй блок, третий блок

2. Есть первый блок, второй блок

3. Есть второй блок, третий блок

4. Есть только второй блок  
  
Также, заметим, что если при любой проверке условие не выполняется, то мы выводим ошибку с номером позиции и символом. **6. Таблица тестирования программы:**

|  |  |
| --- | --- |
| Тест | Вывод |
| (00-11) | Цепочка принадлежит языку |
| - | Цепочка принадлежит языку |
| -11) | Цепочка принадлежит языку |
| (((((((((((((((((((((((((( 00000000000000000000000000 00000000000000000000000000 -111111))) | Цепочка принадлежит языку |
| ((00000-11) | Ошибка: в позиции 7 неожидаемый символ 0 |
| ((0000-1111) | Ошибка: в позиции 13 неожидаемый символ \0 |
| -11)(00 | Ошибка: в позиции 5 неожидаемый символ ( |
| 11)- | Ошибка: в позиции 1 неожидаемый символ 1 |
| (0-011) | Ошибка: в позиции 3 неожидаемый символ - |
| 00(-11) | Ошибка: в позиции 1 неожидаемый символ 0 |
| (00-)11 | Ошибка: в позиции 5 неожидаемый символ ) |
| +(00-11) | Ошибка: в позиции 1 неожидаемый символ + |
| (00+-11) | Ошибка: в позиции 4 неожидаемый символ + |
| (00-11)+ | Ошибка: в позиции 8 неожидаемый символ + |
| (00--11) | Ошибка: в позиции 5 неожидаемый символ - |