



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Московский государственный технический университет имени
Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет по лабораторной работе №2

Название Преобразование грамматик

Дисциплина Конструирование компиляторов

Студент Шацкий Н. С.

Группа ИУ7-21М

Преподаватель Ступников А. А.

Москва — 2024 г.

1 Выполнение лабораторной работы

1.1 Общий вариант для всех: Устранение левой рекурсии.

Определение. Нетерминал A КС-грамматики $G = (N, \Sigma, P, S)$ называется рекурсивным, если $A \Rightarrow^+ \alpha A \beta$ для некоторых α и β . Если $\alpha = \epsilon$, то A называется леворекурсивным. Аналогично, если $\beta = \epsilon$, то A называется праворекурсивным. Грамматика, имеющая хотя бы один леворекурсивный нетерминал, называется леворекурсивной. Аналогично определяется праворекурсивная грамматика. Грамматика, в которой все нетерминалы, кроме, быть может, начального символа, рекурсивные, называется рекурсивной.

Некоторые из алгоритмов разбора не могут работать с леворекурсивными грамматиками. Можно показать, что каждый КС-язык определяется хотя бы одной не леворекурсивной грамматикой.

Постройте программу, которая в качестве входа принимает приведенную КС-грамматику $G = (N, \Sigma, P, S)$ и преобразует ее в эквивалентную КС-грамматику G' без левой рекурсии.

1.2 Преобразование к нормальной форме Хомского.

Определение. КС-грамматика $G = (N, \Sigma, P, S)$ называется грамматикой в нормальной форме Хомского (или в бинарной нормальной форме), если каждое правило из P имеет один из следующих видов:

1. $A \rightarrow BC$, где A, B и C принадлежат N ,
2. $A \rightarrow a$, где $a \in \Sigma$,
3. $S \rightarrow \epsilon$, если $\epsilon \in L(G)$, причем S не встречается в правых частях правил.

Можно показать, что каждый КС-язык порождается грамматикой в нормальной форме Хомского. Этот результат полезен в случаях, когда требуется простая форма представления КС-языка.

Постройте программу, которая в качестве входа принимает приведенную КС-грамматику $G = (N, \Sigma, P, S)$ и преобразует ее в эквивалентную КС-грамматику G' в нормальной форме Хомского.

1.3 Результаты работы программы

Результаты работы программы по преобразованию грамматик приведены на рисунках 1.1–1.14.

1.3.1 Устранение левой рекурсии

$G = (\{E, T, F\}, \{+, *, (,), a\}, P, E)$, где P состоит из правил:

$E \rightarrow E + T \mid T$
 $T \rightarrow T * F \mid F$
 $F \rightarrow a \mid (E)$

Рисунок 1.1 – Исходная грамматика для удаления левой рекурсии (пример 1)

$G = (\{E, E', T, T', F\}, \{+, *, (,), a\}, P, E)$, где P состоит из правил:

$E \rightarrow T E'$
 $E' \rightarrow + T E' \mid \varepsilon$
 $T \rightarrow F T'$
 $T' \rightarrow * F T' \mid \varepsilon$
 $F \rightarrow a \mid (E)$

Рисунок 1.2 – Грамматика после удаления левой рекурсии (пример 1)

$G = (\{S, A\}, \{a, b, c, d\}, P, A)$, где P состоит из правил:

$S \rightarrow A a \mid b$
 $A \rightarrow A c \mid S d \mid \varepsilon$

Рисунок 1.3 – Исходная грамматика для удаления левой рекурсии (пример 2)

$G = (\{S, A, A'\}, \{a, b, c, d\}, P, A)$, где P состоит из правил:

$S \rightarrow A a \mid b$
 $A \rightarrow b d A' \mid A'$
 $A' \rightarrow c A' \mid a d A' \mid \varepsilon$

Рисунок 1.4 – Грамматика после удаления левой рекурсии (пример 2)

```
G = ({E, T, F}, {+, -, *, /, (, ), id}, P, E), где P состоит из правил:
E → E + T | E - T | T
T → T * F | T / F | F
F → ( E ) id
```

Рисунок 1.5 – Исходная грамматика для удаления левой рекурсии (пример 3)

```
G = ({E, E', T, T', F}, {+, -, *, /, (, ), id}, P, E), где P состоит из правил:
E → T E'
E' → + T E' | - T E' | ε
T → F T'
T' → * F T' | / F T' | ε
F → ( E ) id
```

Рисунок 1.6 – Грамматика после удаления левой рекурсии (пример 3)

1.3.2 Устранение левой факторизации

```
G = ({S, E}, {i, t, e, a, b}, P, S), где P состоит из правил:
S → i E t S | i E t S e S | a
E → b
```

Рисунок 1.7 – Исходная грамматика для удаления левой факторизации

```
G = ({S, S', E}, {i, t, e, a, b}, P, S), где P состоит из правил:
S → i E t S S' | a
S' → ε | e S
E → b
```

Рисунок 1.8 – Грамматика после удаления левой факторизации

1.3.3 Преобразование КС-грамматики к нормальной форме Хомского

$G = (\{S, A, B\}, \{a, b\}, P, S)$, где P состоит из правил:

```
S → a A B | B A
A → B B B | a
B → A S | b
```

Рисунок 1.9 – Исходная грамматика перед преобразованием к нормальной форме Хомского (пример 1)

$G = (\{S, A, B, a', \langle AB \rangle, \langle BB \rangle\}, \{a, b\}, P, S)$, где P состоит из правил:

```
S → a' <AB> | B A
A → B <BB> | a
B → A S | b
a' → a
<AB> → A B
<BB> → B B
```

Рисунок 1.10 – Грамматика после преобразования к нормальной форме Хомского (пример 1)

$G = (\{S\}, \{0, 1\}, P, S)$, где P состоит из правил:

```
S → 0 S 1 | 0 1
```

Рисунок 1.11 – Исходная грамматика перед преобразованием к нормальной форме Хомского (пример 2)

$G = (\{S, 0', \langle S1 \rangle, 1'\}, \{0, 1\}, P, S)$, где P состоит из правил:

```
S → 0' <S1> | 0' 1'
0' → 0
<S1> → S 1'
1' → 1
```

Рисунок 1.12 – Грамматика после преобразования к нормальной форме Хомского (пример 2)

$G = (\{S, A, B\}, \{a, b\}, P, S)$, где P состоит из правил:

$S \rightarrow a B \mid b A$
 $A \rightarrow a S \mid b A A \mid a$
 $B \rightarrow b S \mid a B B \mid b$

Рисунок 1.13 – Исходная грамматика перед преобразованием к нормальной форме Хомского (пример 3)

$G = (\{S, A, B, a', b', \langle AA \rangle, \langle BB \rangle\}, \{a, b\}, P, S)$, где P состоит из правил:

$S \rightarrow a' B \mid b' A$
 $A \rightarrow a' S \mid b' \langle AA \rangle \mid a$
 $B \rightarrow b' S \mid a' \langle BB \rangle \mid b$
 $a' \rightarrow a$
 $b' \rightarrow b$
 $\langle AA \rangle \rightarrow A A$
 $\langle BB \rangle \rightarrow B B$

Рисунок 1.14 – Грамматика после преобразования к нормальной форме Хомского (пример 3)

2 Контрольные вопросы

1. Как может быть определён формальный язык?
 - (a) Простым перечислением слов, входящих в данный язык.
 - (b) Словами, порождёнными некоторой формальной грамматикой.
 - (c) Словами, порождёнными регулярным выражением.
 - (d) Словами, распознаваемыми некоторым конечным автоматом.
2. Какими характеристиками определяется грамматика?
 - (a) Σ — множество терминальных символов.
 - (b) N — множество нетерминальных символов.
 - (c) P — множество правил (слева — непустая последовательность терминалов/нетерминалов, содержащая хотя бы один нетерминал, справа — любая последовательность терминалов/нетерминалов).
 - (d) S — начальный символ из множества нетерминалов.
3. Дайте описания грамматик по иерархии Хомского.
 - (a) Регулярные.
 - (b) Контекстно-свободные.
 - (c) Контекстно-зависимые.
 - (d) Неограниченные.
4. Какие абстрактные устройства используются для разбора грамматик?
 - (a) Распознающие грамматики — устройства (алгоритмы), которым на вход подается цепочка языка, а на выходе устройство печатает «Да», если цепочка принадлежит языку, и «Нет» — иначе.
5. Оцените временную и емкостную сложность предложенного вам алгоритма.
 - (a) Алгоритм удаления левой рекурсии
 - $O(N^2)$ — временная сложность;
 - $O(N)$ — ёмкостная сложность.

3 Текст программы

В листингах 3.1–3.3 представлен код программы.

Листинг 3.1 — Основной модуль программы

```
1  import subprocess
2
3  from color import *
4  from grammar import Grammar, readGrammarFromFile
5
6
7  MENU = f"""
8      {YELLOW}\tМеню\n
9      {YELLOW}1.{BASE}  Исходная грамматика;
10     {YELLOW}2.{BASE}  Грамматика после устранения левой рекурсии;
11     {YELLOW}3.{BASE}  Грамматика после устранения левой факторизации;
12     {YELLOW}4.{BASE}  Грамматика после устранения левой рекурсии и левой
    ↪ факторизации;
13     {YELLOW}5.{BASE}  Преобразование КС-грамматики к нормальной форме Хомского.
14
15     {YELLOW}0.{BASE}  Выход.\n
16     {GREEN}Выбор:{BASE}  """
17
18
19  SIZE_MENU = 5
20  OUTPUT_FILE_NAME = "../data/result.txt"
21
22
23  def inputOption(minOptions: int, maxOptions: int, msg: str):
24      try:
25          option = int(input(msg))
26      except:
27          option = -1
28      else:
29          if option < minOptions or option > maxOptions:
30              option = -1
31
32      if option == -1:
33          print(f"{RED}\nОжидался ввод целого числа от {minOptions} до
    ↪ {maxOptions}{BASE}")
34
35      return option
36
37
38  def chooseInputFile() -> str:
39      with open("temp.txt", "w") as f:
```


Продолжение листинга 3.1

```
40     subprocess.run(["ls", "../data"], stdout=f)
41
42     with open("temp.txt") as f:
43         fileNames = [line[:-1] for line in f.readlines()]
44
45     subprocess.run(["rm", "temp.txt"])
46
47     msg = f"\n\t{YELLOW}Входные файлы:{BASE}\n\n"
48     for i in range(len(fileNames)):
49         msg += f"    {YELLOW}{i + 1}.{BASE} {fileNames[i]};\n"
50     msg += f"\n    {GREEN}Выбор:{BASE} "
51
52     option = -1
53     while option == -1:
54         option = inputOption(
55             minOptions=1,
56             maxOptions=len(fileNames),
57             msg=msg,
58         )
59
60     return f"../data/{fileNames[option - 1]}"
61
62
63 def main():
64     inputFile = chooseInputFile()
65     option = -1
66     while option != 0:
67         option = inputOption(
68             minOptions=0,
69             maxOptions=SIZE_MENU,
70             msg=MENU,
71         )
72         match option:
73             case 1:
74                 grammar: Grammar = reedGrammarFromFile(inputFile)
75                 grammar.printGrammar()
76             case 2:
77                 grammar: Grammar = reedGrammarFromFile(inputFile)
78                 grammar.removeLeftRecursion()
79                 grammar.printGrammar()
80                 grammar.createFileFromGrammar(OUTPUT_FILE_NAME)
81             case 3:
82                 grammar: Grammar = reedGrammarFromFile(inputFile)
83                 grammar.removeLeftFactorization()
84                 grammar.printGrammar()
```

Продолжение листинга 3.1

```
85         grammar.createFileFromGrammar(OUTPUT_FILE_NAME)
86     case 4:
87         grammar: Grammar = reedGrammarFromFile(inputFile)
88         grammar.removeLeftRecursion()
89         grammar.removeLeftFactorization()
90         grammar.printGrammar()
91         grammar.createFileFromGrammar(OUTPUT_FILE_NAME)
92     case 5:
93         grammar: Grammar = reedGrammarFromFile(inputFile)
94         grammar.convertToChomskyForm()
95         grammar.printGrammar()
96         grammar.createFileFromGrammar(OUTPUT_FILE_NAME)
97
98
99     if __name__ == '__main__':
100         main()
```

Листинг 3.2 — Модуль для преобразования грамматик

```
1     from functools import reduce
2     from copy import deepcopy
3
4
5     class Grammar:
6         notTerminals: list[str]
7         terminals: list[str]
8         rules: dict[str, list[list[str]]]
9         start: str
10
11     def __init__(
12         self,
13         notTerminals: list[str],
14         terminals: list[str],
15         rules: dict[str, list[list[str]]],
16         start: str
17     ) -> None:
18         self.notTerminals = notTerminals
19         self.terminals = terminals
20         self.rules = rules
21         self.start = start
22
23     def printGrammar(self) -> None:
24         notTerminals = Grammar.__joinListWithSymbol(self.notTerminals, ", ")
25         terminals = Grammar.__joinListWithSymbol(self.terminals, ", ")
26
```

Продолжение листинга 3.2

```

27     print(f"\nG = ({{{notTerminals}}}, {{{terminals}}}, P, {self.start}), где
    ↪ P состоит из правил:\n")
28     for notTerminal in self.notTerminals:
29         rightRules = self.rules[notTerminal]
30         self.__printProduct(notTerminal, rightRules)
31
32     def removeLeftRecursion(self) -> None:
33         i = 0
34         while i < len(self.notTerminals):
35             copyRightRules = self.rules[self.notTerminals[i]].copy()
36             for j in range(i):
37                 self.__replaceProducts(
38                     notTerminal=self.notTerminals[i],
39                     replaceableNotTerminal=self.notTerminals[j],
40                 )
41             if self.__removeDirectLeftRecursion(self.notTerminals[i]):
42                 i += 2
43             else:
44                 self.rules[self.notTerminals[i]] = copyRightRules
45                 i += 1
46
47     def removeLeftFactorization(self) -> None:
48         i = 0
49         while i < len(self.notTerminals):
50             maxPrefix = ""
51             rightRules = self.rules[self.notTerminals[i]]
52             for j in range(len(rightRules)):
53                 prefix = ""
54                 for symbol in rightRules[j]:
55                     indexList = self.__findPrefixMatches(
56                         rightRules=rightRules,
57                         prefix=prefix + symbol,
58                     )
59                     if len(indexList) > 1:
60                         prefix += symbol
61                     else:
62                         break
63
64                 if len(prefix) > len(maxPrefix):
65                     maxPrefix = prefix
66
67             if maxPrefix:
68                 print(f"\nСамый длинный префикс для {self.notTerminals[i]}:
    ↪ {maxPrefix}")

```

Продолжение листинга 3.2

```

69         self.__removeDirectLeftFactorization(self.notTerminals[i],
        ↪ maxPrefix)
70     else:
71         i += 1
72
73     def convertToChomskyForm(self) -> None:
74         for notTerminal in self.notTerminals.copy():
75             for i in range(len(self.rules[notTerminal])):
76                 rightRule = self.rules[notTerminal][i]
77                 if len(rightRule) == 2 and \
78                     rightRule[0] in self.notTerminals and \
79                     rightRule[1] in self.notTerminals or \
80                     len(rightRule) == 1 and rightRule[0] in self.terminals or \
81                     notTerminal == self.start and rightRule[0] == "Epselen":
82                     continue
83
84                 elif len(rightRule) == 2 and \
85                     (rightRule[0] in self.terminals or rightRule[1] in
86                     ↪ self.terminals):
87                     if rightRule[0] in self.terminals:
88                         firstElem = f"{rightRule[0]}"
89                         if not firstElem in self.notTerminals:
90                             self.notTerminals.append(firstElem)
91                             self.rules[firstElem] = [[rightRule[0]]]
92                     else:
93                         firstElem = rightRule[0]
94
95                     if rightRule[1] in self.terminals:
96                         secondElem = f"{rightRule[1]}"
97                         if not secondElem in self.notTerminals:
98                             self.notTerminals.append(secondElem)
99                             self.rules[secondElem] = [[rightRule[1]]]
100                     else:
101                         secondElem = rightRule[1]
102
103                     self.rules[notTerminal][i] = [firstElem, secondElem]
104
105                 elif len(rightRule) > 2:
106                     if rightRule[0] in self.notTerminals:
107                         self.rules[notTerminal][i] = [rightRule[0],
108                         ↪ f"<{"".join(rightRule[1:])}>"]
109                     else:
110                         self.rules[notTerminal][i] = [f"{rightRule[0]}",
111                         ↪ f"<{"".join(rightRule[1:])}>"]
112                     if not f"{rightRule[0]}" in self.notTerminals:

```

Продолжение листинга 3.2

```

110         self.notTerminals.append(f"{rightRule[0]}")
111         self.rules[f"{rightRule[0]}"] = [[rightRule[0]]]
112
113     rightRule = rightRule[1:]
114     newNotTerminal = f"<{"".join(rightRule)}>"
115     while len(rightRule) > 2:
116         if not newNotTerminal in self.notTerminals:
117             self.notTerminals.append(newNotTerminal)
118
119         if rightRule[0] in self.notTerminals:
120             self.rules[newNotTerminal] = [[rightRule[0],
121 ↪ f"<{"".join(rightRule[1:])>"]]]
122         else:
123             self.rules[newNotTerminal] = [[f"{rightRule[0]}",
124 ↪ f"<{"".join(rightRule[1:])>"]]]
125             if not f"{rightRule[0]}" in self.notTerminals:
126                 self.notTerminals.append(f"{rightRule[0]}")
127                 self.rules[f"{rightRule[0]}"] =
128 ↪ [[rightRule[0]]]
129
130     rightRule = rightRule[1:]
131     newNotTerminal = f"<{"".join(rightRule)}>"
132
133     newNotTerminal = f"<{"".join(rightRule)}>"
134     if not newNotTerminal in self.notTerminals:
135         if rightRule[0] in self.terminals:
136             firstElem = f"{rightRule[0]}"
137             if not firstElem in self.notTerminals:
138                 self.notTerminals.append(firstElem)
139                 self.rules[firstElem] = [[rightRule[0]]]
140         else:
141             firstElem = rightRule[0]
142
143         if rightRule[1] in self.terminals:
144             secondElem = f"{rightRule[1]}"
145             if not secondElem in self.notTerminals:
146                 self.notTerminals.append(secondElem)
147                 self.rules[secondElem] = [[rightRule[1]]]
148         else:
149             secondElem = rightRule[1]
150
151     self.notTerminals.append(newNotTerminal)
152     self.rules[newNotTerminal] = [[firstElem, secondElem]]
153
154 def createFileFromGrammar(self, fileName: str) -> None:

```

Продолжение листинга 3.2

```

152         with open(fileName, "w") as f:
153             for i in range(len(self.notTerminals)):
154                 if i:
155                     f.write(" ")
156                     f.write(f"{self.notTerminals[i]}")
157                 f.write("\n")
158
159             for i in range(len(self.terminals)):
160                 if i:
161                     f.write(" ")
162                     f.write(f"{self.terminals[i]}")
163                 f.write("\n")
164
165             for notTerminal in self.notTerminals:
166                 for rightRule in self.rules[notTerminal]:
167                     f.write(f"{notTerminal} ->")
168                     for symbol in rightRule:
169                         f.write(f" {symbol}")
170                     f.write("\n")
171
172             f.write(f"{self.start}\n")
173
174     def __removeDirectLeftFactorization(self, notTerminal: str, maxPrefix: str) ->
175     ↪ None:
176         indexList = self.__findPrefixMatches(
177             rightRules=self.rules[notTerminal],
178             prefix=maxPrefix,
179         )
180         newRightRules = []
181         lenMaxPrefix = len(maxPrefix)
182         for i in indexList:
183             if len(self.rules[notTerminal][i]) > lenMaxPrefix:
184                 newRightRules.append(self.rules[notTerminal][i][lenMaxPrefix:])
185             else:
186                 newRightRules.append(["Epselen"])
187
188         rightRules = []
189         for i in range(len(self.rules[notTerminal])):
190             if not i in indexList:
191                 rightRules.append(self.rules[notTerminal][i])
192
193         newNotTerminal = self.__findNewNotTerminal(notTerminal)
194         self.rules[newNotTerminal] = newRightRules
195         self.rules[notTerminal] = \
196             [list(maxPrefix) + [newNotTerminal]] + rightRules

```

Продолжение листинга 3.2

```

196         indexNotTerminal = self.notTerminals.index(notTerminal)
197     self.notTerminals = \
198         self.notTerminals[:indexNotTerminal + 1] + [newNotTerminal] + \
199         self.notTerminals[indexNotTerminal + 1:]
200
201
202     def __findNewNotTerminal(self, notTerminal: str):
203         try:
204             baseNotTerminal = notTerminal[:notTerminal.index('"')]
205         except ValueError:
206             baseNotTerminal = notTerminal
207
208         quotationMarkCount = 0
209         for item in self.notTerminals:
210             if item.find(baseNotTerminal) != -1:
211                 quotationMarkCount += 1
212
213         return notTerminal + '"' * quotationMarkCount
214
215     def __findPrefixMatches(self, rightRules: list[list[str]], prefix: str) ->
216     ↪ list[int]:
217         indexList = []
218         for i in range(len(rightRules)):
219             if self.__comparePrefixes(rightRules[i], prefix):
220                 indexList.append(i)
221
222         return indexList
223
224     def __comparePrefixes(self, rightRule: list[str], prefix: str) -> bool:
225         if len(rightRule) < len(prefix):
226             return False
227
228         for i in range(len(prefix)):
229             if prefix[i] != rightRule[i]:
230                 return False
231
232         return True
233
234     def __replaceProducts(self, notTerminal: str, replaceableNotTerminal: str) ->
235     ↪ None:
236         flagReplace = False
237         newRightRules = []
238         rightRules = self.rules[notTerminal]
239         for i in range(len(rightRules)):
240             if replaceableNotTerminal not in rightRules[i]:

```

Продолжение листинга 3.2

```

239         newRightRules.append(rightRules[i])
240         continue
241
242     flagReplace = True
243     j = rightRules[i].index(replaceableNotTerminal)
244     for substitutedRightRule in self.rules[replaceableNotTerminal]:
245         newRightRule = rightRules[i][:j]
246         if substitutedRightRule[0] != "Epselen":
247             newRightRule.extend(substitutedRightRule)
248             newRightRule.extend(rightRules[i][j + 1:])
249             newRightRules.append(newRightRule)
250
251     if flagReplace:
252         self.rules[notTerminal] = newRightRules
253         print(f"\nПосле замены {replaceableNotTerminal}: ", end="")
254         self.__printProduct(notTerminal, newRightRules)
255
256     def __removeDirectLeftRecursion(self, notTerminal: str) -> bool:
257         self.rules[notTerminal].sort(
258             key=lambda rightRule: rightRule[0] != notTerminal
259         )
260         newNotTerminal = notTerminal + "'"
261         rightRulesForNewNotTerminal = []
262         rightRules = []
263
264         for rightRule in deepcopy(self.rules[notTerminal]):
265             if rightRule[0] != notTerminal:
266                 if rightRule[0] == "Epselen":
267                     rightRule = [newNotTerminal]
268                 else:
269                     rightRule.append(newNotTerminal)
270                     rightRules.append(rightRule)
271             else:
272                 rightRule = rightRule[1:]
273                 rightRule.append(newNotTerminal)
274                 rightRulesForNewNotTerminal.append(rightRule)
275
276         if len(rightRulesForNewNotTerminal):
277             rightRulesForNewNotTerminal.append(["Epselen"])
278             indexNotTerminal = self.notTerminals.index(notTerminal)
279             self.notTerminals = \
280                 self.notTerminals[:indexNotTerminal + 1] + [newNotTerminal] + \
281                 self.notTerminals[indexNotTerminal + 1:]
282             self.rules[newNotTerminal] = rightRulesForNewNotTerminal
283             self.rules[notTerminal] = rightRules

```


Продолжение листинга 3.2

```
284
285         removedFlag = True
286     else:
287         removedFlag = False
288
289     return removedFlag
290
291 def __printProduct(self, notTerminal: str, rightRules: list[list[str]]):
292     print(f"{notTerminal} -> ", end="")
293     for i in range(len(rightRules)):
294         print(f"{" | " if i != 0 else
295             ↪ " "}{Grammar.__joinListWithSymbol(rightRules[i], " ")}", end="")
296     print()
297
298 @staticmethod
299 def __joinListWithSymbol(arr: list[str], symbol: str) -> str:
300     return reduce(lambda elemPrev, elem: f"{elemPrev}{symbol}{elem}", arr)
301
302 def reedGrammarFromFile(fileName: str) -> Grammar:
303     with open(fileName) as f:
304         lines = [line[:-1] for line in f.readlines()]
305
306     notTerminals = lines[0].split(" ")
307     terminals = lines[1].split(" ")
308     start = lines[-1]
309     rules = {}
310     for notTerminal in notTerminals:
311         rules[notTerminal] = []
312
313     for rule in lines[2:-1]:
314         rule = rule.split(" ")
315         rules[rule[0]].append(rule[2:])
316
317     return Grammar(
318         notTerminals=notTerminals,
319         terminals=terminals,
320         rules=rules,
321         start=start,
322     )
```

Листинг 3.3 — Модуль с вариантами цветов при выводе сообщений в
консоль

```
1  BASE    = "\x1B[0m"  
2  GREEN   = "\x1B[32m"  
3  RED     = "\x1B[31m"  
4  YELLOW  = "\x1B[33m"  
5  BLUE    = "\x1B[34m"  
6  PURPLE  = "\x1B[35m"
```