МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»

**Факультет программной инженерии и компьютерной техники**

**Домашнее задание №3**

по дисциплине

“Разработка компиляторов”

**Вариант № 9**

**Студент:**

Румский А.

Группа P3307

**Преподаватель:**

Лаздин Артур Вячеславович

Санкт-Петербург, 2025 год

Оглавление

[Вариант 3](#_Toc200146460)

[Задание 3](#_Toc200146461)

[Удаление левой рекурсии 4](#_Toc200146462)

[Левая факторизация 4](#_Toc200146463)

[FIRST 5](#_Toc200146464)

[FOLLOW 6](#_Toc200146465)

[Таблица 7](#_Toc200146466)

[Код 8](#_Toc200146467)

[Результаты работы кода 12](#_Toc200146468)

# Вариант

S → ABCC

C → cccA | ccBB | cC | c

B → BBb | BBa | b

A → aAa | c

# Задание

Для грамматики в соответствии с вариантом необходимо:  
**1.** Устранить левую рекурсию (если необходимо).

**2.** Провести левую факторизацию грамматики (если необходимо).

**3.** Для полученной преобразованной грамматики построить  
множества FIRST и FOLLOW для нетерминальных символов  
грамматики.

**4.** Для преобразованной грамматики построить таблицу анализатора

и разработать программную реализацию этого анализатора.  
Примечание: если полученная грамматика окажется не LL(1)  
грамматикой — конфликты в таблице анализатора — то  
согласовать со мной изменения в грамматике с целью приведения  
её к LL(1) виду. Ввиду этого соображения вида: «но грамматика же  
не подходит под LL(1) анализатор – вот я ничего делать и не стал»  
— не работают.  
Результатом работы является работающий код анализатора.

**5.** Отчет должен включать:

a. Исходную грамматику;  
b. Отдельно (для каждого правила) действия по устранению  
прямой левой рекурсии и отдельно действия для левой  
факторизации.  
c. Преобразованную грамматику. Внесения изменений при  
обязательном согласовании со мной.  
d. Таблицы множеств FIRST и FOLLOW для нетерминалов;  
e. Таблица синтаксического анализатора;  
f. Реализацию синтаксического анализатора.  
g. Примеры корректных и ошибочных входных цепочек.  
h. Выводы

# Удаление левой рекурсии

**Правило S: S → ABCC**

Нет левой рекурсии.

**Правило C: C → cccA | ccBB | cC | c**

Нет левой рекурсии

**Правило B: B → BBb | BBa | b**

Это правило имеет прямую левую рекурсию вида B → Bα | β.

Здесь β = b.

α1 = Bb (часть после первого B в BBb)

α2 = Ba (часть после первого B в BBa)

Применяем стандартное правило устранения левой рекурсии:

B → β B'

B' → α1 B' | α2 B' | ε

Получаем:

B → b B'

B' → Bb B' | Ba B' | ε

**Правило A: A → aAa | c**

Нет левой рекурсии.

Грамматика после устранения левой рекурсии (промежуточная):

S → ABCC

C → cccA | ccBB | cC | c

B → b B'

B' → Bb B' | Ba B' | ε

A → aAa | c

# Левая факторизация

**Правило B' : B' → BbB' | BaB' | ε**

Общий префикс B у первых двух альтернатив.

Факторизуем:

B' → BB'' | ε

B''→ bB' | aB' | ε

**Правило C: C → cccA | ccBB | cC | c**

Все альтернативы начинаются с c. Факторизуем c:

C → cC'

C'→ cC'' | C | ε

C''→ cA | BB | ε

**Правило S и A:** не требуют левой факторизации.

Преобразованная грамматика:

S → ABCC

A → aAa | c

B → bB'

B' → B B'' | ε

B''→ bB' | aB' | ε

C → cC'

C'→ cC'' | C | ε

C''→ cA | BB | ε

# FIRST

**FIRST(A):**

Из A → aAa: a

Из A → c: c

=> FIRST(A) = {a, c}

**FIRST(B):**

Из B → bB': b

=> FIRST(B) = {b}

**FIRST(B''):**

Из B''→ bB': b

Из B''→ aB': a

Из B''→ ε: ε

=> FIRST(B'') = {a, b, ε}

**FIRST(B'):**

Из B' → B B'': FIRST(B) = {b}

Из B' → ε: ε

=> FIRST(B') = {b, ε}

**FIRST(C''):**

Из C''→ cA: c

Из C''→ BB: FIRST(B) = {b}

Из C''→ ε: ε

=> FIRST(C'') = {b, c, ε}

**FIRST(C):** (Вычисляем итеративно из-за C' → C)

Итерация 1: FIRST(C) = {}

Итерация 1: FIRST(C') зависит от FIRST(C).

C'→ cC'': c

C'→ C: пока пусто

C'→ ε: ε

=> FIRST(C') = {c, ε}

Из C → cC': FIRST(C) = FIRST(c) = {c}

Итерация 2: FIRST(C) = {c}

Итерация 2: FIRST(C'):

C'→ cC'': c

C'→ C: FIRST(C) = {c}

C'→ ε: ε

=> FIRST(C') = {c, ε}

=> FIRST(C) = {c}

**FIRST(S):**

Из S → ABCC: FIRST(A) = {a, c} (т.к. A не ε)

=> FIRST(S) = {a, c}

**Итог:**

FIRST(S) = {a, c}

FIRST(A) = {a, c}

FIRST(B) = {b}

FIRST(B') = {b, ε}

FIRST(B'') = {a, b, ε}

FIRST(C) = {c}

FIRST(C') = {c, ε}

FIRST(C'') = {b, c, ε}

# FOLLOW

FOLLOW(S) = {$}  
FOLLOW(A) = FIRST(B) + FOLLOW(C'') + {a} = {a, b, c, $}  
FOLLOW(B) = FIRST(B) + FIRST(B'') + FOLLOW(C'') = {b, c, $}  
FOLLOW(B') = FOLLOW(B) + FOLLOW(B'') = {b, c, $}  
FOLLOW(B'') = FOLLOW(B') = {b, c, $}  
FOLLOW(C) = FIRST(C) + {$} + FOLLOW(C') = {c, $}  
FOLLOW(C') = FOLLOW(C) = {c, $}  
FOLLOW(C'') = FOLLOW(C') = {c, $}

# Таблица

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Нетерминал** | **a** | **b** | **c** | **$** |
| **S** | S → ABCC |  | S → ABCC |  |
| **A** | A → aAa |  | A → c |  |
| **B** |  | B → bB' |  |  |
| **B'** |  | B''→ BB''  B'→ ε | B' → ε | B' → ε |
| **B''** | B''→ aB' | B''→ bB'  B''→ ε | B''→ ε | B''→ ε |
| **C** |  |  | C → cC' |  |
| **C'** |  |  | C'→ cC''  C'→ C  C'→ ε | C'→ ε |
| **C''** |  | C''→ BB | C''→ cA  C''→ ε | C''→ ε |

Есть конфликты, нужно исправить грамматику на:

S → ABCC

A → aAa | c

B → bB'

B' → bB' | aB' | d

C → cC'

C' → cC'' | d

C'' → cA | BB

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Нетерминал** | **a** | **b** | **c** | **d** | **$** |
| **S** | S → ABCC |  | S → ABCC |  |  |
| **A** | A → aAa |  | A → c |  |  |
| **B** |  | B → bB' |  |  |  |
| **B'** | B''→ aB' | B'→ bB |  | B'→ d |  |
| **C** |  |  | C → cC' |  |  |
| **C'** |  |  | C→ cC' | C'→ d |  |
| **C''** |  | C''→ BB | C''→ cA |  |  |

# Код

class LL1Parser:

def \_\_init\_\_(self):

self.terminals = {'a', 'b', 'c', 'd', '$'}

self.non\_terminals = {'S', 'A', 'B', 'B\'', 'C', 'C\'', 'C\''}

self.start\_symbol = 'S'

self.parsing\_table = {

'S': {

'a': ['A', 'B', 'C', 'C'],

'c': ['A', 'B', 'C', 'C']

},

'A': {

'a': ['a', 'A', 'a'],

'c': ['c']

},

'B': {

'b': ['b', 'B\'']

},

'B\'': {

'a': ['a', 'B\''],

'b': ['b', 'B\''],

'd': ['d'] # B' -> d

},

'C': {

'c': ['c', 'C\'']

},

'C\'': {

'c': ['c', 'C\'\''],

'd': ['d'] # C' -> d

},

'C\'\'': {

'b': ['B', 'B'],

'c': ['c', 'A']

}

}

self.error\_messages = {}

def add\_custom\_error(self, non\_terminal, terminal, message):

if non\_terminal not in self.error\_messages:

self.error\_messages[non\_terminal] = {}

self.error\_messages[non\_terminal][terminal] = message

def parse(self, input\_string):

input\_tokens = list(input\_string) + ['$']

stack = ['$', self.start\_symbol]

idx = 0

log = []

errors\_encountered = 0

log.append(f"{'Stack':<30} | {'Input':<30} | Action")

log.append("-" \* 80)

while stack[-1] != '$':

current\_stack\_str = ' '.join(stack)

current\_input\_str = ''.join(input\_tokens[idx:])

X = stack[-1]

a = input\_tokens[idx]

log\_entry\_prefix = f"{current\_stack\_str:<30} | {current\_input\_str:<30} | "

if X == a:

action = f"Match and pop '{a}'"

log.append(log\_entry\_prefix + action)

stack.pop()

idx += 1

elif X in self.terminals:

action = f"Error: Mismatch. Stack top: '{X}', Input: '{a}'"

log.append(log\_entry\_prefix + action)

errors\_encountered += 1

print("\nParsing Log:")

print("\n".join(log))

print(f"\nError {errors\_encountered}: Mismatch. Expected '{X}' but got '{a}' at position {idx}.")

return False, errors\_encountered

elif X in self.non\_terminals:

if a in self.parsing\_table[X]:

production = self.parsing\_table[X][a]

action = f"Apply rule: {X} -> {''.join(production)}"

log.append(log\_entry\_prefix + action)

stack.pop()

for symbol in reversed(production):

stack.append(symbol)

else: # Случай 4: Ошибка - нет правила в таблице M[X, a]

action = f"Error: No rule for M[{X}, {a}]"

log.append(log\_entry\_prefix + action)

errors\_encountered += 1

custom\_msg = self.error\_messages.get(X, {}).get(a)

error\_detail = f"No production rule for non-terminal '{X}' and input symbol '{a}' at position {idx}."

if custom\_msg:

error\_detail += f" ({custom\_msg})"

print("\nParsing Log:")

print("\n".join(log))

print(f"\nError {errors\_encountered}: {error\_detail}")

return False, errors\_encountered

else:

action = f"Error: Unknown symbol on stack '{X}'"

log.append(log\_entry\_prefix + action)

errors\_encountered += 1

print("\nParsing Log:")

print("\n".join(log))

print(f"\nError {errors\_encountered}: Unknown symbol '{X}' on stack.")

return False, errors\_encountered

if errors\_encountered > 0 and errors\_encountered >=2:

print(f"Stopped after {errors\_encountered} errors.")

pass

current\_stack\_str = ' '.join(stack)

current\_input\_str = ''.join(input\_tokens[idx:])

log\_entry\_prefix = f"{current\_stack\_str:<30} | {current\_input\_str:<30} | "

if stack[-1] == '$' and input\_tokens[idx] == '$':

log.append(log\_entry\_prefix + "Accept")

print("\nParsing Log:")

print("\n".join(log))

print("\nInput string accepted.")

return True, errors\_encountered

else:

log.append(log\_entry\_prefix + "Error: Input remaining or stack not empty")

errors\_encountered +=1

print("\nParsing Log:")

print("\n".join(log))

if input\_tokens[idx] != '$':

print(f"\nError {errors\_encountered}: Input remaining ('{''.join(input\_tokens[idx:])}') when parsing should have finished.")

else:

print(f"\nError {errors\_encountered}: Stack not empty ('{' '.join(stack)}') when input exhausted.")

return False, errors\_encountered

parser = LL1Parser()

# Промах 1: S не может начинаться с 'b'

parser.add\_custom\_error('S', 'b', "S must start with 'a' or 'c'.")

# Промах 2: A не может начинаться с 'b'

parser.add\_custom\_error('A', 'b', "A must start with 'a' or 'c'.")

# Промах 3: B' не может начинаться с 'c'

parser.add\_custom\_error('B\'', 'c', "B' must start with 'a', 'b' or 'd'.")

print("--- Valid chains ---")

# Пример 1

correct\_chain\_1 = "cbdcdcd"

print(f"\nParsing '{correct\_chain\_1}':")

result, errors = parser.parse(correct\_chain\_1)

print(f"Result: {'Accepted' if result else 'Rejected'}, Errors: {errors}")

# Пример 2

correct\_chain\_2 = "acabdcdcd"

print(f"\nParsing '{correct\_chain\_2}':")

result, errors = parser.parse(correct\_chain\_2)

print(f"Result: {'Accepted' if result else 'Rejected'}, Errors: {errors}")

# Пример 3

correct\_chain\_3 = "cbabbdcccAccbdbbbd"

print(f"\nParsing '{correct\_chain\_3}':")

result, errors = parser.parse(correct\_chain\_3)

print(f"Result: {'Accepted' if result else 'Rejected'}, Errors: {errors}")

print("\n--- Corrupted chains ---")

# Ошибка 1: S не может начинаться с 'b' (сработает пользовательское сообщение)

error\_chain\_1 = "bacd"

print(f"\nParsing '{error\_chain\_1}':")

result, errors = parser.parse(error\_chain\_1)

print(f"Result: {'Accepted' if result else 'Rejected'}, Errors: {errors}")

# Ошибка 2: После 'a' в A->aAa ожидается 'a' или 'c' (для внутреннего A), а не 'b'

print(f"\nParsing '{error\_chain\_2}':")

result, errors = parser.parse(error\_chain\_2)

print(f"Result: {'Accepted' if result else 'Rejected'}, Errors: {errors}")

# Ошибка 3: Неожиданный символ в конце

error\_chain\_3 = "cbdcdcde"

print(f"\nParsing '{error\_chain\_3}':")

result, errors = parser.parse(error\_chain\_3)

print(f"Result: {'Accepted' if result else 'Rejected'}, Errors: {errors}")

# Ошибка 4: Слишком короткая строка

error\_chain\_4 = "cb"

print(f"\nParsing '{error\_chain\_4}':")

result, errors = parser.parse(error\_chain\_4)

print(f"Result: {'Accepted' if result else 'Rejected'}, Errors: {errors}")

# Ошибка 5: Проверка B' -> c (ошибка, так как B' не может начинаться с 'c')

error\_chain\_5 = "cbcdcdcd"

print(f"\nParsing '{error\_chain\_5}':")

result, errors = parser.parse(error\_chain\_5)

print(f"Result: {'Accepted' if result else 'Rejected'}, Errors: {errors}")

# Результаты работы кода







