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

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

```
--- Корректные цепочки ---

Parsing 'cbdcdcd':

Parsing Log:
Stack                          | Input                          | Action
-------------------------------------------------------------------------------
$ S                            | cbdcdcd$                       | Apply rule: S -> ABCC
$ C C B A                      | cbdcdcd$                       | Apply rule: A -> c
$ C C B c                      | cbdcdcd$                       | Match and pop 'c'
$ C C B                        | bdcdcd$                        | Apply rule: B -> bB'
$ C C B' b                     | bdcdcd$                        | Match and pop 'b'
$ C C B'                       | dcdcd$                         | Apply rule: B' -> d
$ C C d                        | dcdcd$                         | Match and pop 'd'
$ C C                          | cdcd$                          | Apply rule: C -> cC'
$ C C' c                       | cdcd$                          | Match and pop 'c'
$ C C'                         | dcd$                           | Apply rule: C' -> d
$ C d                          | dcd$                           | Match and pop 'd'
$ C                            | cd$                            | Apply rule: C -> cC'
$ C' c                         | cd$                            | Match and pop 'c'
$ C'                           | d$                             | Apply rule: C' -> d
$ d                            | d$                             | Match and pop 'd'
$                              | $                              | Accept

Input string accepted.
Result: Accepted, Errors: 0

Parsing 'acabdcdcd':

Parsing Log:
Stack                          | Input                          | Action
-------------------------------------------------------------------------------
$ S                            | acabdcdcd$                     | Apply rule: S -> ABCC
$ C C B A                      | acabdcdcd$                     | Apply rule: A -> aAa
$ C C B a A a                  | acabdcdcd$                     | Match and pop 'a'
$ C C B a A                    | cabdcdcd$                      | Apply rule: A -> c
$ C C B a c                    | cabdcdcd$                      | Match and pop 'c'
$ C C B a                      | abdcdcd$                       | Match and pop 'a'
$ C C B                        | bdcdcd$                        | Apply rule: B -> bB'
$ C C B' b                     | bdcdcd$                        | Match and pop 'b'
$ C C B'                       | dcdcd$                         | Apply rule: B' -> d
$ C C d                        | dcdcd$                         | Match and pop 'd'
$ C C                          | cdcd$                          | Apply rule: C -> cC'
$ C C' c                       | cdcd$                          | Match and pop 'c'
$ C C'                         | dcd$                           | Apply rule: C' -> d
$ C d                          | dcd$                           | Match and pop 'd'
$ C                            | cd$                            | Apply rule: C -> cC'
$ C' c                         | cd$                            | Match and pop 'c'
$ C'                           | d$                             | Apply rule: C' -> d
$ d                            | d$                             | Match and pop 'd'
$                              | $                              | Accept

Input string accepted.
Result: Accepted, Errors: 0
```

```
Parsing 'cbabbdcccAccbdbbbd':

Parsing Log:
Stack                          | Input                          | Action
-------------------------------------------------------------------------------
$ S                            | cbabbdcccAccbdbbbd$            | Apply rule: S -> ABCC
$ C C B A                      | cbabbdcccAccbdbbbd$            | Apply rule: A -> c
$ C C B c                      | cbabbdcccAccbdbbbd$            | Match and pop 'c'
$ C C B                        | babbdcccAccbdbbbd$             | Apply rule: B -> bB'
$ C C B' b                     | babbdcccAccbdbbbd$             | Match and pop 'b'
$ C C B'                       | abbdcccAccbdbbbd$              | Apply rule: B' -> aB'
$ C C B' a                     | abbdcccAccbdbbbd$              | Match and pop 'a'
$ C C B'                       | bbdcccAccbdbbbd$               | Apply rule: B' -> bB'
$ C C B' b                     | bbdcccAccbdbbbd$               | Match and pop 'b'
$ C C B'                       | bdcccAccbdbbbd$                | Apply rule: B' -> bB'
$ C C B' b                     | bdcccAccbdbbbd$                | Match and pop 'b'
$ C C B'                       | dcccAccbdbbbd$                 | Apply rule: B' -> d
$ C C d                        | dcccAccbdbbbd$                 | Match and pop 'd'
$ C C                          | cccAccbdbbbd$                  | Apply rule: C -> cC'
$ C C' c                       | cccAccbdbbbd$                  | Match and pop 'c'
$ C C'                         | ccAccbdbbbd$                   | Apply rule: C' -> cC''
$ C C'' c                      | ccAccbdbbbd$                   | Match and pop 'c'
$ C C''                        | cAccbdbbbd$                    | Apply rule: C'' -> cA
$ C A c                        | cAccbdbbbd$                    | Match and pop 'c'
$ C A                          | Accbdbbbd$                     | Match and pop 'A'
$ C                            | ccbdbbbd$                      | Apply rule: C -> cC'
$ C' c                         | ccbdbbbd$                      | Match and pop 'c'
$ C'                           | cbdbbbd$                       | Apply rule: C' -> cC''
$ C'' c                        | cbdbbbd$                       | Match and pop 'c'
$ C''                          | bdbbbd$                        | Apply rule: C'' -> BB
$ B B                          | bdbbbd$                        | Apply rule: B -> bB'
$ B B' b                       | bdbbbd$                        | Match and pop 'b'
$ B B'                         | dbbbd$                         | Apply rule: B' -> d
$ B d                          | dbbbd$                         | Match and pop 'd'
$ B                            | bbbd$                          | Apply rule: B -> bB'
$ B' b                         | bbbd$                          | Match and pop 'b'
$ B'                           | bbd$                           | Apply rule: B' -> bB'
$ B' b                         | bbd$                           | Match and pop 'b'
$ B'                           | bd$                            | Apply rule: B' -> bB'
$ B' b                         | bd$                            | Match and pop 'b'
$ B'                           | d$                             | Apply rule: B' -> d
$ d                            | d$                             | Match and pop 'd'
$                              | $                              | Accept

Input string accepted.
Result: Accepted, Errors: 0
```

```
--- Ошибочные цепочки ---

Parsing 'bacd':

Parsing Log:
Stack                          | Input                          | Action
-------------------------------------------------------------------------------
$ S                            | bacd$                          | Error: No rule for M[S, b]

Error 1: No production rule for non-terminal 'S' and input symbol 'b' at position 0. (S должен начинаться с 'a' или 'c'.)
Result: Rejected, Errors: 1

Parsing 'ab':

Parsing Log:
Stack                          | Input                          | Action
-------------------------------------------------------------------------------
$ S                            | ab$                            | Apply rule: S -> ABCC
$ C C B A                      | ab$                            | Apply rule: A -> aAa
$ C C B a A a                  | ab$                            | Match and pop 'a'
$ C C B a A                    | b$                             | Error: No rule for M[A, b]

Error 1: No production rule for non-terminal 'A' and input symbol 'b' at position 1. (A должен начинаться с 'a' или 'c'.)
Result: Rejected, Errors: 1

Parsing 'cbdcdcde':

Parsing Log:
Stack                          | Input                          | Action
-------------------------------------------------------------------------------
$ S                            | cbdcdcde$                      | Apply rule: S -> ABCC
$ C C B A                      | cbdcdcde$                      | Apply rule: A -> c
$ C C B c                      | cbdcdcde$                      | Match and pop 'c'
$ C C B                        | bdcdcde$                       | Apply rule: B -> bB'
$ C C B' b                     | bdcdcde$                       | Match and pop 'b'
$ C C B'                       | dcdcde$                        | Apply rule: B' -> d
$ C C d                        | dcdcde$                        | Match and pop 'd'
$ C C                          | cdcde$                         | Apply rule: C -> cC'
$ C C' c                       | cdcde$                         | Match and pop 'c'
$ C C'                         | dcde$                          | Apply rule: C' -> d
$ C d                          | dcde$                          | Match and pop 'd'
$ C                            | cde$                           | Apply rule: C -> cC'
$ C' c                         | cde$                           | Match and pop 'c'
$ C'                           | de$                            | Apply rule: C' -> d
$ d                            | de$                            | Match and pop 'd'
$                              | e$                             | Error: Input remaining or stack not empty

Error 1: Input remaining ('e$') when parsing should have finished.
Result: Rejected, Errors: 1

Parsing 'cb':
```