Задание 12.1 Построение таблицы АМП синтаксического анализатора

Пусть дана грамматика G. Напишите функцию, которая строит таблицу автомата с магазинной памятью нисходящего предиктивного синтаксического анализатора.

Вход/выход: Грамматика считывается из файла.

Пример работы программы 12.1

Грамматика в файле grammar.txt

```
E -> T E'
E' -> + T E'
E' -> e
T -> F T'
T' -> * F T'
T' -> e
F -> ( E )
F -> id
```

Вызовы функции FIRST

```
G.buildSATable();
G.printSATable();
```

Вывод на экран (пример)

```
1. E -> T E'
2. E' -> + T E'
3. E' -> e
4. T -> F T'
5. T' -> * F T'
6. T' -> e
7. F -> ( E )
8. F -> id

      id    +     *     (     )     $
-------------------------------------
E     R1,R                R1,R
E'          R2,R                R3,R  R3,R
T     R4,R                R4,R
T'          R6,R  R5,R          R6,R  R6,R
F     R8,R                R7,R
id    P,A
+           P,A
*                 P,A
(                       P,A
)                             P,A
$                                   Accept
```

План выполнения задания

Добавьте в класс `Grammar` метод `buildSATable()`, который строит таблицу предиктивного синтаксического анализатора, и метод `printSATable()`, который печатает таблицу на экран (в любом человеко-читаемом формате). Метод `buildSATable()` строит таблицу синтаксического анализа **M**, используя алгоритм:

1. для каждого правила `A -> α` грамматики G выполнить:
   - для каждого терминала a из `FIRST(α)` добавить **`Rep(α^T),R`** в ячейку **`M[A][a]`**, где α^T – перевернутая строка α;
   - если ε входит в `FIRST(α)`, то для каждого символа b из `FOLLOW(A)` добавить **`Rep(α^T),R`** в **`M[A][b]`**;
2. для каждого терминала x грамматики G добавляем **`Pop,A`** в ячейку `M[x][x]`;
3. добавляем **`Accept`** в ячейку **`M[$][$]`**;
4. добавляем **`Reject`** во все оставшиеся пустые ячейки.

Метод `buildSATable()` должен возвращать True, если таблица не содержит коллизий (ячеек с двумя и более записями), и False, если грамматика не принадлежит классу LL(1). Добавьте вызов метода `buildSATable()` в функцию `loadGrammar()`. Если метод вернул False, необходимо напечатать об этом на экран.

Оценка задания 12.1

12.1.1. метод `buildSATable()` 80 %.

12.1.2. метод `printSATable()` 20 %.

Баллы за каждую задачу включают:

- правильность решения задания 40 %;
- оформление кода согласно http://prog.tversu.ru/pr3/codeStyle.pdf 10 %;
- модульные тесты, включающие дополнительные примеры 50%.

Задание 12.2 Драйвер табличного синтаксического анализатора

Напишите драйвер нисходящего синтаксического анализатора, который для заданной грамматики по построенной таблице и данной последовательности лексем вернет истину, если последовательность лексем принадлежит языку, задаваемому грамматикой, и ложь в противном случае.

Вход/выход: Грамматика считывается из файла, последовательность лексем (терминалов) вводится с клавиатуры.

Пример работы программы 12.2

Грамматика в файле grammar.txt

```
E -> T E'
E' -> + T E'
E' -> - T E'
E' -> e
T -> F T'
T' -> * F T'
T' -> / F T'
T' -> e
F -> ( E )
F -> id
```

Ввод с клавиатуры

```
(id + id) * id
(id + id
id * * id
id * id + id
```

Вывод на экран

```
True
False
False
True
```

План выполнения задания

Добавьте в класс `Grammar` метод `parse()`, который получает на вход строку из терминалов, разделенных пробелами, и возвращает булевское значение: истина, если строка терминалов принадлежит грамматике, и ложь в противном случае. Метод должен проэмулировать работу автомата с магазинной памятью на таблице, построенной с помощью функции `buildSATable()` из упражнения 10.1.

Фрагмент тестирующего кода:

```
Grammar G = Grammar();
if(G.loadGrammar(ifstream("grammar.txt")))
{
	if(G.parse("(id + id) * id"))
	{
		std::cout << "True" << std::endl;
	}
	else
	{
		std::cout << "False" << std::endl;
	}

	if(G.parse("(id + id"))
	{
		std::cout << "True" << std::endl;
	}
	else
	{
		std::cout << "False" << std::endl;
```

```
        }

        if(G.parse("id * * id"))
        {
                std::cout << "True" << std::endl;
        }
        else
        {
                std::cout << "False" << std::endl;
        }

        if(G.parse("id * id + id"))
        {
                std::cout << "True" << std::endl;
        }
        else
        {
                std::cout << "False" << std::endl;
        }
}
else
{
        std::cout << "Error: not LL(1) grammar" << std::endl;
}
```

Результат работы приведен в условии задачи (пример работы программы 10.2).

Оценка задания 12.2

10.2. метод `parse()` 100 %.

Баллы за задачу включают:

- правильность решения задания 40 %;
- оформление кода согласно http://prog.tversu.ru/pr3/codeStyle.pdf 10 %;
- модульные тесты, включающие дополнительные примеры 50%.