

Задание 9.1 Функция FIRST

Пусть дана грамматика G . Напишите функцию $FIRST(\alpha)$, где α – любая последовательность терминалов и нетерминалов грамматики G , которая возвращает множество терминалов, с которых начинаются строки, выводимые из α . Если из α выводимо ϵ , то ϵ тоже входит в $FIRST(\alpha)$. Перед выполнением упражнения обязательно прочтите секцию «Комментарии к заданиям».

Вход/выход: Грамматика считывается из файла, выражения для функции $FIRST$ вводятся с клавиатуры.

Пример работы программы 9.1

Грамматика в файле <code>grammar.txt</code>	Вывод на экран
<pre>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</pre>	<pre>[(, id] [+, -, e] [(, id] [*, /, e] [(, id] []] [id]</pre>
Вызовы функции FIRST	
<pre>G.FIRST("E"); G.FIRST("E'"); G.FIRST("T"); G.FIRST("T'"); G.FIRST("F"); G.FIRST(""); G.FIRST("id");</pre>	

План выполнения задания

Задача 9.1.1

Реализуйте класс `Grammar`¹ для хранения грамматики. Класс должен включать в себя методы:

- метод `loadGrammar(istream &stream)`, считывающий грамматику из потока;
- оператор вывода в поток, печатающий грамматику (в любом формате).

Фрагмент тестирующего кода:

```
Grammar G = Grammar();
G.loadGrammar(istream("grammar.txt"));
std::cout << G;
```

Результат работы:

```
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
```

¹ Готовый шаблон класса можно загрузить с сайта <http://prog.tversu.ru>

Задача 9.1.2

Добавьте в класс `Grammar` новое поле типа «словарь» `FIRSTForG`, которое будет хранить значения функции `FIRST`, предварительно вычисленные для каждого символа грамматики, а также добавьте следующие методы:

- метод `initFIRSTWithTerminalsAndEpsilon()`, который:
 - для каждого нетерминала N из левых частей правил в грамматике G создает запись для `FIRSTForG[N]` с пустым множеством в качестве значения;
 - для каждого терминала X в грамматике G добавляет в `FIRSTForG[X]` множество $\{X\}$, состоящее из одного этого терминала;
 - для каждого правила вида $X \rightarrow \varepsilon$ добавляет 'e' в `FIRSTForG[X]`;

добавьте вызов этого метода в метод `loadGrammar()`;

- метод `printFIRST(ostream& stream)`, который печатает в поток текущее содержимое словаря `FIRSTForG` (в любом формате).

Фрагмент тестирующего кода:

```
Grammar G = Grammar();
G.loadGrammar(ifstream("grammar.txt"));
G.printFIRST(std::cout);
```

Результат работы:

```
E' = [e]
T' = [e]
+ = [+]
- = [-]
* = [*]
/ = [/]
( = [(]
) = [)]
id = [id]
```

Задача 9.1.3

Добавьте в класс `Grammar` метод `FIRST(str)`, который:

1. принимает на вход произвольную последовательность терминалов/нетерминалов грамматики G ; можно реализовать любой способ представления str – либо как строку символов, либо как список строк, либо как список объектов пользовательского типа;
2. возвращает список терминалов, с которых начинаются слова, выводимые из str , находя их с помощью следующего алгоритма:
 - если $str = 'e'$, то возвращаем 'e';
 - если str – одиночный терминал/нетерминал, то возвращаем `FIRSTForG[str]`;
 - пусть $str = X_1 X_2 X_3 \dots X_n$, где X_i – произвольный терминал или нетерминал грамматики:
 - a) добавляем к результату значение `FIRST(X1)` кроме 'e'; если 'e' входит в `FIRST(X1)`, то переходим к шагу (b), если нет, то выходим из функции `FIRST`;
 - b) добавляем к результату значение `FIRST(X2)` кроме 'e'; если 'e' входит в `FIRST(X2)`, то переходим к шагу (c), если нет, то выходим из функции `FIRST`;
 - c) ...
 - d) добавляем к результату значение `FIRST(Xn)` кроме 'e'; если 'e' входит в `FIRST(Xn)`, то добавляем 'e' в результат, иначе просто выходим из функции `FIRST`.

Фрагмент тестирующего кода (в целях упрощения записи str задается строкой символов):

```
Grammar G = Grammar();
G.loadGrammar(ifstream("grammar.txt"));

vector<string> str;
str.push_back("+"); str.push_back("T"); str.push_back("E'");
std::cout << G.FIRST(str);
str.clear();
```

```
str.push_back("E"); str.push_back("+"); str.push_back("T");
std::cout << G.FIRST(str);
str.clear();

str.push_back("E"); str.push_back("T");
std::cout << G.FIRST(str);
```

Результат работы:

```
[+]
[+]
[e]
```

Задача 9.1.4

Добавьте в класс `Grammar` метод `initFIRSTWithNonTerminals()`, который инициализирует словарь `FIRSTForG` нетерминалами грамматики `G` по следующему алгоритму:

1. для каждого правила грамматики `G` вида $X \rightarrow \alpha$, добавляем `FIRST(α)` в `FIRSTForG[X]`;
2. если во время выполнения шага 1 в `FIRSTForG` было добавлено хотя бы одно новое значение, то переходим снова к шагу 1, иначе выходим из функции.

Добавьте вызов этого метода в метод `loadGrammar()`.

Фрагмент тестирующего кода:

```
Grammar G = Grammar();
G.loadGrammar(istream("grammar.txt"));

vector<string> alpha;
alpha.push_back("E");
std::cout << G.FIRST(alpha);
alpha.clear();

alpha.push_back("E");
std::cout << G.FIRST(alpha);
alpha.clear();

alpha.push_back("T");
std::cout << G.FIRST(alpha);
alpha.clear();

alpha.push_back("T");
std::cout << G.FIRST(alpha);
alpha.clear();

alpha.push_back("F");
std::cout << G.FIRST(alpha);
alpha.clear();

alpha.push_back("id");
std::cout << G.FIRST(alpha);
alpha.clear();
```

Результат работы приведен в условии задачи (пример работы программы 9.1).

Оценка задания 9.1

Задание 9.1.1. класс <code>Grammar</code> , <code>loadGrammar</code> , оператор вывода для грамматики	20 %.
Задание 9.1.2. <code>initFIRSTWithTerminalsAndEpsilon</code>	20 %.
Задание 9.1.3. <code>FIRST</code>	30 %.
Задание 9.1.4. <code>initFIRSTWithNonTerminals</code>	30 %.

Баллы за каждую задачу включают:

- правильность решения задания 60 %;
- оформление кода согласно <http://prog.tversu.ru/pr3/codeStyle.pdf> 10 %;
- программа компилируется без предупреждений компилятора на 4-ом уровне 10%;
- модульные тесты, включающие дополнительные примеры 20%.

Задание 9.2 Функция FOLLOW

Пусть дана грамматика G . Напишите функцию $FOLLOW(A)$, где A – произвольный нетерминал грамматики G , возвращающую множество терминалов, которые могут располагаться непосредственно после (справа от) A в некоторой сентенциальной форме. Если A окажется крайним справа символом в некоторой сентенциальной форме, то в $FOLLOW(A)$ также входит $\$$ (маркер конца входного потока). Перед выполнением упражнения обязательно прочтите секцию «Комментарии к заданиям».

Вход/выход: Грамматика считывается из файла, выражения для функции $FOLLOW$ вводятся с клавиатуры.

Пример работы программы 9.2

Грамматика в файле <code>grammar.txt</code>	Вывод на экран
<pre> 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 </pre>	<pre> [') ', '\$'] [') ', '\$'] ['+', '-', ') ', '\$'] ['+', '-', ') ', '\$'] ['+', '-', '*', '/', ') ', '\$'] </pre>
Вызовы функции <code>FOLLOW</code>	
<pre> G.FOLLOW("E"); G.FOLLOW("E'"); G.FOLLOW("T"); G.FOLLOW("T'"); G.FOLLOW("F"); </pre>	

План выполнения задания

Задача 9.2.1

Добавьте в класс `Grammar` новое поле/словарь `FOLLOWForG`, которое будет хранить значения функции `FOLLOW`, предварительно вычисленные для каждого нетерминала грамматики, а также добавьте метод `initFOLLOW()`, который:

- для каждого нетерминала N из левых частей правил в грамматике G создает запись в словаре `FOLLOWForG[N]` с пустым множеством в качестве значения;
- добавляет '\$' в `FOLLOWForG[S]`, где S – стартовый нетерминал грамматики G .

Задача 9.2.2

Добавьте в класс `Grammar` новый метод `calculateFOLLOW()`, который вычислит значение `FOLLOW` для каждого нетерминала грамматики по следующему алгоритму:

1. вызываем метод `initFOLLOW()`;
2. для каждого правила грамматики G выполняем одно из двух действий:
 - если правило вида $A \rightarrow \alpha B \beta$, то добавляем в `FOLLOWForG[B]` все элементы `FIRST(β)` кроме 'e';
 - если правило вида $A \rightarrow \alpha B$ или $A \rightarrow \alpha B \beta$, где `FIRST(β)` содержит 'e', то добавляем в `FOLLOWForG[B]` все элементы `FOLLOWForG[A]`;

- если во время выполнения шага 2 в FOLLOWForG было добавлено хотя бы одно новое значение, то переходим снова к шагу 2, иначе выходим из функции.

Добавьте вызов этого метода в метод loadGrammar().

Задача 9.2.3

Добавьте в класс Grammar метод FOLLOW(), который:

- принимает на вход произвольный нетерминал A грамматики G;
- возвращает значение FOLLOWForG[A].

Фрагмент тестирующего кода:

```
Grammar G = Grammar();
G.loadGrammar(istream("grammar.txt"));
std::cout << G.FOLLOW(string("E"));
std::cout << G.FOLLOW(string("E'"));
std::cout << G.FOLLOW(string("T"));
std::cout << G.FOLLOW(string("T'"));
std::cout << G.FOLLOW(string("F"));
```

Результат работы приведен в условии задачи (пример работы программы 9.2).

Оценка задания 9.2

Задание 9.2.1. initFOLLOW	15 %.
Задание 9.2.2. calculateFOLLOW	80 %.
Задание 9.2.3. FOLLOW	5 %.

Баллы за каждую задачу включают:

- правильность решения задания 60 %;
- оформление кода согласно <http://prog.tversu.ru/pr3/codeStyle.pdf> 10 %;
- программа компилируется без предупреждений компилятора на 4-ом уровне 10%;
- модульные тесты, включающие дополнительные примеры 20%.

Комментарии к заданиям

- Для выполнения задания можно загрузить шаблон класса с сайта <http://prog.tversu.ru>. Типы данных внутренних полей класса для хранения грамматики, FIRSTForG и FOLLOWForG можно изменить.
- Создание дополнительных функций разрешено, удаление/комбинирование описанных выше функций приводит к снижению количества баллов.
- Во входных и выходных данных ε обозначается маленькой латинской буквой e.
- Оба задания используют один и тот же класс Grammar.
- Можно предполагать, что во входном файле задана корректная грамматика.
- Нетерминал в левой части первого правила считается стартовым.
- Формат описания грамматики в текстовом файле можно выбрать любой.