

Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное
учреждение высшего образования «Национальный исследовательский
университет ИТМО»

Факультет инфокоммуникационных технологий

Математическая лингвистика

Практическая работа №5-6

Выполнили:

студент группы К34422

Малаев Степан Геннадьевич

Проверил:

доцент практики, КТН

Болгова Екатерина Владимировна

Санкт-Петербург

2025

Ход работы.

1. Выбрать нотацию языка программирования. Описать его грамматику.

Был выбран язык Go.

Грамматика:

1. Программа и структура модуля:

- a. <Программа> ::= <Пакет> { <Импорт> } { <Функция> }
- b. <Пакет> ::= “package” <Идентификатор>
- c. <Импорт> ::= “import” <СтроковыйЛитерал>

2. Определение функции и блока операторов:

- a. <Функция> ::= “func” <Идентификатор> “(” [<Параметры>] “)” <Блок>
- b. <Параметры> ::= <Параметр> { “,” <Параметр> }
- c. <Параметр> ::= <Идентификатор> <Тип>
- d. <Блок> ::= “{” { <Оператор> } “}”

3. Операторы и конструкции управления:

- a. <Оператор> ::= <Объявление> | <Присвоение> | <Условие> | <Цикл> | <Выражение> “;”
- b. <Условие> ::= “if” <Выражение> <Блок> [“else” <Блок>]
- c. <Цикл> ::= “for” <Выражение> <Блок>
 - | “for” <Присвоение> “;” <Выражение> “;”

<Выражение> <Блок>

- d. <Присвоение> ::= <Идентификатор> “=” <Выражение>

4. Выражения, идентификаторы и литералы:

- a. $\langle \text{Выражение} \rangle ::= \langle \text{Терм} \rangle \{ ("+" \mid "-") \langle \text{Терм} \rangle \}$
- b. $\langle \text{Терм} \rangle ::= \langle \text{Фактор} \rangle \{ ("'" \mid "/" \mid "%") \langle \text{Фактор} \rangle \}$
- c. $\langle \text{Фактор} \rangle ::= \langle \text{Литерал} \rangle \mid \langle \text{Идентификатор} \rangle \mid (" \langle \text{Выражение} \rangle ")$
- d. $\langle \text{Литерал} \rangle ::= \langle \text{Число} \rangle \mid \langle \text{СтроковыйЛитерал} \rangle$
- e. $\langle \text{Идентификатор} \rangle ::= [\text{A-Za-z}][\text{A-Za-z0-9}_]$

2. Что должно быть в грамматике:

1. Алфавит:

Содержит латинские буквы, цифры, символы “_” и знаки, используемые в операторах и разделителях, по типу: ((), {}, [], ;, ,).

2. Множество терминальных символов, множество символов-разделителей на лексемы:

Ключевые слова (package, import, func, var, if, else, for, return), операторы (+, -, *, /, %, =) и разделители (скобки, запятая, точка с запятой).

3. Множество символов, определяющее математические операции (сложение, вычитание, деление, умножение):

Определяются символами: +, -, *, /, %

4. Множество правил определяющее: идентификаторы (переменные), зарезервированные слова, числа:

- a. Идентификатор: $[\text{A-Za-z}][\text{A-Za-z0-9}_]^*$
- b. Зарезервированные слова: package, import, func, var, if, else, for, return.
- c. Числа, задаются стандартными регулярными выражениями для целых и вещественных чисел: $-?\backslash d+(\backslash.\backslash d+)?([eE][-+]?\backslash d+)?$

5. В качестве конструкций языка программирования необходимо определить: следование, условие, цикл:

- a. Последовательность операторов разделяются точкой с запятой или переводом строки.
- b. Конструкция условия if с необязательным else.
- c. Цикл представляется конструкцией for, допускающей как простую форму с условием, так и форму с тремя частями: инициализация, условие, итерация.

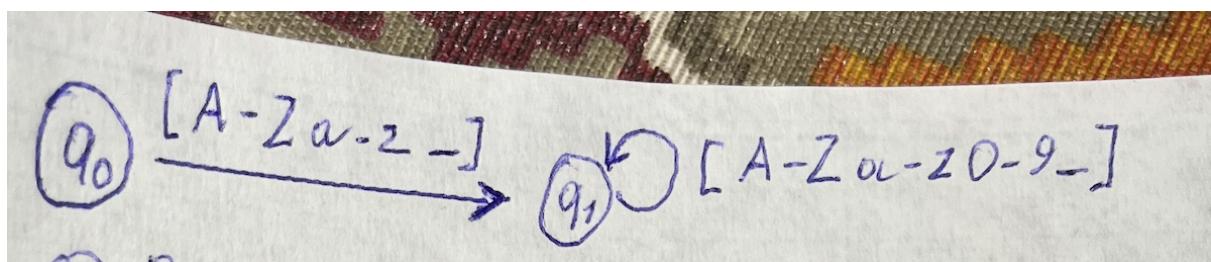
3. Построить конечный автомат (-ы) для лексического анализатора согласно построенной грамматики.

Автомат для идентификаторов.

Структура:

- q_0 — начальное:
 - По символу из [A-Za-z] или _ $\rightarrow q_1$.
 - Иначе \rightarrow отказ.
- q_1 — принимающее:
 - По символу [A-Za-z0-9_] остаёмся в q_1 .
 - Иначе \rightarrow завершение лексемы.

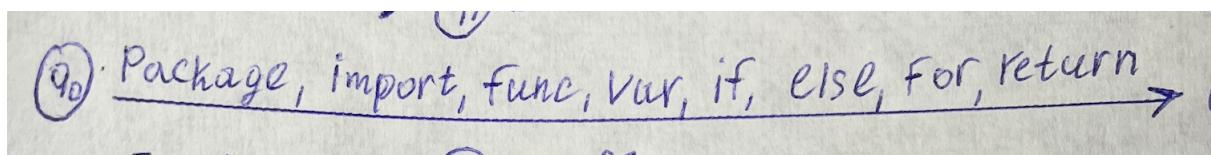
Таким образом, любая строка, удовлетворяющая регулярному выражению $^*[A-Za-z][A-Za-z0-9_]*\$$, будет принята.



Автомат для зарезервированных слов

Структура:

- q_0 — начальное:
 - По первой букве определяем ветвь.
 - Если не совпадает ни с одним началом ключевого слова → отказ.
- Дальше для каждого ключевого слова строим цепочку состояний, где каждый переход соответствует одной букве слова.
- Последняя буква слова ведёт в конечное, принимающее состояние.



Автомат для чисел

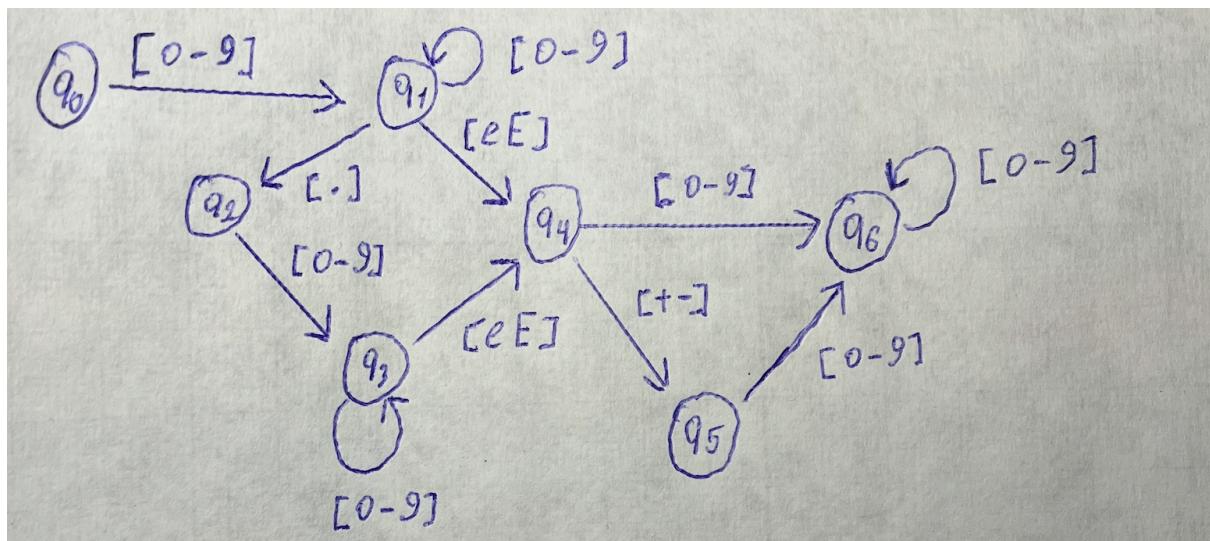
Структура:

1. q_0 — начальное состояние:
 - а. По цифре из [0-9] → переход в q_1 .
 - б. Иначе → отказ.
2. q_1 — принимающее, целая часть:
 - а. По цифре из [0-9] остаёмся в q_1 .
 - б. По символу . → переход в q_2 , начало дробной части.
 - с. По символу е или Е → переход в q_4 , начало экспоненты.
 - д. Иначе → завершение лексемы, принимается целое число.
3. q_2 — после точки, ожидаем дробную часть:
 - а. По цифре из [0-9] → переход в q_3 .
 - б. Иначе → отказ.

4. q_3 — принимающее, дробная часть:
- По цифре из [0-9] остаётся в q_3 .
 - По символу е или Е → переход в q_4 , начало экспоненты.
 - Иначе → завершение лексемы, принимается вещественное число без экспоненты.
5. q_4 — после символа экспоненты:
- По символу + или - → переход в q_5 .
 - По цифре из [0-9] → переход в q_6 .
 - Иначе → отказ.
6. q_5 — после знака экспоненты:
- По цифре из [0-9] → переход в q_6 .
 - Иначе → отказ.
7. q_6 — принимающее, часть экспоненты:
- По цифре из [0-9] остаётся в q_6 .
 - Иначе → завершение лексемы, принимается число с экспонентой.

Соответствует регулярному выражению:

$^{\text{[0-9]}}+(\text{[0-9]})?(\text{[eE]}[\text{+-}]?[\text{0-9}])? \$$

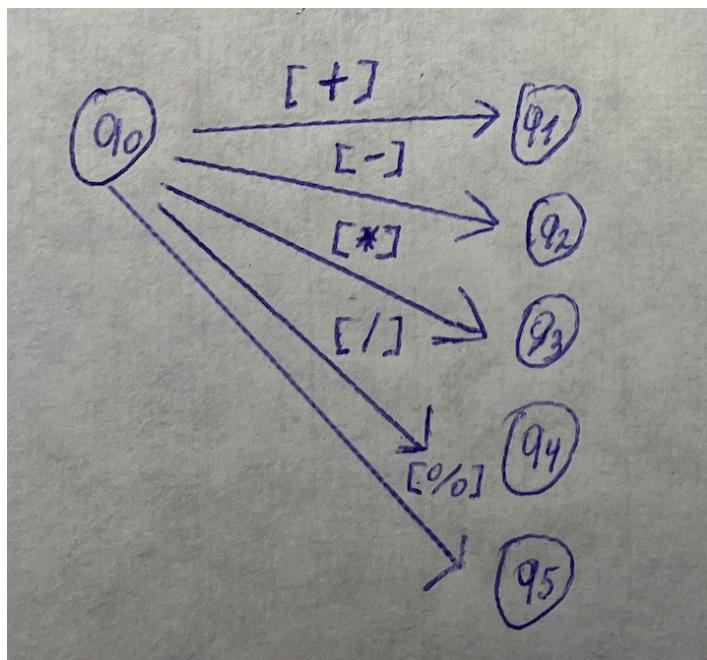


Автомат для операций

Структура:

- q_0 — начальное:
 - По $+$ $\rightarrow q_1$, принимающее.
 - По $-$ $\rightarrow q_2$, принимающее.
 - По $*$ $\rightarrow q_3$, принимающее.
 - По $/$ $\rightarrow q_4$, принимающее.
 - По $\%$ $\rightarrow q_5$, принимающее.
 - Иначе \rightarrow отказ.

Каждое q_i — конечное состояние для соответствующей операции.

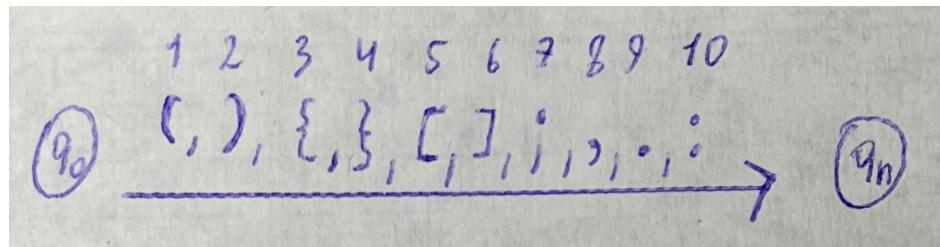


Автомат для символов-разделителей

Структура:

1. q_0 — начальное:

- a. По (→ q_1 , принимающее.
- b. По) → q_2 , принимающее.
- c. По { → q_3 , принимающее.
- d. По } → q_4 , принимающее.
- e. По [→ q_5 , принимающее.
- f. По] → q_6 , принимающее.
- g. По ; → q_7 , принимающее.
- h. По , → q_8 , принимающее.
- i. По . → q_9 , принимающее.
- j. По : → q_{10} , принимающее.
- k. Иначе → отказ.



4. Написать лексический анализатор. Вход — текст программы; выход — есть/нет лексических ошибок, если ошибок нет — классы распознанных лексем (идентификаторы, зарезервированные слова, спец символы, терминальные символы, числа). Для тестирования необходимо подготовить тестовые наборы текстов программ.

Ознакомиться с кодом реализации можно по следующей ссылки: [\[ссылка\]](#)

Было проведено 3 тест-кейса:

1. Фрагмент корректной программы с пакетом, импортом, функцией, условным оператором и вызовом печати:

```
package main
import "fmt"
func main() {
    var x = 42
    if x > 0 {
        fmt.Println("Positive")
    } else {
        fmt.Println("Non-positive")
    }
}
```

Результат анализа:

----- Тест 1 -----

Лексемы:

```
TokenInstance(token=RESERVED, value='package', start=0, end=7)
TokenInstance(token=IDENTIFIER, value='main', start=8, end=12)
TokenInstance(token=RESERVED, value='import', start=13, end=19)
TokenInstance(token=STRING, value=""fmt"", start=20, end=25)
TokenInstance(token=RESERVED, value='func', start=26, end=30)
TokenInstance(token=IDENTIFIER, value='main', start=31, end=35)
TokenInstance(token=SEPARATOR, value='(', start=35, end=36)
TokenInstance(token=SEPARATOR, value=')', start=36, end=37)
TokenInstance(token=SEPARATOR, value='{', start=38, end=39)
TokenInstance(token=RESERVED, value='var', start=44, end=47)
TokenInstance(token=IDENTIFIER, value='x', start=48, end=49)
TokenInstance(token=OPERATOR, value='=', start=50, end=51)
TokenInstance(token=NUMBER, value='42', start=52, end=54)
TokenInstance(token=RESERVED, value='if', start=59, end=61)
```

```
TokenInstance(token=IDENTIFIER, value='x', start=62, end=63)
TokenInstance(token=OPERATOR, value='>', start=64, end=65)
TokenInstance(token=NUMBER, value='0', start=66, end=67)
TokenInstance(token=SEPARATOR, value='{', start=68, end=69)
TokenInstance(token=IDENTIFIER, value='fmt', start=78, end=81)
TokenInstance(token=SEPARATOR, value='.', start=81, end=82)
TokenInstance(token=IDENTIFIER, value='Println', start=82, end=89)
TokenInstance(token=SEPARATOR, value='(', start=89, end=90)
TokenInstance(token=STRING, value='"Positive"', start=90, end=100)
TokenInstance(token=SEPARATOR, value=')', start=100, end=101)
TokenInstance(token=SEPARATOR, value='}', start=106, end=107)
TokenInstance(token=RESERVED, value='else', start=108, end=112)
TokenInstance(token=SEPARATOR, value='{', start=113, end=114)
TokenInstance(token=IDENTIFIER, value='fmt', start=123, end=126)
TokenInstance(token=SEPARATOR, value='.', start=126, end=127)
TokenInstance(token=IDENTIFIER, value='Println', start=127, end=134)
TokenInstance(token=SEPARATOR, value='(', start=134, end=135)
TokenInstance(token=STRING,      value='"Non-positive"',      start=135,
end=149)
TokenInstance(token=SEPARATOR, value=')', start=149, end=150)
TokenInstance(token=SEPARATOR, value='}', start=155, end=156)
TokenInstance(token=SEPARATOR, value='}', start=157, end=158)
Лексический анализ завершён успешно. Ошибок нет.
```

2. Программа с ошибкой – в числе присутствует недопустимый символ \$:

```
package main
func main() {
    var x = 3$14
}
```

Результат анализа:

----- Тест 2 -----

Лексическая ошибка: Лексическая ошибка в позиции 40: '\$14\n}'

3. Фрагмент корректной программы с определением функции, целочисленными, вещественными литералами, экспоненциальной записью и арифметическими операциями:

```
package calc
func add(a int, b int) int {
    return a + b
}
func main() {
    var result = add(3.14, -2.71e-1)
    fmt.Println(result)
}
```

Результат анализа:

----- Тест 3 -----

Лексемы:

```
TokenInstance(token=RESERVED, value='package', start=0, end=7)
TokenInstance(token=IDENTIFIER, value='calc', start=8, end=12)
TokenInstance(token=RESERVED, value='func', start=13, end=17)
TokenInstance(token=IDENTIFIER, value='add', start=18, end=21)
TokenInstance(token=SEPARATOR, value='(', start=21, end=22)
TokenInstance(token=IDENTIFIER, value='a', start=22, end=23)
```

```
TokenInstance(token=IDENTIFIER, value='int', start=24, end=27)
TokenInstance(token=SEPARATOR, value=',', start=27, end=28)
TokenInstance(token=IDENTIFIER, value='b', start=29, end=30)
TokenInstance(token=IDENTIFIER, value='int', start=31, end=34)
TokenInstance(token=SEPARATOR, value=')', start=34, end=35)
TokenInstance(token=IDENTIFIER, value='int', start=36, end=39)
TokenInstance(token=SEPARATOR, value='{', start=40, end=41)
TokenInstance(token=RESERVED, value='return', start=46, end=52)
TokenInstance(token=IDENTIFIER, value='a', start=53, end=54)
TokenInstance(token=OPERATOR, value='+', start=55, end=56)
TokenInstance(token=IDENTIFIER, value='b', start=57, end=58)
TokenInstance(token=SEPARATOR, value='}', start=59, end=60)
TokenInstance(token=RESERVED, value='func', start=61, end=65)
TokenInstance(token=IDENTIFIER, value='main', start=66, end=70)
TokenInstance(token=SEPARATOR, value='(', start=70, end=71)
TokenInstance(token=SEPARATOR, value=')', start=71, end=72)
TokenInstance(token=SEPARATOR, value='{', start=73, end=74)
TokenInstance(token=RESERVED, value='var', start=79, end=82)
TokenInstance(token=IDENTIFIER, value='result', start=83, end=89)
TokenInstance(token=OPERATOR, value='=', start=90, end=91)
TokenInstance(token=IDENTIFIER, value='add', start=92, end=95)
TokenInstance(token=SEPARATOR, value='(', start=95, end=96)
TokenInstance(token=NUMBER, value='3.14', start=96, end=100)
TokenInstance(token=SEPARATOR, value=',', start=100, end=101)
TokenInstance(token=NUMBER, value='-2.71e-1', start=102, end=110)
TokenInstance(token=SEPARATOR, value=')', start=110, end=111)
TokenInstance(token=IDENTIFIER, value='fmt', start=116, end=119)
TokenInstance(token=SEPARATOR, value='.', start=119, end=120)
TokenInstance(token=IDENTIFIER, value='Println', start=120, end=127)
TokenInstance(token=SEPARATOR, value='(', start=127, end=128)
```

TokenInstance(token=IDENTIFIER, value='result', start=128, end=134)

TokenInstance(token=SEPARATOR, value=')', start=134, end=135)

TokenInstance(token=SEPARATOR, value='}', start=136, end=137)

Лексический анализ завершён успешно. Ошибок нет.

5. Дополнить ПР5 синтаксическим анализатором. Синтаксический анализатор на основе грамматики и построенных КА должен определить: корректность синтаксических конструкций, корректность расстановки скобок. На выходе должно быть: после лексической проверки, синтаксическая проверка: все конструкции записаны верно (присваивание, выражения, условия, циклы). Вывести количество верных конструкций либо ошибки с описанием ошибок.

Ознакомиться с кодом реализации можно по следующей ссылке: [\[ссылка\]](#)

Было проведено 3 тест-кейса:

1. Два условия с блоком else, одно из которых является вложенным.

Пример корректной как для Go, так и для описанной грамматики.

```
if x + 5 > 0 {  
    x = x + 1;  
    if x - 3 < 4 {  
        x = x + 2;  
    } else {  
        x = x - 2;  
    }  
} else {  
    y = 0;  
}
```

Результат анализа:

----- Синтаксический тест 1 -----

Синтаксический анализ завершён успешно.

2. Два цикла for с 3 и 1 частями, является корректным примером кода:

```
for i = 10; i > 0; i = i - 1 {  
    sum = sum + i;  
}  
for x + 1 > 0 {  
    x = x + 2;  
}
```

Результат анализа:

----- Синтаксический тест 2 -----

Синтаксический анализ завершён успешно.

3. Условие с блоком else с отсутствующей закрывающей фигурной скобкой для блока if. Имеет синтаксическую ошибку:

```
if a + 5 > 0 {  
    a = a + 2;  
else {  
    a = a + 3;  
}
```

Результат анализа:

----- Синтаксический тест 3 -----

Синтаксическая ошибка: Неожиданный токен 'else' на позиции 30

Вывод.

В ходе данной практической работы была разработана упрощенная грамматика языка Go, построены соответствующие конечные автоматы для лексического анализа, а также реализован рекурсивный спусковой синтаксический анализатор. Выполненные тестовые примеры подтвердили корректность идентификации лексем, включая операторы, разделители, идентификаторы, числа и зарезервированные слова, а также верное распознавание основных синтаксических конструкций присваивания, условных операторов с ветвью else, циклов for в двух формах. При наличии структурных ошибок анализатор корректно выявляет и сообщает о синтаксической несогласованности.