编译原理第3次作业

姓名:胡瑞康

学号:22336087

Exercise 3.1

Give the recognized tokens of the following program in Pascal.

```
function max(i, j: integer): integer;
{return the maximum of integers i and j}
begin
  if i > j then max := i else max := j
end;
```

(关键字, `function`)

(标识符, `max`)

(分隔符, `(`)

(标识符, `i`)

(分隔符, `,`)

(标识符, `j`)

(分隔符, `:`)

(标识符, `integer`)

(分隔符, `)`)

(分隔符, `:`)

(标识符, `integer`)

(分隔符, `;`)

(注释, `{return the maximum of integers i and j}`)

(关键字, `begin`)

(关键字, `if`)

(标识符, `i`)

(运算符, `>`)

(标识符, `j`)

(关键字, `then`)

(标识符, `max`)

(运算符, `:=`)

(标识符, `i`)

(关键字, `else`)

(标识符, `max`)

(运算符, `:=`)

(标识符, `j`)

(关键字, `end`)

(分隔符, `;`)

Exercise 3.2

0.1 题目

(DBv2, Ch.3, pp.125, ex.3.3.2) Describe the languages denoted by the following regular expressions:

- `a (a | b)* a`
- `a* b a* b a* b a*`

正则表达式 `a (a | b)* a`

该正则表达式表示所有以字母 **a** 开头并以字母 **a** 结尾，中间部分由任意个（包括 0 个）字母 **a** 或 **b** 的字符串。

例如：`aa`、`aba`、`abba`、`aabbaa` 等。

正则表达式 `a* b a* b a* b a*`

该正则表达式描述的语言是所有只由字母 **a** 和 **b** 构成且恰好包含三个 **b** 的字符串。
具体来看：

- `a*` 表示任意个（包括 0 个）字母 **a**；
- 中间的 `b` 是固定的；
- 总共有三个 `b` 分别被不同的 `a*` 包围。

也就是说，该语言中的任一字符串都可以分解为：

任意个 **a** + 第一个 **b** + 任意个 **a** + 第二个 **b** + 任意个 **a** + 第三个 **b** + 任意个 **a**。

例如：`bbb`、`ababb`、`aababaaba` 等。

Exercise 3.3

(DBv2, Ch.3, pp.125, ex.3.3.4)
Most languages are case sensitive, so keywords can be written only one way, and the regular expressions describing their lexemes are very simple.

However, some languages, like Pascal and SQL, are case insensitive. For example, the SQL keyword `SELECT` can also be written `select`, `Select`, `Or sELEcT`.

Show how to write a regular expression for a keyword in a case insensitive language. Illustrate your idea by writing the expression for `SELECT` in SQL.

在不借助正则表达式引擎的内置选项（如 `(?i)`）的情况下，可以通过为每个字母提供一个字符集来匹配其大写和小写形式。

例如，对于 SQL 中的关键字 `SELECT`，可以写成如下正则表达式：

```
[sS][eE][lL][eE][cC][tT]
```

该表达式的含义是：

- `[sS]` 匹配字符 `s` 或 `S`
- `[eE]` 匹配字符 `e` 或 `E`
- `[lL]` 匹配字符 `l` 或 `L`
- `[eE]` 匹配字符 `e` 或 `E`
- `[cC]` 匹配字符 `c` 或 `C`
- `[tT]` 匹配字符 `t` 或 `T`

这样，无论 `SELECT` 是以什么大小写组合出现，这个正则表达式都可以正确匹配。

Exercise 3.4

Given the following regular expression `1*(0 | 01)*`

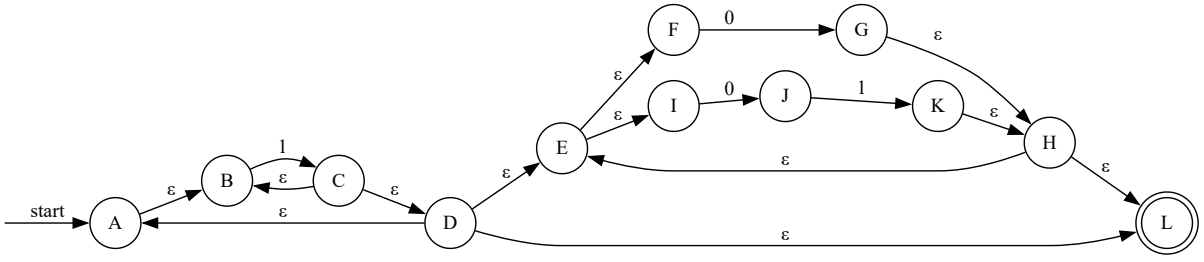
---

1 (1) Transform it to an equivalent finite automaton.

`A,B,C,D`节点群构成`1*`

`E,F,G,H,I,J,K`节点群构成`(0|01)`

`D-L`节点群构成`(0|01)*`



---

2 (2) Construct an equivalent DFA for the result of exercise (1).

令 $q_0 = \varepsilon-closure(\{A\}) = \{A, B, D, E, F, I, L\}$

$\delta(q_0, 0) = \varepsilon-closure(\{G, J\}) = \{G, J, H, L, E, F, I\} = q_1$

$\delta(q_0, 1) = \varepsilon-closure(\{C\}) = \{B, C, D, E, F, I, L\} = q_2$

$\delta(q_1, 0) = \varepsilon-closure(\{G, J\}) = q_1$

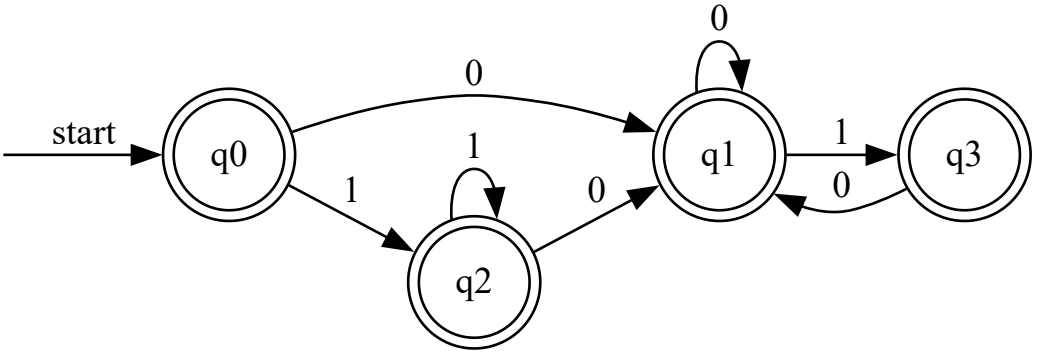
$\delta(q_1, 1) = varepsilon\text{-closure}(\{K\}) = \{K, H, E, F, I, L\} = q_3$

$\delta(q_2, 0) = \varepsilon-closure(\{G, J\}) = q_1$

$\delta(q_2, 1) = \varepsilon-closure(\{C\}) = q_2$

$\delta(q_3, 0) = \varepsilon-closure(\{G, J\}) = q_1$

$\delta(q_3, 1) = \emptyset$



---

3 (3) Reduce the result of (2) and get a reduced DFA.

1.将状态划分为终态和非终态，该题只有终态$\{q_0, q_1, q_2, q_3\}$

2.寻找子集中的不等价状态

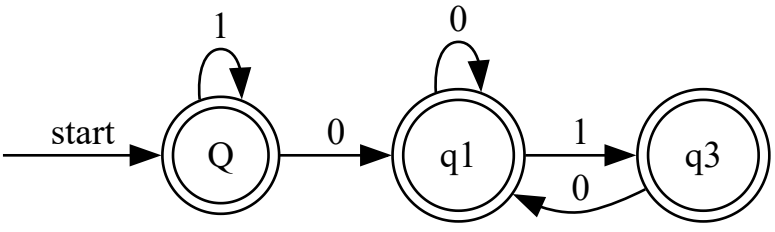
q0 输入0 变为 q1, 输入1 变为 q2

q2 输入0 变为 q1，输入1 变为 q2

因此可以合并

$\{q_0, q_1, q_2, q_3\}$ => $\{q_0, q_2\}\{q_1\}\{q_3\}$

3.令Q代表$\{q_0, q_2\}$



Exercise 3.5

Given the alphabet Σ = { z, o, / }, a comment in a program over Σ begins with "/o" and ends with "o/". Embedded comments are not permitted.

(1) Draw a DFA that recognizes nothing but all the comments in the source programs.

(2) Write a single regular expression that exactly describes all the comments in the source programs.

给定字母表 Σ = { z, o, / }，程序中以 Σ 为字符集的注释以 "/o" 开头，以 "o/" 结尾。不允许有嵌套注释。
(1) 画出一个确定有限自动机（DFA），它只识别源程序中的所有注释。
(2) 写出一个正则表达式，精确描述源程序中的所有注释。

1 DFA

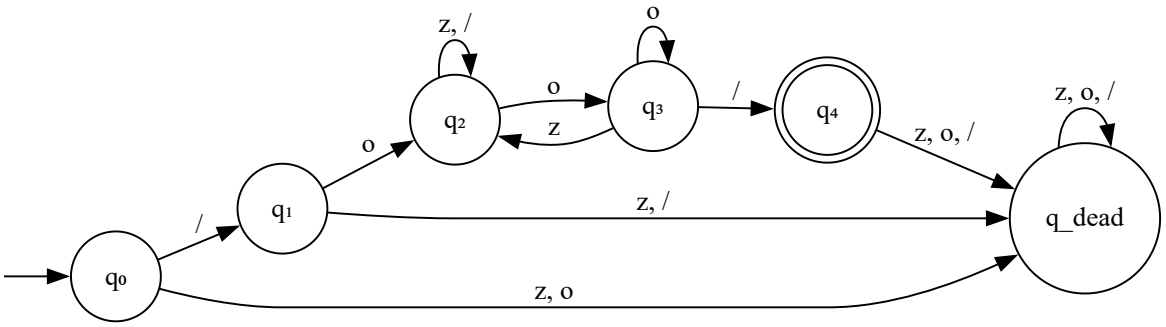
状态定义：

- **q_0**：初始状态，期待注释的开始。
- **q_1**：已经看到 "/"，期待 "o" 以开始注释。
- **q_2**：进入注释内部，最后一个字符不是 "o"。
- **q_3**：进入注释内部，最后一个字符是 "o"（需要检查下一个字符是否为 "/"）。
- **q_4**：已经看到 "o/"，如果字符串在这里结束则接受。
- **q_dead**：死状态，表示字符串无效。

状态转移规则：

- 从 **q_0（初始状态）**：
  - 输入 / → q_1（注释可能开始）
  - 输入 z, o → q_dead（必须以 "/" 开头）
- 从 **q_1（看到 "/"）**：
  - 输入 o → q_2（"/o" 完成，进入注释）
  - 输入 z, / → q_dead（"/z" 或 "//" 无效）
- 从 **q_2（注释内部，非 "o" 结尾）**：
  - 输入 z → q_2（继续）
  - 输入 / → q_2（继续）
  - 输入 o → q_3（记录 "o"，检查下一个字符）
- 从 **q_3（注释内部，以 "o" 结尾）**：
  - 输入 z → q_2（"o z" 合法）
  - 输入 o → q_3（"o o" 合法，继续观察）
  - 输入 / → q_4（"o/" 出现，可能结束）
- 从 **q_4（看到 "o/"）**：
  - 输入 z, o, / → q_dead（"o/" 后不能有字符）
- 从 **q_dead（死状态）**：
  - 输入 z, o, / → q_dead（保持无效）
- 初始状态：q_0
- 接受状态：q_4（表示完整注释结束）

图片：



2　编写正则表达式

因为直接用课堂定义的标准正则表达式语法较难通过我自己编写的几个测试用例，在此使用了Python特殊的一些正则表达式符号便于编写。

让正则表达式只捕获注释部分，并允许注释结束后后面有额外字符（符合“源程序中”注释可能嵌入其他代码的情形）。

同时，为了确保提取的是“最早出现的”结束分界符，需要用非贪婪匹配。

正则表达式如下：

```
pattern = r'^/o((?:(?!o/)[zo/])*?)o/.*$'
```

- `^/o`：要求以“/o”开头。
- `((?:(?!o/)[zo/])*?)`：用非贪婪模式捕获注释内容，其中
  - `[zo/]` 表示允许的字符；
  - `(?!o/)` 的负向前瞻确保在捕获过程中不把后面紧跟的“o/”误认为是内容的一部分。
- `o/`：作为结束分界符。
- `.*$`：允许结束分界符后有额外字符（例如程序中注释后可能还有其他代码）。

使用下面的文件对正则表达式结果进行验证

```
import re

#需要完成的正则表达式
pattern = r'^/o((?:(?!o/)[zo/])*?)o/.*$'

# 测试函数，使用 re.fullmatch 来确保整个字符串都被匹配
def test_comment(test_case):
    comment, expected = test_case
    match = re.fullmatch(pattern, comment)
    print(f"测试字符串: '{comment}'")
    print(f"预期结果: '{expected}'")

    if match:
        groups = match.groups()
        actual = groups[0] if groups else ""
        print(f"实际结果: '{actual}'")
        if actual == expected:
            print("✓ 测试通过！")
        else:
            print("✗ 测试失败！")
    else:
        print(f"实际结果: False(不匹配)")
        if expected == False:
            print("✓ 测试通过！")
        else:
            print("✗ 测试失败！")

# 测试用例：(输入字符串，预期的groups[0]结果)
test_cases = [
    ("/oo/", ""),
    ("/ozoo/", "zo"),
    ("/oz/oo/", "z/o")
```

```
    ("/oz/zoo/", "z/zo"),
    ("/ooz/zoo/", "oz/zo"),
    ("/ooz/oo/", "oz/o"),
    ("/ozo/z/", "z"),
    ("/ooo/z/", "o"),
    ("/zzz/", False),
    ("/o/", False),
]

print("开始运行测试用例...")
for tc in test_cases:
    test_comment(tc)
    print("--------------------------------")
```

测试结果如下：

```
(base) PS D:\homework\CompilerTheoryAssignment\Theory\3> python test.py
开始运行测试用例...
测试字符串: '/oo/'
预期结果: ''
实际结果: ''
✓ 测试通过!
--------------------------------
测试字符串: '/ozoo/'
预期结果: 'zo'
实际结果: 'zo'
✓ 测试通过!
--------------------------------
测试字符串: '/oz/oo/'
预期结果: 'z/o'
实际结果: 'z/o'
✓ 测试通过!
--------------------------------
测试字符串: '/oz/zoo/'
预期结果: 'z/zo'
实际结果: 'z/zo'
✓ 测试通过!
--------------------------------
测试字符串: '/ooz/zoo/'
预期结果: 'oz/zo'
实际结果: 'oz/zo'
✓ 测试通过!
--------------------------------
测试字符串: '/ooz/oo/'
预期结果: 'oz/o'
实际结果: 'oz/o'
✓ 测试通过!
--------------------------------
测试字符串: '/ozo/z/'
预期结果: 'z'
实际结果: 'z'
✓ 测试通过!
--------------------------------
测试字符串: '/ooo/z/'
预期结果: 'o'
实际结果: 'o'
```

✓ 测试通过!

测试字符串: '/zzz/'
预期结果: 'False'
实际结果: False(不匹配)
✓ 测试通过!

测试字符串: '/o/'
预期结果: 'False'
实际结果: False(不匹配)
✓ 测试通过!
