ソフトウェア演習 1
第 1 回レポート課題

J2200071 齊藤 隆斗

2024 年 5 月 7 日

1　課題 2.3

正規表現の字句解析ルーチンを作成し、正規表現入力を字句解析して得られた、トークンと意味値 (あるときのみ) の列を表示するプログラムを作成せよ.

2　実行結果

実行例 1: 正規表現の文字に対する字句解析が行えているかを確認

```
$ ./kadai1 '(a|b).c*\ed\0'
LPAR
LETTER(a)
VERT
LETTER(b)
RPAR
CONC
LETTER(c)
AST
EPSILON
LETTER(d)
EMPTY
EOREG
```

実行例 2: エスケープシーケンスの挙動が期待通りであることを確認

```
$ ./kadai1 '\(ab\c\)\*\\\.\d'
LETTER(()
LETTER(a)
LETTER(b)
LETTER(c)
```

```
LETTER())
LETTER(*)
LETTER(\)
LETTER(.)
LETTER(d)
EOREG
```

実行例 3: 最後の文字がエスケープシーケンスだった場合、そのエスケープシーケンスを無視して出力することを確認

```
$ ./kadai1 '(a|b)*c\'
LPAR
LETTER(a)
VERT
LETTER(b)
RPAR
AST
LETTER(c)
EOREG
```

3 プログラムの流れ

字句解析のプログラムの流れついて述べる．まず、文字列を 1 文字だけ読み込む. ここで、ヌル文字であった場合、EOREG を出力してプログラムを停止する．そうでなく、読み込んだ文字列が対応する正規表現の入力文字であると判定された場合、その正規表現のトークンを出力し、存在するときのみ意味値も出力する．しかし、対応する正規表現の入力文字が \0 や、\e のようにエスケープシーケンスを含むような 2 文字であるものがある．したがって、エスケープシーケンスが読み込まれたとき、通常と異なる制御を行う. 具体的には、エスケープシーケンスが読み込まれたら、更にもう 1 文字読む. それが 0 であれば EMPTY、e であれば EPLSILON、ヌル文字であれば EOREG、それ以外の文字が読み込まれたら意味値がその読み込んだ文字であるようなトークンである LETTER((読み込んだ文字)) を出力する．このエスケープシーケンスの処理においても、ヌル文字であった場合は処理が停止することに注意する．同様の処理を入力文字列の終端に達するまで行う．

4 考察

考えたことや工夫したことは 3 つある．

1 つ目は、入力文字に対する分岐を switch 文で工夫することによって実現したことである．正規表現の対応する入力文字にマッチしているかどうかを判定する部分で、エスケープシーケンスを考えなければ、正規表現に対応する入力文字は常に 1 文字であるため、サンプルプログラムの switch 文で対応することができる．しかし、$\emptyset$ や ϵ に対応する入力文字列はそれぞれ、\0,\e であり 2 文字となっている．switch において case のラ

ベルは整数の定数でなくてはいけないため、1 文字ずつしか読み込むことができない．よって、サンプルプログラムの switch 文を同じように使うことによって分岐を実装することはできない．そこで、1 つ目の switch 文においてエスケープシーケンスにマッチした後、次の 1 文字を読み込み、さらにその文字に対して分岐処理を行うことで $\emptyset$ や ϵ の 2 文字の入力文字にも対応することできた．つまり、エスケープシーケンスにマッチする case において、さらに switch 文をネストすることで解決した．

2 つ目は、文字列の最後にエスケープシーケンスが現れた際の分岐処理で簡潔なプログラムで処理できるようにしたことである．文字列の最後にエスケープシーケンスが入力された場合にどのように処理するべきかについて考えた．エスケープシーケンスを読み込んだ後にヌル文字を読み込んだ場合に EOREG を出力することで、エスケープシーケンスが文字列の最後に来てしまった場合に、そのエスケープシーケンスを無視するという挙動にすることでプログラムを簡潔にすることができた．

3 つ目は、文字列の読み込み方と今回のプログラムの相性について考えた．C のライブラリには文字列比較の関数があり、2 文字の文字列の処理にこの関数を使用することを考えたが、文字列を 1 文字目から終端まで検討していく今回のプログラムでは上で述べたような switch 文をネストして分岐を制御するほうが相性が良いと考えた．なぜならば、2 文字の分岐はエスケープシーケンスが読み込まれたときのみであり、1 文字目は常に \ であるため、残りの 1 文字の分岐が行えれば十分であり、わざわざ文字列比較の関数を使用する必要はないからである．