Übung 2

**Inhaltsverzeichnis**

[**Aufgabe a)** 2](#_Toc384366321)

[Zusammenfassung 2](#_Toc384366322)

[Lösungsidee 2](#_Toc384366323)

[Quellcode 3](#_Toc384366324)

[Testfälle 3](#_Toc384366325)

[**Aufgabe b)** 4](#_Toc384366326)

[Zusammenfassung 4](#_Toc384366327)

[Lösungsidee 4](#_Toc384366328)

[Quellcode 4](#_Toc384366329)

[Testfälle 5](#_Toc384366330)

[**Aufgabe c)** 6](#_Toc384366331)

[Zusammenfassung 6](#_Toc384366332)

[Lösungsidee 6](#_Toc384366333)

[Quellcode 6](#_Toc384366334)

[Testfälle 6](#_Toc384366335)

[**Aufgabe d)** 8](#_Toc384366336)

[Zusammenfassung 8](#_Toc384366337)

[Lösungsidee 8](#_Toc384366338)

[Quellcode 8](#_Toc384366339)

[Testfälle 8](#_Toc384366340)

[**Aufgabe e)** 9](#_Toc384366341)

[Zusammenfassung 9](#_Toc384366342)

[Lösungsidee 9](#_Toc384366343)

[Quellcode 9](#_Toc384366344)

[Testfälle 9](#_Toc384366345)

[Erweiterung 10](#_Toc384366346)

[Lösungsidee 10](#_Toc384366347)

[Quellcode 10](#_Toc384366348)

[Quellcode Gesamtaufgabe 11](#_Toc384366349)

# **Aufgabe a)**

## Zusammenfassung

Als Gesamtaufgabe soll mit Hilfe des BruteForce-Algorithmus die Suche eines Patterns in einem String durchgeführt werden.

## Lösungsidee

Für die erste Aufgabe reicht es hierbei aus, den BruteForce-Algorithmus im Standard zu verwenden, da hierbei keine Regular Expressions benötigt werden.

Die grundlegende Funktionsweise des BruteForce-Algorithmus lässt sich wie folgt beschreiben:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| i | **1** | 2 | 3 | 4 | 5 | 6 |
| String | S | t | r | i | n | g |
| Pattern | t | r | o |  |  |  |
| j | **1** | 2 | 3 |  |  |  |

Beim ersten Suchen nach von Pattern im String kommt es zu einem Mismatch.

Als nächsten Schritt wird im String um ein Zeichen weitergegangen:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| i | 1 | **2** | 3 | 4 | 5 | 6 |
| String | S | t | r | i | n | g |
| Pattern | t | r | o |  |  |  |
| j | **1** | 2 | 3 |  |  |  |

Das Pattern p[j] und s[i] stimmen jetzt überein, beide wandern also um eine Stelle nach vorne:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| i | **1** | 2 | **3** | 4 | 5 | 6 |
| String | S | t | r | i | n | g |
| Pattern | t | r | o |  |  |  |
| j | **1** | **2** | 3 |  |  |  |

Auch hier stimmen die Zeichen, beide Indizes wandern eins nach vorne:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| i | 1 | 2 | 3 | **4** | 5 | 6 |
| String | S | t | r | i | n | g |
| Pattern | t | r | o |  |  |  |
| j | 1 | 2 | **3** |  |  |  |

An dieser Stelle tritt ein Mismatch auf, die Folge daraus ist i um eine Stelle weiterwandert, und j auf eins zurückwandert:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| i | 1 | 2 | 3 | 4 | **5** | 6 |
| String | S | t | r | i | n | g |
| Pattern | t | r | o |  |  |  |
| j | **1** | 2 | 3 |  |  |  |

## Quellcode

**FUNCTION** BruteForceA **(** s**,** p **:** **STRING)** **:** INTEGER**;**

**VAR**

sLen**,** pLen**,** i**,** j**,** pos **:** INTEGER**;**

**BEGIN**

sLen **:=** Length**(**s**);**

pLen **:=** Length**(**p**);**

pos **:=** 0**;**

i **:=** 1**;** // erste Stelle im String

// solange nicht gefunden -> pos = 0

// und noch nicht das Ende erreicht

**WHILE** **(**pos **=** 0**)** **AND** **(**i **<=** sLen **-** pLen **+** 1**)** **DO** **BEGIN**

j **:=** 1**;**

**WHILE** **(**j **<=** pLen**)** **AND** **(**s**[**i **+** j **-** 1**]** **=** p**[**j**])** **DO**

Inc**(**j**);**

**IF** j **>** pLen **THEN**

pos **:=** i

**ELSE**

Inc**(**i**);**

**END;**//while

BruteForceA **:=** pos**;**

**END;**//BruteForceA

## Testfälle

|  |  |  |
| --- | --- | --- |
| String | Hagenbrg | ✓ |
| Pattern | brg |
| Erwartetes Ergebnis | 6 |
| Tatsächliches Ergebnis | 6 |

|  |  |  |
| --- | --- | --- |
| String | Hagenberg | ✓ |
| Pattern | berg |
| Erwartetes Ergebnis | 6 |
| Tatsächliches Ergebnis | 6 |

|  |  |  |
| --- | --- | --- |
| String | Hagenburg | ✓ |
| Pattern | berg |
| Erwartetes Ergebnis | 0 |
| Tatsächliches Ergebnis | 0 |

# **Aufgabe b)**

## Zusammenfassung

In dieser Aufgabe soll der Algorithmus um ein Sonderzeichen **.** als Regular Expression ergänzt werden. Der **.** steht dabei für **1** beliebiges Zeichen das im String vorkommen darf, der Rest des Patterns muss jedoch matchen.

## Lösungsidee

Der bereits bestehende Brute-Force Algorithmus wird ergänzt und ergibt auch ein Match wenn nicht nur s[i] und p[j] identisch sind, sondern auch wenn p[j] = **.** ist.

**IF** **(** **(**p**[**j**]** **=** '.' **)** **OR** **(**p**[**j**]** **=** s**[**i **+** index**])** **)** **THEN**

Inc**(**j**)**

## Quellcode

**FUNCTION** BruteForceRegEx **(** s**,** p **:** **STRING)** **:** INTEGER**;**

**VAR**

sLen**,** pLen**,** i**,** j**,** index**,** pos **:** INTEGER**;**

spl **:** **STRING;**

valid **:** BOOLEAN**;**

**BEGIN**

sLen **:=** Length**(**s**);**

pLen **:=** Length**(**p**);**

pos **:=** 0**;**

spl **:=** ''**;**

i **:=** 1**;** // erste Stelle im String

// begin only if regex is valid

**IF** **(**checkRegex**(**p**))** **THEN** **BEGIN**

**WHILE** **(**pos **=** 0**)** **AND** **(**i **<=** sLen**)** **DO** **BEGIN**

valid **:=** TRUE**;**

j **:=** 1**;**

index **:=** 0**;**

**WHILE** **(** **(**j **<=** pLen**)** **AND** **(**valid **=** TRUE**)** **AND** **(**index **<=** sLen**)** **)** **DO** **BEGIN**

**IF** **(** **(**p**[**j**]** **=** '.' **)** **OR** **(**p**[**j**]** **=** s**[**i **+** index**])** **)** **THEN**

Inc**(**j**)**

**ELSE** **IF** **(**

**(**p**[**j**]** **=** '^'**)** **AND**

**(**p**[**j**+**1**]** **=** '['**)** **AND**

**(**stringMatchesRegEx**(**s**[**i **+** index**],** Split**(**p**,** j**,** pLen**),** j**))**

**)** **THEN** **BEGIN**

spl **:=** Split**(**p**,** j**+**2**,** pLen**);**

j **:=** j **+** BruteForceA**(**spl**,** ']'**)** **+** 2**;**

**END** **ELSE** **IF** **(**

**(**p**[**j**]** **=** '['**)** **AND**

**(**stringMatchesRegEx**(**s**[**i **+** index **],** Split**(**p**,** j**,** pLen**),** j**))**

**)** **THEN** **BEGIN**

spl **:=** Split**(**p**,** j**+**1**,** pLen**);**

j **:=** j **+** BruteForceA**(**spl**,** ']'**)** **+** 1**;**

**END** **ELSE** **IF** **(**

**(**p**[**j**]** **=** '^'**)** **AND**

**(**p**[**j**+**1**]** **<>** '['**)** **AND**

**(**p**[**j**+**1**]** **<>** s**[**i **+** index**])**

**)** **THEN**

j **:=** j **+** 2

**ELSE**

valid **:=** FALSE**;**

index **:=** index **+** 1**;**

**END;**//while

**IF** **(** **(**valid **=** TRUE**)** **AND** **(**j **>** pLen**)** **)** **THEN**

pos **:=** i**;**

Inc**(**i**);**

**END;**//while

**END;**//if

BruteForceRegEx **:=** pos**;**

**END;**//BruteForceRegEx

## Testfälle

|  |  |  |
| --- | --- | --- |
| String | Hagenbrg | ✓ |
| Pattern | b.rg |
| Erwartetes Ergebnis | 0 |
| Tatsächliches Ergebnis | 0 |

|  |  |  |
| --- | --- | --- |
| String | Hagenberg | ✓ |
| Pattern | b.rg |
| Erwartetes Ergebnis | 6 |
| Tatsächliches Ergebnis | 6 |

|  |  |  |
| --- | --- | --- |
| String | Hagenburg | ✓ |
| Pattern | b.rg |
| Erwartetes Ergebnis | 6 |
| Tatsächliches Ergebnis | 6 |

|  |  |  |
| --- | --- | --- |
| String | Hagenberg | ✓ |
| Pattern | b..rg |
| Erwartetes Ergebnis | 0 |
| Tatsächliches Ergebnis | 0 |

# **Aufgabe c)**

## Zusammenfassung

Die Regular Expressions sollen nun auch um Klammern ergänzt werden **[]** um. Die Klammern können enthalten dabei eine Reihe von Zeichen, die an der Stelle s[i] vorkommen dürfen.

Sobald hier an der Stelle s[i] eines der Zeichen gefunden wurde muss berechnet werden um wie viele Stellen das Pattern nach vorne gerückt werden muss, da dieses durch die Angabe der Zeichen in **[]** länger geworden ist.

## Lösungsidee

Es wird eine zusätzliche Funktion *stringMatchesRegEx* eingebaut die s[i] erhält und das Pattern sowie die Startposition der Regular Expression im Pattern erhält.

Die Funktion splittet dann das Pattern auf den Inhalt der Expression auf und vergleicht ob an der Stelle s[i] einer der Werte vorkommt.

Hierfür wird eine Funktion *Split* eingebaut, die einen String von einem Start-Index zu einem End-Index aufsplittet (um Klammern wegzuschneiden).

Der Vergleich selbst wird wie folgt durchgeführt:

**FUNCTION** stringMatchesRegEx**(** s **:** CHAR**;** p **:** **STRING;** start **:** INTEGER **)** **:** BOOLEAN**;**

...

**WHILE** **(**i **<=** Length**(**modPattern**))** **AND** **(**matchResult **=** FALSE**)** **DO** **BEGIN**

**IF** **(**s **=** modPattern**[**i**])** **THEN**

matchResult **:=** TRUE**;**

Inc**(**i**);**

**END;**//while

...

**END;**//stringMatchesRegEx

## Quellcode

**FUNCTION** Split**(**text**:** **STRING;** startIdx**,** endIdx**:** INTEGER**):** **STRING;**

**VAR**

s**:** **STRING;**

i**:** INTEGER**;**

**BEGIN**

s **:=** ''**;**

// Validate range

**IF** **(**startIdx **<=** endIdx**)** **AND** **(**startIdx **>=** 1**)** **AND** **(**endIdx **<=** Length**(**text**))** **THEN**

**BEGIN**

**FOR** i **:=** startIdx **TO** endIdx **DO** **BEGIN**

s **:=** s **+** text**[**i**];**

**END;**//for

**END;**//if

Split **:=** s**;**

**END;**//Split

## Testfälle

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *Zeichenkette* | *vonPos* | | *bisPos* | *erwartet* | *tatsächlich* |  |
| String | 2 | | 4 | tri | tri | ✓ |
| HelloWorld | 1 | | 6 | HelloW | HelloW | ✓ |
| Pascal | 2 | | 5 | asca | asca | ✓ |
| String | | Hagenberg | | | | | | ✓ |
| Pattern | | b[eu]rg | | | | | |
| Erwartetes Ergebnis | | 6 | | | | | |
| Tatsächliches Ergebnis | | 6 | | | | | |

|  |  |  |
| --- | --- | --- |
| String | Hagenburg | ✓ |
| Pattern | b[eu]rg |
| Erwartetes Ergebnis | 6 |
| Tatsächliches Ergebnis | 6 |

|  |  |  |
| --- | --- | --- |
| String | Hagenborg | ✓ |
| Pattern | b[aeiu]rg |
| Erwartetes Ergebnis | 0 |
| Tatsächliches Ergebnis | 0 |

|  |  |  |
| --- | --- | --- |
| String | Hagenbeurg | ✓ |
| Pattern | b[eu]rg |
| Erwartetes Ergebnis | 0 |
| Tatsächliches Ergebnis | 0 |

# **Aufgabe d)**

## Zusammenfassung

Ergänzend zum Angeben von Zeichenketten in **[]** sollen jetzt auch ganze Bereiche angegeben werden können, zum Beispiel **[a-z]** oder **[A-Z0-9]** um die Suche mit Regular Expressions einfacher durchführen zu können.

## Lösungsidee

Wenn in Klammern **[]** ein Bereich vorkommt, so ist dieser (nach dem Entfernen der Klammern) immer gleich aufgebaut:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pattern | [ | a | - | z | ] |
| Position |  | 1 | 2 | 3 |  |

Wenn sich also in den Klammern an der Position 2 das Zeichen **–** befindet so definieren die Stellen 1 und 3 den Wertebereich.

Um eine einfache Lösung zu implementieren kann hier das Zeichen aus dem String s[i] in einen Ordinalwert umgewandelt werden. Danach lässt sich prüfen ob dieser zwischen den Ordinalwerten des Wertebereichs a-z liegt und gegebenenfalls ein Match erzeugen.

Ebenso wie in der vorherigen Lösung muss das Pattern dann um die komplette Länge der Regular Expression nach vorne geschoben werden!

## Quellcode

**FUNCTION** stringMatchesRegEx**(** s **:** CHAR**;** p **:** **STRING;** start **:** INTEGER **)** **:** BOOLEAN**;**

**...**

**IF** **(**

**(**Ord**(**s**)** **>=** Ord**(**modPattern**[**1**]))** **AND**

**(**Ord**(**s**)** **<=** Ord**(**modPattern**[**3**]))**

**)** **THEN**

matchResult **:=** TRUE**;**

**...**

**END;**//stringMatchesRegEx

## Testfälle

|  |  |  |
| --- | --- | --- |
| String | Hagenberg | ✓ |
| Pattern | genbe[a-z]g |
| Erwartetes Ergebnis | 3 |
| Tatsächliches Ergebnis | 3 |

|  |  |  |
| --- | --- | --- |
| String | Hegenborg7 | ✓ |
| Pattern | borg[0-9] |
| Erwartetes Ergebnis | 6 |
| Tatsächliches Ergebnis | 6 |

|  |  |  |
| --- | --- | --- |
| String | Hagenbeurg | ✓ |
| Pattern | [A-Z]agen[c-f] |
| Erwartetes Ergebnis | 0 |
| Tatsächliches Ergebnis | 0 |

# **Aufgabe e)**

## Zusammenfassung

Die bereits in den vorher implementieren Regular Expressions sollen jetzt noch die Möglichkeit einer Negation enthalten und mit dem Zeichen **^** dargestellt sein.

## Lösungsidee

Die Implementierung der Lösung aus d) muss hier lediglich ergänzt werden, um festzustellen ob es sich um eine Negation handelt.

Folgende Einschränkungen gibt es hierbei:

* **^** darf nicht am Ende des Patterns vorkommen
* nach dem **^** darf eine **[** vorkommen, diese muss aber auch wieder geschlossen werden
* kommt nach dem **^** keine Klammer, ist das darauf folgende Zeichen zu negieren

Dazu muss der Pattern ^[…] beim Splitten um eine Position weiter nach hinten wandern um den Inhalt des Patterns zu ermitteln.

## Quellcode

**FUNCTION** stringMatchesRegEx**(** s **:** CHAR**;** p **:** **STRING;** start **:** INTEGER **)** **:** BOOLEAN**;**

…

**IF** negation **THEN**

modPattern **:=** split**(**p**,** 3**,** BruteForceA**(**p**,** ']'**)-**1**)**

**ELSE**

modPattern **:=** split**(**p**,** 2**,** BruteForceA**(**p**,** ']'**)-**1**);**

…

**IF** negation **THEN**

matchResult **:=** **NOT** matchResult**;**

**END;**//stringMatchesRegEx

## Testfälle

|  |  |  |
| --- | --- | --- |
| String | Hagenbergburg | ✓ |
| Pattern | b^erg |
| Erwartetes Ergebnis | 10 |
| Tatsächliches Ergebnis | 10 |

|  |  |  |
| --- | --- | --- |
| String | Hagenborg7 | ✓ |
| Pattern | borg[0-9] |
| Erwartetes Ergebnis | 6 |
| Tatsächliches Ergebnis | 6 |

|  |  |  |
| --- | --- | --- |
| String | Hagenbeurg | ✓ |
| Pattern | [A-Z]agen[c-f] |
| Erwartetes Ergebnis | 0 |
| Tatsächliches Ergebnis | 0 |

|  |  |  |
| --- | --- | --- |
| String | Hagenberg | ✓ |
| Pattern | ^Iagenb^[a-d]rg |
| Erwartetes Ergebnis | 0 |
| Tatsächliches Ergebnis | 0 |

# Erweiterung

## Lösungsidee

Um zu verhindern dass ein ungültiger regulärer Ausdruck eingegeben wird, wurde eine Validierung eingebaut um die Korrektheit zu prüfen.

Die Regular Expression wird als ungültig betrachtet, wenn:

* **^** Negation am Ende des Patterns
* **^.** auf die Negation ein beliebiges Zeichen folgt
* **]** auf eine Klammer **[** keine **]** folgt
* **[b-a]** der Wertebereich nicht korrekt ist

## Quellcode

**FUNCTION** checkRegex **(** p **:** **STRING** **)** **:** BOOLEAN**;**

**VAR**

i**,**pos **:** INTEGER**;**

valid **:** BOOLEAN**;**

spl **:** **STRING;**

**BEGIN**

valid **:=** TRUE**;**

spl **:=** ''**;**

i **:=** 0**;**

pos **:=** 0**;**

**IF** **(**p**[**Length**(**p**)]** **=** '^'**)** **THEN**

valid **:=** FALSE

**ELSE** **BEGIN**

**WHILE** **(**valid**)** **AND** **(**i **<=** Length**(**p**))** **DO** **BEGIN**

**IF** **(**p**[**i**]** **=** '^'**)** **AND** **(**p**[**i**+**1**]** **=** '.'**)** **THEN**

valid **:=** FALSE

**ELSE** **IF** **(**p**[**i**]** **=** '['**)** **THEN** **BEGIN**

spl **:=** Split**(**p**,** i**+**1**,** Length**(**p**));**

pos **:=** BruteForceA**(**spl**,** ']'**);**

**IF** pos **=** 0 **THEN** **BEGIN**

valid **:=** FALSE**;** // no closing bracket

**END** **ELSE** **BEGIN**

spl **:=** Split**(**spl**,** 1**,** pos**-**1**);**

**IF** **(** **(**Length**(**spl**)** **=** 0**)** **OR**

**(**BruteForceA**(**spl**,** '^'**)** **<>** 0 **)** **OR**

**(** BruteForceA**(**spl**,** '.'**)** **<>** 0 **)** **OR**

**(** // inner range check

**(**Length**(**spl**)** **=** 3**)** **AND**

**(**spl**[**2**]** **=** '-'**)** **AND**

**(** Ord**(**spl**[**1**])** **>** Ord**(**spl**[**3**])** **)**

**)**

**)** **THEN**

valid **:=** FALSE**;**

i **:=** i **+** pos**;** // pos : closing bracket

**END;**//if

**END** **ELSE** **IF** **(**p**[**i**]** **=** ']'**)** **THEN**

valid **:=** FALSE**;**

Inc**(**i**);**

**END;**//while

**END;**//if

checkRegex **:=** valid**;**

**END;**//checkRegex

# Quellcode Gesamtaufgabe

**PROGRAM** Regex**;**

**USES** Formatting**;**

(\* --------------------------------------------------------------------\*)

**FUNCTION** BruteForceA **(** s**,** p **:** **STRING)** **:** INTEGER**;**

**VAR**

sLen**,** pLen**,** i**,** j**,** pos **:** INTEGER**;**

**BEGIN**

sLen **:=** Length**(**s**);**

pLen **:=** Length**(**p**);**

pos **:=** 0**;**

i **:=** 1**;** // erste Stelle im String

// solange nicht gefunden -> pos = 0

// und noch nicht das Ende erreicht

**WHILE** **(**pos **=** 0**)** **AND** **(**i **<=** sLen **-** pLen **+** 1**)** **DO** **BEGIN**

j **:=** 1**;**

**WHILE** **(**j **<=** pLen**)** **AND** **(**s**[**i **+** j **-** 1**]** **=** p**[**j**])** **DO**

Inc**(**j**);**

**IF** j **>** pLen **THEN**

pos **:=** i

**ELSE**

Inc**(**i**);**

**END;**//while

BruteForceA **:=** pos**;**

**END;**//BruteForceA

(\* --------------------------------------------------------------------\*)

**FUNCTION** Split**(**text**:** **STRING;** startIdx**,** endIdx**:** INTEGER**):** **STRING;**

**VAR**

s**:** **STRING;**

i**:** INTEGER**;**

**BEGIN**

s **:=** ''**;**

// Validate range

**IF** **(**startIdx **<=** endIdx**)** **AND** **(**startIdx **>=** 1**)** **AND** **(**endIdx **<=** Length**(**text**))** **THEN** **BEGIN**

**FOR** i **:=** startIdx **TO** endIdx **DO** **BEGIN**

s **:=** s **+** text**[**i**];**

**END;**//for

**END;**//if

Split **:=** s**;**

**END;**//Split

(\* --------------------------------------------------------------------\*)

**FUNCTION** checkRegex **(** p **:** **STRING** **)** **:** BOOLEAN**;**

**VAR**

i**,**pos **:** INTEGER**;**

valid **:** BOOLEAN**;**

spl **:** **STRING;**

**BEGIN**

valid **:=** TRUE**;**

spl **:=** ''**;**

i **:=** 0**;**

pos **:=** 0**;**

**IF** **(**p**[**Length**(**p**)]** **=** '^'**)** **THEN**

valid **:=** FALSE

**ELSE** **BEGIN**

**WHILE** **(**valid**)** **AND** **(**i **<=** Length**(**p**))** **DO** **BEGIN**

**IF** **(**p**[**i**]** **=** '^'**)** **AND** **(**p**[**i**+**1**]** **=** '.'**)** **THEN**

valid **:=** FALSE

**ELSE** **IF** **(**p**[**i**]** **=** '['**)** **THEN** **BEGIN**

spl **:=** Split**(**p**,** i**+**1**,** Length**(**p**));**

pos **:=** BruteForceA**(**spl**,** ']'**);**

**IF** pos **=** 0 **THEN** **BEGIN**

valid **:=** FALSE**;** // no closing bracket

**END** **ELSE** **BEGIN**

spl **:=** Split**(**spl**,** 1**,** pos**-**1**);**

**IF** **(** **(**Length**(**spl**)** **=** 0**)** **OR**

**(**BruteForceA**(**spl**,** '^'**)** **<>** 0 **)** **OR**

**(** BruteForceA**(**spl**,** '.'**)** **<>** 0 **)** **OR**

**(** // inner range check

**(**Length**(**spl**)** **=** 3**)** **AND**

**(**spl**[**2**]** **=** '-'**)** **AND**

**(** Ord**(**spl**[**1**])** **>** Ord**(**spl**[**3**])** **)**

**)**

**)** **THEN**

valid **:=** FALSE**;**

i **:=** i **+** pos**;** // pos : closing bracket

**END;**//if

**END** **ELSE** **IF** **(**p**[**i**]** **=** ']'**)** **THEN**

valid **:=** FALSE**;**

Inc**(**i**);**

**END;**//while

**END;**//if

checkRegex **:=** valid**;**

**END;**//checkRegex

(\* --------------------------------------------------------------------\*)

**FUNCTION** getPatternLength **(** p **:** **STRING;** start **:** INTEGER **)** **:** INTEGER**;**

**VAR**

i**,** spos**,** epos **:** INTEGER**;**

**BEGIN**

spos **:=** 0**;**

epos **:=** 0**;**

**FOR** i **:=** start **TO** Length**(**p**)** **DO** **BEGIN**

**IF** **(**p**[**i**]** **=** '^'**)** **THEN**

spos **:=** i

**ELSE** **IF** **(**p**[**i**]** **=** '['**)** **THEN**

spos **:=** i

**ELSE** **IF** **(**p**[**i**]** **=** ']'**)** **THEN**

epos **:=** i**;**

**END;**//for

getPatternLength **:=** epos **-** spos **-**1**;**

**END;**//getPatternLength

(\* --------------------------------------------------------------------\*)

**FUNCTION** stringMatchesRegEx**(** s **:** CHAR**;** p **:** **STRING;** start **:** INTEGER **)** **:** BOOLEAN**;**

**VAR**

i **:** INTEGER**;**

modPattern **:** **STRING;**

matchResult**,** negation **:** BOOLEAN**;**

**BEGIN**

i **:=** 0**;**

matchResult **:=** FALSE**;**

negation **:=** p**[**1**]** **=** '^'**;**

**IF** negation **THEN**

modPattern **:=** split**(**p**,** 3**,** BruteForceA**(**p**,** ']'**)-**1**)**

**ELSE**

modPattern **:=** split**(**p**,** 2**,** BruteForceA**(**p**,** ']'**)-**1**);**

**IF** **((**Length**(**modPattern**)** **=** 3**)** **AND**

**(**modPattern**[**2**]** **=** '-'**))** **THEN** **BEGIN**

**IF** **(**

**(**Ord**(**s**)** **>=** Ord**(**modPattern**[**1**]))** **AND**

**(**Ord**(**s**)** **<=** Ord**(**modPattern**[**3**]))**

**)** **THEN**

matchResult **:=** TRUE**;**

**END** **ELSE** **BEGIN**

**WHILE** **(**i **<=** Length**(**modPattern**))** **AND** **(**matchResult **=** FALSE**)** **DO** **BEGIN**

**IF** **(**s **=** modPattern**[**i**])** **THEN**

matchResult **:=** TRUE**;**

Inc**(**i**);**

**END;**//while

**END;** //if

**IF** negation **THEN**

matchResult **:=** **NOT** matchResult**;**

stringMatchesRegEx **:=** matchResult**;**

**END;**//stringMatchesRegEx

(\* --------------------------------------------------------------------\*)

**FUNCTION** BruteForceRegEx **(** s**,** p **:** **STRING)** **:** INTEGER**;**

**VAR**

sLen**,** pLen**,** i**,** j**,** index**,** pos **:** INTEGER**;**

spl **:** **STRING;**

valid **:** BOOLEAN**;**

**BEGIN**

sLen **:=** Length**(**s**);**

pLen **:=** Length**(**p**);**

pos **:=** 0**;**

spl **:=** ''**;**

i **:=** 1**;** // erste Stelle im String

// begin only if regex is valid

**IF** **(**checkRegex**(**p**))** **THEN** **BEGIN**

**WHILE** **(**pos **=** 0**)** **AND** **(**i **<=** sLen**)** **DO** **BEGIN**

valid **:=** TRUE**;**

j **:=** 1**;**

index **:=** 0**;**

**WHILE** **(** **(**j **<=** pLen**)** **AND** **(**valid **=** TRUE**)** **AND** **(**index **<=** sLen**)** **)** **DO** **BEGIN**

**IF** **(** **(**p**[**j**]** **=** '.' **)** **OR** **(**p**[**j**]** **=** s**[**i **+** index**])** **)** **THEN**

Inc**(**j**)**

**ELSE** **IF** **(**

**(**p**[**j**]** **=** '^'**)** **AND**

**(**p**[**j**+**1**]** **=** '['**)** **AND**

**(**stringMatchesRegEx**(**s**[**i **+** index**],** Split**(**p**,** j**,** pLen**),** j**))**

**)** **THEN** **BEGIN**

spl **:=** Split**(**p**,** j**+**2**,** pLen**);**

j **:=** j **+** BruteForceA**(**spl**,** ']'**)** **+** 2**;**

**END** **ELSE** **IF** **(**

**(**p**[**j**]** **=** '['**)** **AND**

**(**stringMatchesRegEx**(**s**[**i **+** index **],**

Split**(**p**,** j**,** pLen**),** j**))**

**)** **THEN** **BEGIN**

spl **:=** Split**(**p**,** j**+**1**,** pLen**);**

j **:=** j **+** BruteForceA**(**spl**,** ']'**)** **+** 1**;**

**END** **ELSE** **IF** **(**

**(**p**[**j**]** **=** '^'**)** **AND**

**(**p**[**j**+**1**]** **<>** '['**)** **AND**

**(**p**[**j**+**1**]** **<>** s**[**i **+** index**])**

**)** **THEN**

j **:=** j **+** 2

**ELSE**

valid **:=** FALSE**;**

index **:=** index **+** 1**;**

**END;**//while

**IF** **(** **(**valid **=** TRUE**)** **AND** **(**j **>** pLen**)** **)** **THEN**

pos **:=** i**;**

Inc**(**i**);**

**END;**//while

**END;**//if

BruteForceRegEx **:=** pos**;**

**END;**//BruteForceRegEx

(\* --------------------------------------------------------------------\*)

**VAR**

pos **:** INTEGER**;**

**BEGIN**

printTitle**(**'Mini Regex'**);**

// PART 1

printHeader**(**'BruteForce (1)'**);**

pos **:=** BruteForceA**(**'Hagenbrg'**,** 'brg'**);**

Write**(**'Hagenbrg | brg: '**);** WriteLn**(**pos**);**

pos **:=** BruteForceA**(**'Hagenberg'**,** 'berg'**);**

Write**(**'Hagenberg | berg: '**);** WriteLn**(**pos**);**

pos **:=** BruteForceA**(**'Hagenburg'**,** 'berg'**);**

Write**(**'Hagenburg | berg: '**);** WriteLn**(**pos**);**

WriteLn**;**

// PART 2

printHeader**(**'BruteForce (2)'**);**

pos **:=** BruteForceRegEx**(**'Hagenbrg'**,** 'b.rg'**);**

WriteLn**(**'Hagenbrg | b.rg: '**,** pos**);**

pos **:=** BruteForceRegEx**(**'Hagenberg'**,** 'b.rg'**);**

WriteLn**(**'Hagenberg | b.rg: '**,** pos**);**

pos **:=** BruteForceRegEx**(**'Hagenburg'**,** 'b.rg'**);**

WriteLn**(**'Hagenburg | b.rg: '**,** pos**);**

pos **:=** BruteForceRegEx**(**'Hagenberg'**,** 'b..rg'**);**

WriteLn**(**'Hagenberg | b..rg: '**,** pos**);**

WriteLn**;**

// Split

printHeader**(**'Split'**);**

WriteLn**(**'String [2-4] = '**,** Split**(**'String'**,** 2**,** 4**));**

WriteLn**(**'HelloWorld [1-6] = '**,** Split**(**'HelloWorld'**,** 1**,** 6**));**

WriteLn**(**'Pascal [2-5] = '**,** Split**(**'Pascal'**,** 2**,** 5**));**

WriteLn**;**

// PART 3

printHeader**(**'BruteForce RegEx (3)'**);**

pos **:=** BruteForceRegEx**(**'Hagenberg'**,** 'b[eu]rg'**);**

WriteLn**(**'Hagenberg | b[eu]rg: '**,** pos**);**

pos **:=** BruteForceRegEx**(**'Hagenburg'**,** 'b[eu]rg'**);**

WriteLn**(**'Hagenburg | b[eu]rg: '**,** pos**);**

pos **:=** BruteForceRegEx**(**'Hagenborg'**,** 'b[aeiu]rg'**);**

WriteLn**(**'Hagenborg | b[aeiu]rg: '**,** pos**);**

pos **:=** BruteForceRegEx**(**'Hagenbeurg'**,** 'b[eu]rg'**);**

WriteLn**(**'Hagenbeurg | b[eu]rg: '**,** pos**);**

BruteForceRegEx**(**'Hagenberg'**,** 'b^[a-b]r[f-g]'**);**

Write**(**'Hagenberg | b^[a-b]r[f-g]: '**);** WriteLn**(**pos**);**

WriteLn**;**

// PART 4

printHeader**(**'BruteForce RegEx (4)'**);**

pos **:=** BruteForceRegEx**(**'Hagenberg'**,** 'genbe[a-z]g'**);**

WriteLn**(**'Hagenberg | genbe[a-z]g: '**,** pos**);**

pos **:=** BruteForceRegEx**(**'Hagenborg7'**,** 'borg[0-9]'**);**

WriteLn**(**'Hagenborg7 | borg[0-9]: '**,** pos**);**

pos **:=** BruteForceRegEx**(**'Hagenbeurg'**,** '[A-Z]agen[c-f]'**);**

WriteLn**(**'Hagenbeurg | [A-Z]agen[c-f]: '**,** pos**);**

WriteLn**;**

// PART 5

printHeader**(**'BruteForce (5)'**);**

pos **:=** BruteForceRegEx**(**'Hagenbergburg'**,** 'b^erg'**);**

WriteLn**(**'Hagenbergburg | b^erg: '**,** pos**);**

pos **:=** BruteForceRegEx**(**'Hagenborg7'**,** 'borg[0-9]'**);**

WriteLn**(**'Hagenborg7 | borg[0-9]: '**,** pos**);**

pos **:=** BruteForceRegEx**(**'Hagenbeurg'**,** '[A-Z]agen[c-f]'**);**

WriteLn**(**'Hagenbeurg | [A-Z]agen[c-f]: '**,** pos**);**

pos **:=** BruteForceRegEx**(**'Hagenberg'**,** '^Iagenb^[a-d]rg'**);**

WriteLn**(**'Hagenberg | ^Iagenb^[a-d]rg: '**,** pos**);**

WriteLn**;**

**END.**

(\* --------------------------------------------------------------------\*)

**UNIT** Formatting**;**

**INTERFACE**

**PROCEDURE** printTitle**(** s **:** **STRING** **);**

**PROCEDURE** printHeader**(** s **:** **STRING** **);**

**IMPLEMENTATION**

**USES** Crt**;**

**PROCEDURE** printTitle**(** s **:** **STRING** **);**

**VAR**

i **:** INTEGER**;**

**BEGIN**

TextBackground**(**Blue**);**

TextColor**(**LightGreen**);**

WriteLn**(**s**);**

**FOR** i **:=** 1 **TO** Length**(**s**)** **DO**

Write**(**'='**);**

WriteLn**();**

NormVideo**();**

WriteLn**();**

**END;**

**PROCEDURE** printHeader**(** s **:** **STRING** **);**

**BEGIN**

TextColor**(**Blue**);**

WriteLn**(**s**);**

NormVideo**();**

**END;**

**PROCEDURE** printHeader**(** s **:** **STRING;** c **:** BYTE**)** **overload;**

**BEGIN**

TextColor**(**c**);**

WriteLn**(**s**);**

NormVideo**();**

**END;**

**BEGIN**

**END.**