[1 Lösungsidee 2](#_Toc384060058)

[2 Source 4](#_Toc384060059)

[2.1 RegexUtils 4](#_Toc384060060)

[2.2 RegexUtilsTest 12](#_Toc384060061)

[3 Tests 22](#_Toc384060062)

[3.1 IsValidString 22](#_Toc384060063)

[3.2 IsValidPattern 22](#_Toc384060064)

[3.3 CreateIgnoreRegex 23](#_Toc384060065)

[3.4 CreateNgationRegex 23](#_Toc384060066)

[3.5 CreateCharacterSetRegex 24](#_Toc384060067)

[3.6 CreateCharacterRangeRegex 24](#_Toc384060068)

[3.7 CreateNegationCharacterSetRegex 25](#_Toc384060069)

[3.8 CreateNegationCharacterRangeRegex 25](#_Toc384060070)

[3.9 Matches 26](#_Toc384060071)

# Lösungsidee

Folgend ist die Lösungsidee für die Übung 2 angeführt, wobei eine simple Funktionalität einer Regex zu implementieren ist.

Hierbei sollen die unterstützten Regex Zeichen und Patterns vordefiniert und als Konstanten gehalten werden. Dem Aufrufer sollen Funktionen und Prozeduren zur Verfügung gestellt werden, die gültige Regular Expressions für die unterstützten Regex Patterns erstellen, wobei der Aufrufer hier keine bereits erstellte Regular Expression als Zeichenkette übergeben darf, sondern lediglich die Zeichen die innerhalb der erstellten Regex verwendet werden sollen.

Es obliegt dem Aufrufer die erstellten Regular Expressions zu einem größeren Pattern, der mehrere Regular Expression beinhaltet zusammenzufügen. Damit soll gewährleitstet werden, dass keine ungültigen Regular Expressions produziert werden, und die Syntax vorm Aufrufer abstrahiert ist.

Des Weiteren soll gewährleistet werden, dass dem Algorithmus keine nicht handhabbaren Pattern übergeben werden, da ein Aufrufer sich die Zeichenkette auch selber zusammenstellen könnte, was aber vermieden werden sollte. Daher soll der übergebene Pattern auf Gültigkeit geprüft werden, bevor dieser angewendet wird.

Für die Funktion Matches soll ein BruteForce Algorithmus verwendet werden, der die verschiedenen Regular Expressions handhaben kann. Da die tatsächliche Pattern Länge nicht mehr feststellbar ist, soll dieser Algorithmus bis zum Ende der zu durchsuchenden Zeichenkette laufen. Es könnte auch anders implementiert werden, indem die Regular Expression entfernt werden und sich ein Pattern aufgebaut wird, der der tatsächlichen Pattern Länge entspricht, aber in dieser Implementierung soll darauf verzichtet werden. Es dürfen keine vordefinierten Regex Zeichen als Zeichen innerhalb des Patterns verwendet werden, sie sind also für die Regular Expressions reserviert. Alle anderen Zeichen auch Sonderzeichen sind als Zeichen erlaubt auch innerhalb einer Range Regex, da hier lediglich die Ordinal Werte herangezogen werden. Es soll unterstützt werden, dass sich mehrere Regular Expressions innerhalb eines erstellten Pattern befinden.

Bsp.:

1. Hagen.b^[a-e].g
2. .ag.n^er^[ab].^g
3. …

Die erstellten Patterns sollen folgender Syntax folgen.

**Ignore Regex**

Diese Regex ist an folgenden Positionen nicht erlaubt.

1. Hagen[.abcde]
2. Hagen[.-b]
3. Hagen[a-.]
4. Hagen^.erg

**Negation Regex**

Diese Regex ist an folgenden Positionen nicht erlaubt.

1. Hagenberg^
2. Hagen[^abc]
3. Hagen[^-c]
4. Hagen[a-^]
5. Hage^.berg

**CharacterSet Regex**

Folgende Syntax dieser Regex ist nicht erlaubt.

1. Hagenbe[]
2. Hage]acd[

**CharacterRange Regex**

Folgende Syntax dieser Regex ist nicht erlaubt.

1. Hagenb[aa-v]
2. Hagenb[a-vvv]
3. Hagenb[v-a]
4. Hagenb]a-v[

# Source

Folgend ist der Source der implementierten RegexUtils und dessen Tests angeführt. Die verwendeten implementierten Util Untis wurden hier nicht explizit angeführt. Es sei hierbei auf den Source verwiesen.

## RegexUtils

Folgend ist der Source der RegexUtils Unit angeführt.

**Unit** RegexUtils**;**

// ############################################ Interface part ############################################

**Interface**

**Uses**

IntegerUtils**,** StringUtils**;**

**Type**

{ The predefined error codes }

Error **=** **(**NONE**,** INVALID\_CHARACTER**,** INVALID\_COUNT**,** INVALID\_PATTERN**);**

{

Creates a regex for irgnore. E.g.: 'AB...CD'

@param

prefix: the prefix which prefixes the regex

@param

postfix: the postfix postfixes the regex

@param

count: the count of regex to be added

@return

the created regex

@return

result: the result of this procedure invocation

INVALID\_CHARACTER: If the prefix and/or postfix contains predefined regex characters

INVALID\_COUNT: If the count < 0

}

**Function** CreateIgnoreRegex**(**prefix**,** postfix**:** **String;** count**:** Integer**;** **Var** result**:** Error**):** **String;**

{

Creates a regex for a character set. E.g.: 'AB[abcde]CD'

@param

prefix: the prefix which prefixes the regex

@param

postfix: the postfix postfixes the regex

@param

characters: the character t be contained in the character set

@return

the created regex

@return

result: the result of this procedure invocation

INVALID\_CHARACTER: If the prefix and/or postfix contains predefined regex characters, or the given characters are empty

INVALID\_COUNT: If the count < 0

}

**Function** CreateCharacterSetRegex**(**prefix**,** characters**,** postfix**:** **String;** **Var** result**:** Error**):** **String;**

{

Creates a regex for a character range. E.g.: 'AB[a-b]CD'

@param

prefix: the prefix which prefixes the regex

@param

postfix: the postfix postfixes the regex

@param

lowerRange: the char representing the lower range

@param

higherRange: the char representing the higher range

@return

the created regex

@return

result: the result of this procedure invocation

INVALID\_CHARACTER: If the prefix and/or postfix contains predefined regex characters,

or the given characters ordinal values overflow each other

or the given character represent predfined regex characters

INVALID\_COUNT: If the count < 0

}

**Function** CreateCharacterRangeRegex**(**prefix**,** postfix**:** **String;** lowerRange**,** higherRange**:** Char**;** **Var** result**:** Error**):** **String;**

{

Creates a regex for negation. E.g.: 'AB^CD'

@param

prefix: the prefix which prefixes the regex

@param

postfix: the postfix postfixes the regex

@return

the created regex

@return

result: the result of this procedure invocation

INVALID\_CHARACTER: If the prefix and/or postfix contains predefined regex characters,

or if postfix is an empty string

INVALID\_COUNT: If the count < 0

}

**Function** CreateNegationRegex**(**prefix**,** postfix**:** **String;** **Var** result**:** Error**):** **String;**

{

Creates a regex for a negation character set. E.g.: 'AB^[abcde]CD'

@param

prefix: the prefix which prefixes the regex

@param

postfix: the postfix postfixes the regex

@param

characters: the character t be contained in the character set

@return

the created regex

@return

result: the result of this procedure invocation

INVALID\_CHARACTER: If the prefix and/or postfix contains predefined regex characters, or the given characters are empty

INVALID\_COUNT: If the count < 0

}

**Function** CreateNegationCharacterSetRegex**(**prefix**,** characters**,** postfix**:** **String;** **Var** result**:** Error**):** **String;**

{

Creates a regex for a negation character range. E.g.: 'AB^[a-b]CD'

@param

prefix: the prefix which prefixes the regex

@param

postfix: the postfix postfixes the regex

@param

lowerRange: the char representing the lower range

@param

higherRange: the char representing the higher range

@return

the created regex

@return

result: the result of this procedure invocation

INVALID\_CHARACTER: If the prefix and/or postfix contains predefined regex characters,

or the given characters ordinal values overflow each other

or the given character represent predfined regex characters

INVALID\_COUNT: If the count < 0

}

**Function** CreateNegationCharacterRangeRegex**(**prefix**,** postfix**:** **String;** lowerRange**,** higherRange**:** Char**;** **Var** result**:** Error**):** **String;**

{

Answers the question if the given pattern is contained in the given text,

where the pattern is allowed to contain regular expressions.

@param

text: the text to searched

@param

pattern: the pattern ot be searched in the text

@return

the position of the contained text, 0 if not found or if invalid pattern.

}

**Function** Matches**(**text**,** pattern**:** **String):** Integer**;**

// ############################################ Interface part ############################################

// ############################################ Just for testing ############################################

**Function** IsValidString**(**text**:** **String):** Boolean**;**

**Function** IsValidPattern**(**pattern**:** **String):** Boolean**;**

// ############################################ Just for testing ############################################

// ############################################ Implementation part ############################################

**Implementation**

**Type**

{ The predefined regex types }

RegexType **=** **(**IGNORE**,** CHARACTER\_SET**,** CHARACTER\_RANGE**,** NEGATION**);**

**Const**

{ the predefined regex characters }

supportedRegex**:** **Array** **[**RegexType**]** **of** **String** **=** **(**

**(**'.'**),**

**(**'[]'**),**

**(**'-'**),**

**(**'^'**)**

**);**

// ############################################ Private part ############################################

{

Validates if the given text contains any predefined regex characters which is not allowed.

@param

text: the text which gets validated

@return

true if the text is valid false otherwise

}

**Function** IsValidString**(**text**:** **String):** Boolean**;**

**Var**

i**,** j**,** pLength**,** rLength**:** Integer**;**

rType**:** RegexType**;**

valid**:** Boolean**;**

regex**:** **String;**

**Begin**

i **:=** 1**;**

valid **:=** true**;**

pLength **:=** Length**(**text**);**

// Iterator over all characters of the text

**while** **(**i **<=** pLength**)** **and** **(**valid**)** **do** **begin**

**for** rType **in** RegexType **do** **begin**

regex **:=** supportedRegex**[**rType**];**

rLength **:=** Length**(**regex**);**

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

// Iterate over all regex predefined characters

**while** **(**j **<=** rLength**)** **and** **(**valid**)** **do** **begin**

valid **:=** text**[**i**]** **<>** regex**[**j**];**

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

**end;**

// Break if valid, used because index overflow not possible with enumeration

**if** **(not** valid**)** **then** **begin**

Break**;**

**end;**

**end;**

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

**end;**

IsValidString **:=** valid**;**

**End;**

{

Creates a single valued regex for the given regex type.

@param

prefix: the prefix which prefixes the regex

@param

prefix: the postfix which postfixes the regex

@param

count: the count how often the regex shall be added

@return

the created regex string

@return

result: the result of this procedure invocation

INVALID\_CHARACTER: If the prefix and/or postfix contains regex characters

INVALID\_COUNT: If the count <= 0

}

**Function** CreateSingleValueRegex**(**prefix**,** postfix**:** **String;** count**:** Integer**;** rType**:** RegexType**;** **Var** result**:** Error**):** **String;**

**Var**

regex**:** **String;**

i**:** Integer**;**

**Begin**

CreateSingleValueRegex **:=** ''**;**

result **:=** Error**.**NONE**;**

// Validate if pattern contains already regex characters

**if** **(NOT** IsValidString**(**prefix**))** **or** **(NOT** IsValidString**(**postfix**))** **then** **begin**

result **:=** Error**.**INVALID\_CHARACTER**;**

**end**

// Validate given count

**else** **if** **(**count **<=** 0**)** **then** **begin**

result **:=** Error**.**INVALID\_COUNT**;**

**end**

// Otherwise create regex string

**else** **begin**

regex **:=** prefix**;**

**for** i **:=** 1 **to** count **do** **begin**

regex **:=** regex **+** supportedRegex**[**rType**];**

**end;**

CreateSingleValueRegex **:=** regex **+** postfix**;**

**end;**

**End;**

{

Validates the given pattern if it contains a valid regex which can be handled by the matches function.

@param

pattern: the pattern which shall be validated

@return

true if the pattern is valid, false otherwise

}

**Function** IsValidPattern**(**pattern**:** **String):** Boolean**;**

**Var**

pos**,** pos1**,** pos2**:** Integer**;**

split**,** tempPattern**:** **String;**

valid**:** Boolean**;**

**Begin**

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

pos1 **:=** 0**;**

pos2 **:=** 0**;**

valid **:=** true**;**

tempPattern **:=** pattern**;**

// Validate contained multiple regex

pos1 **:=** StringUtils**.**PatternPosition**(**tempPattern**,** supportedRegex**[**CHARACTER\_SET**][**1**]);**

pos2 **:=** StringUtils**.**PatternPosition**(**tempPattern**,** supportedRegex**[**CHARACTER\_SET**][**2**]);**

// Validate if ready to check for valid multiple regex

**while** **(**valid**)** **and** **((**pos1 **<>** 0**)** **or** **(**pos2 **<>** 0**))** **do** **begin**

{ writeln('tempPattern: ':15, tempPattern); }

**if** **(**pos1 **<>** 0**)** **and** **(**pos2 **<>** 0**)** **and** **(**pos2 **>** **(**pos1 **+** 1**))** **then** **begin**

split **:=** StringUtils**.**Split**(**tempPattern**,** pos1 **+** 1**,** pos2 **-** 1**);**

{ writeln('split string: ':15, split);}

**if** **(**StringUtils**.**Contains**(**split**,** supportedRegex**[**IGNORE**],** false**))** // multiple select contains single regex

**or** **(**StringUtils**.**Contains**(**split**,** supportedRegex**[**NEGATION**],** false**))** // no negation within braces

**or** **(**StringUtils**.**Contains**(**split**,** supportedRegex**[**CHARACTER\_SET**][**1**],** false**))** // no additional left brace embedded

**or** **(**StringUtils**.**Contains**(**split**,** supportedRegex**[**CHARACTER\_SET**][**2**],** false**))** **then** **begin** // no additional right brace embedded

valid **:=** false**;**

**end**

// Check for range regex

**else** **if** **(**valid**)** **then** **begin**

pos **:=** StringUtils**.**PatternPosition**(**split**,** supportedRegex**[**CHARACTER\_RANGE**]);**

**if** **(**pos **<>** 0**)** **then** **begin**

**if** **(**Length**(**split**)** **<>** 3**)** **or** **(**split**[**2**]** **<>** supportedRegex**[**CHARACTER\_RANGE**])**

**or** **(not** IntegerUtils**.**IsValidRange**(**Ord**(**split**[**1**]),** Ord**(**split**[**3**])))**

**or** **(**split**[**1**]** **=** supportedRegex**[**CHARACTER\_RANGE**])**

**or** **(**split**[**3**]** **=** supportedRegex**[**CHARACTER\_RANGE**])** **then** **begin**

valid **:=** false**;**

**end;**

**end;**

tempPattern **:=** StringUtils**.**Split**(**tempPattern**,** pos2 **+** 1**,** Length**(**tempPattern**));**

pos1 **:=** StringUtils**.**PatternPosition**(**tempPattern**,** supportedRegex**[**CHARACTER\_SET**][**1**]);**

pos2 **:=** StringUtils**.**PatternPosition**(**tempPattern**,** supportedRegex**[**CHARACTER\_SET**][**2**]);**

**end;**

**end**

// Invalid multiple regex

**else** **begin**

valid **:=** false**;**

**end;**

tempPattern **:=** StringUtils**.**Split**(**tempPattern**,** pos2 **+** 1**,** Length**(**tempPattern**));**

pos1 **:=** StringUtils**.**PatternPosition**(**tempPattern**,** supportedRegex**[**CHARACTER\_SET**][**1**]);**

pos2 **:=** StringUtils**.**PatternPosition**(**tempPattern**,** supportedRegex**[**CHARACTER\_SET**][**2**]);**

**end;**

// Validate single regex which can not be invalid positioned in multiple regex at this point

tempPattern **:=** pattern**;**

pos **:=** StringUtils**.**PatternPosition**(**tempPattern**,** supportedRegex**[**IGNORE**]);**

**while** **(**valid**)** **and** **(**pos **<>** 0**)** **do** **begin**

**if** **(**pos **>** 1**)** **and** **(**tempPattern**[**pos **-** 1**]** **=** supportedRegex**[**NEGATION**])** **then** **begin**

valid **:=** false**;**

**end;**

tempPattern **:=** StringUtils**.**Split**(**tempPattern**,** pos **+** 1**,** Length**(**tempPattern**));**

pos **:=** StringUtils**.**PatternPosition**(**tempPattern**,** supportedRegex**[**IGNORE**]);**

**end;**

// Validate negation regex if it stands alone

tempPattern **:=** pattern**;**

pos **:=** StringUtils**.**PatternPosition**(**tempPattern**,** supportedRegex**[**NEGATION**]);**

**while** **(**valid**)** **and** **(**pos **<>** 0**)** **do** **begin**

{ writeln('split: ', tempPattern); }

**if** **(**pos **=** Length**(**tempPattern**))**

**or** **(**pos **<** Length**(**tempPattern**))** **and** **(**tempPattern**[**pos **+** 1**]** **=** supportedRegex**[**NEGATION**])** **then** **begin**

valid **:=** false**;**

**end;**

tempPattern **:=** StringUtils**.**Split**(**tempPattern**,** pos **+** 1**,** Length**(**tempPattern**));**

pos **:=** StringUtils**.**PatternPosition**(**tempPattern**,** supportedRegex**[**NEGATION**]);**

**end;**

IsValidPattern **:=** valid**;**

**End;**

// ############################################ Private part ############################################

// ############################################ Public part ############################################

{ CreateIgnoreRegex }

**Function** CreateIgnoreRegex**(**prefix**,** postfix**:** **String;** count**:** Integer**;** **Var** result**:** Error**):** **String;**

**Begin**

CreateIgnoreRegex **:=** CreateSingleValueRegex**(**prefix**,** postfix**,** count**,** IGNORE**,** result**);**

**End;**

{ CreateCharacterSetRegex }

**Function** CreateCharacterSetRegex**(**prefix**,** characters**,** postfix**:** **String;** **Var** result**:** Error**):** **String;**

**Begin**

CreateCharacterSetRegex **:=** ''**;**

result **:=** Error**.**NONE**;**

// Validate if pattern contains already regex characters

**if** **(NOT** IsValidString**(**prefix**))** **or** **(NOT** IsValidString**(**postfix**)** **or** **(NOT** IsValidString**(**characters**))** **OR** **(**Length**(**characters**)** **=** 0**))** **then** **begin**

result **:=** Error**.**INVALID\_CHARACTER**;**

**end**

// Otherwise create regex string

**else** **begin**

CreateCharacterSetRegex **:=** prefix **+** supportedRegex**[**CHARACTER\_SET**][**1**]** **+** characters **+** supportedRegex**[**CHARACTER\_SET**][**2**]** **+** postfix**;**

**end;**

**End;**

{ CreateCharacterRangeRegex }

**Function** CreateCharacterRangeRegex**(**prefix**,** postfix**:** **String;** lowerRange**,** higherRange**:** Char**;** **Var** result**:** Error**):** **String;**

**Begin**

CreateCharacterRangeRegex **:=** ''**;**

result **:=** Error**.**NONE**;**

// Validate if pattern contains already regex characters

**if** **(NOT** IsValidString**(**prefix**))** **or** **(NOT** IsValidString**(**postfix**))** **or** **(NOT** IsValidString**(**lowerRange **+** higherRange**))** **or** **(**Ord**(**lowerRange**)** **>** Ord**(**higherRange**))** **then** **begin**

result **:=** Error**.**INVALID\_CHARACTER**;**

**end**

// Otherwise create regex string

**else** **begin**

CreateCharacterRangeRegex **:=** prefix **+** supportedRegex**[**CHARACTER\_SET**][**1**]** **+** lowerRange **+** supportedRegex**[**CHARACTER\_RANGE**]** **+** higherRange **+** supportedRegex**[**CHARACTER\_SET**][**2**]** **+** postfix**;**

**end;**

**End;**

{ CreateNegationRegex }

**Function** CreateNegationRegex**(**prefix**,** postfix**:** **String;** **Var** result**:** Error**):** **String;**

**Begin**

CreateNegationRegex **:=** ''**;**

result **:=** Error**.**NONE**;**

// Validate if pattern contains already regex characters

**if** **(NOT** IsValidString**(**prefix**))** **or** **(NOT** IsValidString**(**postfix**))** **or** **(**Length**(**postfix**)** **=** 0**)** **then** **begin**

result **:=** Error**.**INVALID\_CHARACTER**;**

**end**

// Otherwise create regex string

**else** **begin**

CreateNegationRegex **:=** CreateSingleValueRegex**(**prefix**,** postfix**,** 1**,** NEGATION**,** result**);**

**end;**

**End;**

{ CreateNegationCharacterSetRegex }

**Function** CreateNegationCharacterSetRegex**(**prefix**,** characters**,** postfix**:** **String;** **Var** result**:** Error**):** **String;**

**Begin**

CreateNegationCharacterSetRegex **:=** ''**;**

result **:=** Error**.**NONE**;**

// Validate if pattern contains already regex characters

**if** **(NOT** IsValidString**(**prefix**))** **or** **(NOT** IsValidString**(**postfix**)** **or** **(NOT** IsValidString**(**characters**))** **OR** **(**Length**(**characters**)** **=** 0**))** **then** **begin**

result **:=** Error**.**INVALID\_CHARACTER**;**

**end**

// Otherwise create regex string

**else** **begin**

CreateNegationCharacterSetRegex **:=** prefix **+** supportedRegex**[**NEGATION**]** **+** supportedRegex**[**CHARACTER\_SET**][**1**]** **+** characters **+** supportedRegex**[**CHARACTER\_SET**][**2**]** **+** postfix**;**

**end;**

**End;**

{ CreateNegationCharacterRangeRegex }

**Function** CreateNegationCharacterRangeRegex**(**prefix**,** postfix**:** **String;** lowerRange**,** higherRange**:** Char**;** **Var** result**:** Error**):** **String;**

**Begin**

CreateNegationCharacterRangeRegex **:=** ''**;**

result **:=** Error**.**NONE**;**

// Validate if pattern contains already regex characters

**if** **(NOT** IsValidString**(**prefix**))** **or** **(NOT** IsValidString**(**postfix**))** **or** **(**Ord**(**lowerRange**)** **>** Ord**(**higherRange**))** **then** **begin**

result **:=** Error**.**INVALID\_CHARACTER**;**

**end**

// Otherwise create regex string

**else** **begin**

CreateNegationCharacterRangeRegex **:=** prefix **+** supportedRegex**[**NEGATION**]** **+** supportedRegex**[**CHARACTER\_SET**][**1**]** **+** lowerRange **+** supportedRegex**[**CHARACTER\_RANGE**]** **+** higherRange **+** supportedRegex**[**CHARACTER\_SET**][**2**]** **+** postfix**;**

**end;**

**End;**

{ Matches }

**Function** Matches**(**text**,** pattern**:** **String):** Integer**;**

**Var**

textLength**,** patternLength**,** partLength**,** i **,**j**,** k**,** idx**,** pos**:**Integer**;**

textPart**,** part**,** patternBuf**:** **String;**

negate**,** run**:** Boolean**;**

**Begin**

Matches **:=** 0**;**

**if** **(**IsValidPattern**(**pattern**))** **then** **begin**

textLength **:=** Length**(**text**);**

patternLength **:=** Length**(**pattern**);**

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

i **:=** 1**;**

// Iterate over string to last possible position

**while** **(**pos **=** 0**)** **and** **(**i **<=** textLength**)** **do** **begin**

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

idx **:=** 1**;**

run **:=** true**;**

{\* writeln; \*}

{\* writeln('j: ', j, ' - i: ', i); \*}

// Iterate over pattern an check for match

**while** **(**run**)** **and** **(**j **<=** patternLength**)** **and** **((**i **+** idx **-** 1**)** **<=** textLength**)** **do** **begin**

textPart **:=** text**[**i **+** idx **-** 1**];**

negate **:=** false**;**

// Ignore when single regex

**if** **(**pattern**[**j**]** **=** supportedRegex**[**IGNORE**])** **then** **begin**

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

**end**

**else** **begin**

// Negation

**if** **(**pattern**[**j**]** **=** supportedRegex**[**NEGATION**])** **then** **begin**

negate **:=** true**;**

{\* writeln('negate':15); \*}

**if** **(**pattern**[**j **+** 1**]** **=** supportedRegex**[**CHARACTER\_SET**][**1**])** **then** **begin**

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

{\* writeln('increased because of negated range'); \*}

**end;**

**end;**

// Check if multiple regex

**if** **(**pattern**[**j**]** **=** supportedRegex**[**CHARACTER\_SET**][**1**])** **then** **begin**

// Need to cut the already handled parts because StringUtils.PatternPosition will always return the first occurance

// which would be the wrong one if contained multiple times

patternBuf **:=** StringUtils**.**split**(**pattern**,** j**,** Length**(**pattern**));**

part **:=** StringUtils**.**split**(**pattern**,** j **+** 1**,** StringUtils**.**PatternPosition**(**patternBuf**,** supportedRegex**[**CHARACTER\_SET**][**2**])** **+** j **-** 2**);**

partLength **:=** Length**(**part**);**

// Range

**if** **(**part**[**2**]** **=** supportedRegex**[**CHARACTER\_RANGE**])** **then** **begin**

**if** **((not** negate**)** **and** **(not** StringUtils**.**IsWithinOrdinalRange**(**part**[**1**],** part**[**3**],** textPart**[**1**])))**

**or** **((**negate**)** **and** **(**StringUtils**.**IsWithinOrdinalRange**(**part**[**1**],** part**[**3**],** textPart**[**1**])))** **then** **begin**

run **:=** false**;**

{\* writeln('negate: ':15, negate);

writeln('text char: ':15, textPart[1]);

writeln('pattern char: ':15, part[1]);

writeln('pattern char: ':15, part[3]); \*}

**end;**

**end**

// No Range

**else** **begin**

writeln**(**'text char: '**:**15**,** textPart**);**

k **:=** 1**;**

**while** **(**k **<=** partLength**)** **and** **(**part**[**k**]** **<>** textPart**)** **do** **begin**

{\* writeln('part char: ':15, textPart); \*}

Inc**(**k**);**

**end;**

**if** **((not** negate**)** **and** **(**k **>** partLength**))**

**or** **((**negate**)** **and** **(**k **<=** partLength**))** **then** **begin**

run **:=** false**;**

**end;**

**end;**

**if** **(**run**)** **then** **begin**

j **:=** j **+** partLength **+** Length**(**supportedRegex**[**CHARACTER\_SET**]);**

**end;**

**end**

**else** **begin**

// Compare character under consideration of negation

run **:=** **((**negate**)** **and** **(**pattern**[**j **+** 1**]** **<>** textPart**));**

**if** **(**run**)** **then** **begin**

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

**end**

**else** **begin**

run **:=** **((not** negate**)** **and** **(**pattern**[**j**]** **=** textPart**));**

**if** **(**run**)** **then** **begin**

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

**end;**

**end;**

**end;**

**end;**

Inc**(**idx**);**

// Cut out the already handled pattern parts

**end;**

// run still set and j overflows patternLength then we found the pattern

**if** **(**run**)** **and** **(**j **>** patternLength**)** **then** **begin**

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

{\* writeln('found when j: ', j); \*}

**end**

// No need to increase i when match has been found

**else** **begin**

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

**end;**

**end;**

Matches **:=** pos**;**

**end;**

**End;**

// ############################################ Public part ############################################

// ############################################ Implementation part ############################################

**Begin**

**End.**

## RegexUtilsTest

Folgend ist der Source der RegexUtilsTest angeführt, welche die RegexUtils Unit testet.

**Program** RegexUtilsTest**;**

**Uses**

RegexUtils**,** PrintUtils**,** Crt**;**

**Const**

headerLength **=** 40**;**

{

Tests the internal validation method which validates if a given string

which has benn passed to a regex generating method is a valid one,

means that this string does not contain any regex specifc characters.

}

**Procedure** TestIsValidString**;**

**Var**

v**:** **String;**

**Begin**

PrintUtils**.**PrintHeader**(**'Test IsValidString (Private)'**,** headerLength**);**

v **:=** 'AB.CD'**;**

writeln**(**v**,** ' -> '**:**15**,** RegexUtils**.**IsValidString**(**v**));**

v **:=** 'AB^CD'**;**

writeln**(**v**,** ' -> '**:**15**,** RegexUtils**.**IsValidString**(**v**));**

v **:=** 'AB[CD'**;**

writeln**(**v**,** ' -> '**:**15**,** RegexUtils**.**IsValidString**(**v**));**

v **:=** 'AB]CD'**;**

writeln**(**v**,** ' -> '**:**15**,** RegexUtils**.**IsValidString**(**v**));**

v **:=** 'AB-CD'**;**

writeln**(**v**,** ' -> '**:**15**,** RegexUtils**.**IsValidString**(**v**));**

v **:=** 'ABCD'**;**

writeln**(**v**,** ' -> '**:**15**,** RegexUtils**.**IsValidString**(**v**));**

**End;**

{

Tests the CreateIgnoreRegex prozedure which creates a pattern which cotnains

regex to be able to ignore characters on specific positions in the pattern

}

**Procedure** TestCreateIgnoreRegex**;**

**Var**

pre**,** pos**,** regex**:** **String;**

result**:** Error**;**

**Begin**

// invalid character on prefix

PrintUtils**.**PrintHeader**(**'invalid character on prefix'**,** headerLength**);**

pre **:=** 'PR.E'**;**

pos **:=** 'SUF'**;**

writeln**(**'Prefix: '**:**20**,** pre**);**

writeln**(**'Suffix: '**:**20**,** pos**);**

regex **:=** RegexUtils**.**CreateIgnoreRegex**(**pre**,** pos**,** 0**,** result**);**

writeln**(**'generated regex -> '**:**20**,** regex**);**

writeln**(**'procedure result -> '**:**20**,** result**);**

// invalid character on suffix

PrintUtils**.**PrintHeader**(**'invalid character on suffix'**,** headerLength**);**

pre **:=** 'PRE'**;**

pos **:=** 'PO.S'**;**

writeln**(**'Prefix: '**:**20**,** pre**);**

writeln**(**'Suffix: '**:**20**,** pos**);**

regex **:=** RegexUtils**.**CreateIgnoreRegex**(**pre**,** pos**,** 0**,** result**);**

writeln**(**'generated regex -> '**:**20**,** regex**);**

writeln**(**'procedure result -> '**:**20**,** result**);**

// Invalid count

PrintUtils**.**PrintHeader**(**'Invalid count'**,** headerLength**);**

pre **:=** 'PRE'**;**

pos **:=** 'SUF'**;**

writeln**(**'Prefix: '**:**20**,** pre**);**

writeln**(**'Suffix: '**:**20**,** pos**);**

regex **:=** RegexUtils**.**CreateIgnoreRegex**(**pre**,** pos**,** 0**,** result**);**

writeln**(**'generated regex -> '**:**20**,** regex**);**

writeln**(**'procedure result -> '**:**20**,** result**);**

// 1 time

PrintUtils**.**PrintHeader**(**'1 time'**,** headerLength**);**

pre **:=** 'PRE'**;**

pos **:=** 'SUF'**;**

writeln**(**'Prefix: '**:**20**,** pre**);**

writeln**(**'Suffix: '**:**20**,** pos**);**

regex **:=** RegexUtils**.**CreateIgnoreRegex**(**pre**,** pos**,** 1**,** result**);**

writeln**(**'generated regex -> '**:**20**,** regex**);**

writeln**(**'procedure result -> '**:**20**,** result**);**

// 5 times

PrintUtils**.**PrintHeader**(**'5 times'**,** headerLength**);**

pre **:=** 'PRE'**;**

pos **:=** 'SUF'**;**

writeln**(**'Prefix: '**:**20**,** pre**);**

writeln**(**'Suffix: '**:**20**,** pos**);**

regex **:=** RegexUtils**.**CreateIgnoreRegex**(**pre**,** pos**,** 5**,** result**);**

writeln**(**'generated regex -> '**:**20**,** regex**);**

writeln**(**'procedure result -> '**:**20**,** result**);**

// 1 time no prefix

PrintUtils**.**PrintHeader**(**'1 times no prefix'**,** headerLength**);**

pre **:=** 'PRE'**;**

pos **:=** 'SUF'**;**

writeln**(**'Prefix: '**:**20**,** pre**);**

writeln**(**'Suffix: '**:**20**,** pos**);**

regex **:=** RegexUtils**.**CreateIgnoreRegex**(**''**,** pos**,** 1**,** result**);**

writeln**(**'generated regex -> '**:**20**,** regex**);**

writeln**(**'procedure result -> '**:**20**,** result**);**

// 1 time no suffix

PrintUtils**.**PrintHeader**(**'1 times no suffix'**,** headerLength**);**

pre **:=** 'PRE'**;**

pos **:=** 'SUF'**;**

writeln**(**'Prefix: '**:**20**,** pre**);**

writeln**(**'Suffix: '**:**20**,** pos**);**

regex **:=** RegexUtils**.**CreateIgnoreRegex**(**pre**,** ''**,** 1**,** result**);**

writeln**(**'generated regex -> '**:**20**,** regex**);**

writeln**(**'procedure result -> '**:**20**,** result**);**

// 1 time no pre and suffix

PrintUtils**.**PrintHeader**(**'1 times no pre- and suffix'**,** headerLength**);**

pre **:=** 'PRE'**;**

pos **:=** 'SUF'**;**

writeln**(**'Prefix: '**:**20**,** pre**);**

writeln**(**'Suffix: '**:**20**,** pos**);**

regex **:=** RegexUtils**.**CreateIgnoreRegex**(**''**,** ''**,** 1**,** result**);**

writeln**(**'generated regex -> '**:**20**,** regex**);**

writeln**(**'procedure result -> '**:**20**,** result**);**

**End;**

{

Tests the procedure CreateNegationRegex which creates a pattern which cotnains

´a regex which allows to negate the following character.

}

**Procedure** TestCreateNegationRegex**;**

**Var**

pre**,** pos**,** regex**:** **String;**

result**:** Error**;**

**Begin**

// Invalid character on prefix

PrintUtils**.**PrintHeader**(**'invalid character on prefix'**,** headerLength**);**

pre **:=** 'PR.E'**;**

pos **:=** 'SUF'**;**

writeln**(**'Prefix: '**:**20**,** pre**);**

writeln**(**'Suffix: '**:**20**,** pos**);**

regex **:=** RegexUtils**.**CreateNegationRegex**(**pre**,** pos**,** result**);**

writeln**(**'generated regex -> '**:**20**,** regex**);**

writeln**(**'procedure result -> '**:**20**,** result**);**

// Invalid character on suffix

PrintUtils**.**PrintHeader**(**'invalid character on suffix'**,** headerLength**);**

pre **:=** 'PRE'**;**

pos **:=** 'PO.S'**;**

writeln**(**'Prefix: '**:**20**,** pre**);**

writeln**(**'Suffix: '**:**20**,** pos**);**

regex **:=** RegexUtils**.**CreateNegationRegex**(**pre**,** pos**,** result**);**

writeln**(**'generated regex -> '**:**20**,** regex**);**

writeln**(**'procedure result -> '**:**20**,** result**);**

// Invalid no suffix

pre **:=** 'PRE'**;**

pos **:=** ''**;**

PrintUtils**.**PrintHeader**(**'Invalid no suffix'**,** headerLength**);**

writeln**(**'Prefix: '**:**20**,** pre**);**

writeln**(**'Suffix: '**:**20**,** pos**);**

regex **:=** RegexUtils**.**CreateNegationRegex**(**pre**,** ''**,** result**);**

writeln**(**'generated regex -> '**:**20**,** regex**);**

writeln**(**'procedure result -> '**:**20**,** result**);**

// Valid perfix and suffix

pre **:=** 'PRE'**;**

pos **:=** 'SUF'**;**

PrintUtils**.**PrintHeader**(**'Valid prefix and suffix'**,** headerLength**);**

writeln**(**'Prefix: '**:**20**,** pre**);**

writeln**(**'Suffix: '**:**20**,** pos**);**

regex **:=** RegexUtils**.**CreateNegationRegex**(**pre**,** pos**,** result**);**

writeln**(**'generated regex -> '**:**20**,** regex**);**

writeln**(**'procedure result -> '**:**20**,** result**);**

// No prefix

pre **:=** ''**;**

pos **:=** 'SUF'**;**

PrintUtils**.**PrintHeader**(**'Valid no prefix'**,** headerLength**);**

writeln**(**'Prefix: '**:**20**,** pre**);**

writeln**(**'Suffix: '**:**20**,** pos**);**

regex **:=** RegexUtils**.**CreateNegationRegex**(**pre**,** pos**,** result**);**

writeln**(**'generated regex -> '**:**20**,** regex**);**

writeln**(**'procedure result -> '**:**20**,** result**);**

**End;**

{

Tests teh procedure CreateCharacterSetRegex which creates a pattern which allows to specify

a set of characters which are allowed on the specified position.

}

**Procedure** TestCreateCharacterSetRegex**;**

**Var**

pre**,** pos**,** characters**,** regex**:** **String;**

result**:** Error**;**

**Begin**

// invalid character on prefix

PrintUtils**.**PrintHeader**(**'Invalid character on prefix'**,** headerLength**);**

pre **:=** 'PR.E'**;**

pos **:=** 'SUF'**;**

characters **:=** 'absce'**;**

writeln**(**'Prefix: '**:**20**,** pre**);**

writeln**(**'Suffix: '**:**20**,** pos**);**

writeln**(**'characters: '**:**20**,** characters**);**

regex **:=** RegexUtils**.**CreateCharacterSetRegex**(**pre**,** characters**,** pos**,** result**);**

writeln**(**'generated regex -> '**:**20**,** regex**);**

writeln**(**'procedure result -> '**:**20**,** result**);**

// invalid character on suffix

PrintUtils**.**PrintHeader**(**'Invalid character on suffix'**,** headerLength**);**

pre **:=** 'PRE'**;**

pos **:=** 'SU.F '**;**

characters **:=** 'absce'**;**

writeln**(**'Prefix: '**:**20**,** pre**);**

writeln**(**'Suffix: '**:**20**,** pos**);**

writeln**(**'characters: '**:**20**,** characters**);**

regex **:=** RegexUtils**.**CreateCharacterSetRegex**(**pre**,** characters**,** pos**,** result**);**

writeln**(**'generated regex -> '**:**20**,** regex**);**

writeln**(**'procedure result -> '**:**20**,** result**);**

// invalid character on suffix

PrintUtils**.**PrintHeader**(**'Invalid character on characters'**,** headerLength**);**

pre **:=** 'PRE'**;**

pos **:=** 'SUF'**;**

characters **:=** 'abs.ce'**;**

writeln**(**'Prefix: '**:**20**,** pre**);**

writeln**(**'Suffix: '**:**20**,** pos**);**

writeln**(**'characters: '**:**20**,** characters**);**

regex **:=** RegexUtils**.**CreateCharacterSetRegex**(**pre**,** characters**,** pos**,** result**);**

writeln**(**'generated regex -> '**:**20**,** regex**);**

writeln**(**'procedure result -> '**:**20**,** result**);**

// Empty characters

PrintUtils**.**PrintHeader**(**'Empty characters'**,** headerLength**);**

pre **:=** 'PRE'**;**

pos **:=** 'SUF'**;**

characters **:=** ''**;**

writeln**(**'Prefix: '**:**20**,** pre**);**

writeln**(**'Suffix: '**:**20**,** pos**);**

writeln**(**'characters: '**:**20**,** characters**);**

regex **:=** RegexUtils**.**CreateCharacterSetRegex**(**pre**,** characters**,** pos**,** result**);**

writeln**(**'generated regex -> '**:**20**,** regex**);**

writeln**(**'procedure result -> '**:**20**,** result**);**

// Valid

PrintUtils**.**PrintHeader**(**'Valid'**,** headerLength**);**

pre **:=** 'PRE'**;**

pos **:=** 'SUF'**;**

characters **:=** 'absce'**;**

writeln**(**'Prefix: '**:**20**,** pre**);**

writeln**(**'Suffix: '**:**20**,** pos**);**

writeln**(**'characters: '**:**20**,** characters**);**

regex **:=** RegexUtils**.**CreateCharacterSetRegex**(**pre**,** characters**,** pos**,** result**);**

writeln**(**'generated regex -> '**:**20**,** regex**);**

writeln**(**'procedure result -> '**:**20**,** result**);**

**End;**

{

Tests the procedure CreateCharacterSetRangeRegex which creates a regex

which specifies a ordinal range the following character must fit.

}

**Procedure** TestCreateCharacterRangeRegex**;**

**Var**

pre**,** pos**,** regex**:** **String;**

lower**,** higher**:** Char**;**

result**:** Error**;**

**Begin**

// Invalid character on prefix

PrintUtils**.**PrintHeader**(**'Invalid character on prefix'**,** headerLength**);**

pre **:=** 'PR.E'**;**

pos **:=** 'SUF'**;**

lower **:=** 'a'**;**

higher **:=** 'c'**;**

writeln**(**'Prefix: '**:**20**,** pre**);**

writeln**(**'Suffix: '**:**20**,** pos**);**

writeln**(**'lower: '**:**20**,** lower**);**

writeln**(**'higher: '**:**20**,** higher**);**

regex **:=** RegexUtils**.**CreateCharacterRangeRegex**(**pre**,** pos**,** lower**,** higher**,** result**);**

writeln**(**'generated regex -> '**:**20**,** regex**);**

writeln**(**'procedure result -> '**:**20**,** result**);**

// Invalid character on suffix

PrintUtils**.**PrintHeader**(**'Invalid character on suffix'**,** headerLength**);**

pre **:=** 'PRE'**;**

pos **:=** 'PO.S'**;**

lower **:=** 'a'**;**

higher **:=** 'c'**;**

writeln**(**'Prefix: '**:**20**,** pre**);**

writeln**(**'Suffix: '**:**20**,** pos**);**

writeln**(**'lower: '**:**20**,** lower**);**

writeln**(**'higher: '**:**20**,** higher**);**

regex **:=** RegexUtils**.**CreateCharacterRangeRegex**(**pre**,** pos**,** lower**,** higher**,** result**);**

writeln**(**'generated regex -> '**:**20**,** regex**);**

writeln**(**'procedure result -> '**:**20**,** result**);**

// Invalid character on range left

PrintUtils**.**PrintHeader**(**'Invalid character on range left'**,** headerLength**);**

pre **:=** 'PRE'**;**

pos **:=** 'POS'**;**

lower **:=** '.'**;**

higher **:=** 'c'**;**

writeln**(**'Prefix: '**:**20**,** pre**);**

writeln**(**'Suffix: '**:**20**,** pos**);**

writeln**(**'lower: '**:**20**,** lower**);**

writeln**(**'higher: '**:**20**,** higher**);**

regex **:=** RegexUtils**.**CreateCharacterRangeRegex**(**pre**,** pos**,** lower**,** higher**,** result**);**

writeln**(**'generated regex -> '**:**20**,** regex**);**

writeln**(**'procedure result -> '**:**20**,** result**);**

// Invalid character on range right

PrintUtils**.**PrintHeader**(**'Invalid character on range right'**,** headerLength**);**

pre **:=** 'PRE'**;**

pos **:=** 'POS'**;**

lower **:=** 'a'**;**

higher **:=** '.'**;**

writeln**(**'Prefix: '**:**20**,** pre**);**

writeln**(**'Suffix: '**:**20**,** pos**);**

writeln**(**'lower: '**:**20**,** lower**);**

writeln**(**'higher: '**:**20**,** higher**);**

regex **:=** RegexUtils**.**CreateCharacterRangeRegex**(**pre**,** pos**,** lower**,** higher**,** result**);**

writeln**(**'generated regex -> '**:**20**,** regex**);**

writeln**(**'procedure result -> '**:**20**,** result**);**

// Invalid ordinal order

PrintUtils**.**PrintHeader**(**'Invalid range ordinal'**,** headerLength**);**

pre **:=** 'PRE'**;**

pos **:=** 'SUF'**;**

lower **:=** 'c'**;**

higher **:=** 'a'**;**

writeln**(**'Prefix: '**:**20**,** pre**);**

writeln**(**'Suffix: '**:**20**,** pos**);**

writeln**(**'lower: '**:**20**,** lower**);**

writeln**(**'higher: '**:**20**,** higher**);**

regex **:=** RegexUtils**.**CreateCharacterRangeRegex**(**pre**,** pos**,** lower**,** higher**,** result**);**

writeln**(**'generated regex -> '**:**20**,** regex**);**

writeln**(**'procedure result -> '**:**20**,** result**);**

// Valid

PrintUtils**.**PrintHeader**(**'Valid'**,** headerLength**);**

pre **:=** 'PRE'**;**

pos **:=** 'SUF'**;**

lower **:=** 'a'**;**

higher **:=** 'c'**;**

writeln**(**'Prefix: '**:**20**,** pre**);**

writeln**(**'Suffix: '**:**20**,** pos**);**

writeln**(**'lower: '**:**20**,** lower**);**

writeln**(**'higher: '**:**20**,** higher**);**

regex **:=** RegexUtils**.**CreateCharacterRangeRegex**(**pre**,** pos**,** lower**,** higher**,** result**);**

writeln**(**'generated regex -> '**:**20**,** regex**);**

writeln**(**'procedure result -> '**:**20**,** result**);**

**End;**

{

Tests teh procedure CreateCharacterSetRegex which creates a pattern which allows to specify

a set of characters which are not allowed on the specified position.

}

**Procedure** TestCreateNegationCharacterSetRegex**;**

**Var**

pre**,** pos**,** characters**,** regex**:** **String;**

result**:** Error**;**

**Begin**

// invalid character on prefix

PrintUtils**.**PrintHeader**(**'Invalid character on prefix'**,** headerLength**);**

pre **:=** 'PR.E'**;**

pos **:=** 'SUF'**;**

characters **:=** 'absce'**;**

writeln**(**'Prefix: '**:**20**,** pre**);**

writeln**(**'Suffix: '**:**20**,** pos**);**

writeln**(**'characters: '**:**20**,** characters**);**

regex **:=** RegexUtils**.**CreateNegationCharacterSetRegex**(**pre**,** characters**,** pos**,** result**);**

writeln**(**'generated regex -> '**:**20**,** regex**);**

writeln**(**'procedure result -> '**:**20**,** result**);**

// invalid character on suffix

PrintUtils**.**PrintHeader**(**'Invalid character on suffix'**,** headerLength**);**

pre **:=** 'PRE'**;**

pos **:=** 'SU.F '**;**

characters **:=** 'absce'**;**

writeln**(**'Prefix: '**:**20**,** pre**);**

writeln**(**'Suffix: '**:**20**,** pos**);**

writeln**(**'characters: '**:**20**,** characters**);**

regex **:=** RegexUtils**.**CreateNegationCharacterSetRegex**(**pre**,** characters**,** pos**,** result**);**

writeln**(**'generated regex -> '**:**20**,** regex**);**

writeln**(**'procedure result -> '**:**20**,** result**);**

// invalid character on suffix

PrintUtils**.**PrintHeader**(**'Invalid character on characters'**,** headerLength**);**

pre **:=** 'PRE'**;**

pos **:=** 'SUF'**;**

characters **:=** 'abs.ce'**;**

writeln**(**'Prefix: '**:**20**,** pre**);**

writeln**(**'Suffix: '**:**20**,** pos**);**

writeln**(**'characters: '**:**20**,** characters**);**

regex **:=** RegexUtils**.**CreateNegationCharacterSetRegex**(**pre**,** characters**,** pos**,** result**);**

writeln**(**'generated regex -> '**:**20**,** regex**);**

writeln**(**'procedure result -> '**:**20**,** result**);**

// Empty characters

PrintUtils**.**PrintHeader**(**'Empty characters'**,** headerLength**);**

pre **:=** 'PRE'**;**

pos **:=** 'SUF'**;**

characters **:=** ''**;**

writeln**(**'Prefix: '**:**20**,** pre**);**

writeln**(**'Suffix: '**:**20**,** pos**);**

writeln**(**'characters: '**:**20**,** characters**);**

regex **:=** RegexUtils**.**CreateNegationCharacterSetRegex**(**pre**,** characters**,** pos**,** result**);**

writeln**(**'generated regex -> '**:**20**,** regex**);**

writeln**(**'procedure result -> '**:**20**,** result**);**

// Valid

PrintUtils**.**PrintHeader**(**'Valid'**,** headerLength**);**

pre **:=** 'PRE'**;**

pos **:=** 'SUF'**;**

characters **:=** 'absce'**;**

writeln**(**'Prefix: '**:**20**,** pre**);**

writeln**(**'Suffix: '**:**20**,** pos**);**

writeln**(**'characters: '**:**20**,** characters**);**

regex **:=** RegexUtils**.**CreateNegationCharacterSetRegex**(**pre**,** characters**,** pos**,** result**);**

writeln**(**'generated regex -> '**:**20**,** regex**);**

writeln**(**'procedure result -> '**:**20**,** result**);**

**End;**

{

Tests the procedure CreateCharacterSetRangeRegex which creates a regex

which specifies a ordinal range the following character must not fit.

}

**Procedure** TestCreateNegationCharacterRangeRegex**;**

**Var**

pre**,** pos**,** regex**:** **String;**

lower**,** higher**:** Char**;**

result**:** Error**;**

**Begin**

// Invalid character on prefix

PrintUtils**.**PrintHeader**(**'Invalid character on prefix'**,** headerLength**);**

pre **:=** 'PR.E'**;**

pos **:=** 'SUF'**;**

lower **:=** 'a'**;**

higher **:=** 'c'**;**

writeln**(**'Prefix: '**:**20**,** pre**);**

writeln**(**'Suffix: '**:**20**,** pos**);**

writeln**(**'lower: '**:**20**,** lower**);**

writeln**(**'higher: '**:**20**,** higher**);**

regex **:=** RegexUtils**.**CreateNegationCharacterRangeRegex**(**pre**,** pos**,** lower**,** higher**,** result**);**

writeln**(**'generated regex -> '**:**20**,** regex**);**

writeln**(**'procedure result -> '**:**20**,** result**);**

// Invalid character on suffix

PrintUtils**.**PrintHeader**(**'Invalid character on suffix'**,** headerLength**);**

pre **:=** 'PRE'**;**

pos **:=** 'PO.S'**;**

lower **:=** 'a'**;**

higher **:=** 'c'**;**

writeln**(**'Prefix: '**:**20**,** pre**);**

writeln**(**'Suffix: '**:**20**,** pos**);**

writeln**(**'lower: '**:**20**,** lower**);**

writeln**(**'higher: '**:**20**,** higher**);**

regex **:=** RegexUtils**.**CreateNegationCharacterRangeRegex**(**pre**,** pos**,** lower**,** higher**,** result**);**

writeln**(**'generated regex -> '**:**20**,** regex**);**

writeln**(**'procedure result -> '**:**20**,** result**);**

// Invalid character on range left

PrintUtils**.**PrintHeader**(**'Invalid character on range left'**,** headerLength**);**

pre **:=** 'PRE'**;**

pos **:=** 'POS'**;**

lower **:=** '.'**;**

higher **:=** 'c'**;**

writeln**(**'Prefix: '**:**20**,** pre**);**

writeln**(**'Suffix: '**:**20**,** pos**);**

writeln**(**'lower: '**:**20**,** lower**);**

writeln**(**'higher: '**:**20**,** higher**);**

regex **:=** RegexUtils**.**CreateNegationCharacterRangeRegex**(**pre**,** pos**,** lower**,** higher**,** result**);**

writeln**(**'generated regex -> '**:**20**,** regex**);**

writeln**(**'procedure result -> '**:**20**,** result**);**

// Invalid character on range right

PrintUtils**.**PrintHeader**(**'Invalid character on range right'**,** headerLength**);**

pre **:=** 'PRE'**;**

pos **:=** 'POS'**;**

lower **:=** 'a'**;**

higher **:=** '.'**;**

writeln**(**'Prefix: '**:**20**,** pre**);**

writeln**(**'Suffix: '**:**20**,** pos**);**

writeln**(**'lower: '**:**20**,** lower**);**

writeln**(**'higher: '**:**20**,** higher**);**

regex **:=** RegexUtils**.**CreateNegationCharacterRangeRegex**(**pre**,** pos**,** lower**,** higher**,** result**);**

writeln**(**'generated regex -> '**:**20**,** regex**);**

writeln**(**'procedure result -> '**:**20**,** result**);**

// Invalid ordinal order

PrintUtils**.**PrintHeader**(**'Invalid range ordinal'**,** headerLength**);**

pre **:=** 'PRE'**;**

pos **:=** 'SUF'**;**

lower **:=** 'c'**;**

higher **:=** 'a'**;**

writeln**(**'Prefix: '**:**20**,** pre**);**

writeln**(**'Suffix: '**:**20**,** pos**);**

writeln**(**'lower: '**:**20**,** lower**);**

writeln**(**'higher: '**:**20**,** higher**);**

regex **:=** RegexUtils**.**CreateNegationCharacterRangeRegex**(**pre**,** pos**,** lower**,** higher**,** result**);**

writeln**(**'generated regex -> '**:**20**,** regex**);**

writeln**(**'procedure result -> '**:**20**,** result**);**

// Valid

PrintUtils**.**PrintHeader**(**'Valid'**,** headerLength**);**

pre **:=** 'PRE'**;**

pos **:=** 'SUF'**;**

lower **:=** 'a'**;**

higher **:=** 'c'**;**

writeln**(**'Prefix: '**:**20**,** pre**);**

writeln**(**'Suffix: '**:**20**,** pos**);**

writeln**(**'lower: '**:**20**,** lower**);**

writeln**(**'higher: '**:**20**,** higher**);**

regex **:=** RegexUtils**.**CreateNegationCharacterRangeRegex**(**pre**,** pos**,** lower**,** higher**,** result**);**

writeln**(**'generated regex -> '**:**20**,** regex**);**

writeln**(**'procedure result -> '**:**20**,** result**);**

**End;**

{

Tests the function IsValidPattern which validates the given pattern is it

contains only valid regular expressions

}

**Procedure** TestIsValidPattern**;**

**Var**

result**:** Boolean**;**

pattern**:** **String;**

**Begin**

PrintUtils**.**PrintHeader**(**'1.'**,** headerLength**);**

pattern **:=** 'Hagenberg^'**;**

result **:=** RegexUtils**.**IsValidPattern**(**pattern**);**

writeln**(**'pattern -> '**:**20**,** pattern**);**

writeln**(**'result -> '**:**20**,** result**);**

PrintUtils**.**PrintHeader**(**'2.'**,** headerLength**);**

pattern **:=** 'Hag^.enberg'**;**

result **:=** RegexUtils**.**IsValidPattern**(**pattern**);**

writeln**(**'pattern -> '**:**20**,** pattern**);**

writeln**(**'result -> '**:**20**,** result**);**

PrintUtils**.**PrintHeader**(**'3.'**,** headerLength**);**

pattern **:=** 'Hag[abcd.]enberg'**;**

result **:=** RegexUtils**.**IsValidPattern**(**pattern**);**

writeln**(**'pattern -> '**:**20**,** pattern**);**

writeln**(**'result -> '**:**20**,** result**);**

PrintUtils**.**PrintHeader**(**'4.'**,** headerLength**);**

pattern **:=** 'Hag[aa-b]enberg'**;**

result **:=** RegexUtils**.**IsValidPattern**(**pattern**);**

writeln**(**'pattern -> '**:**20**,** pattern**);**

writeln**(**'result -> '**:**20**,** result**);**

PrintUtils**.**PrintHeader**(**'5.'**,** headerLength**);**

pattern **:=** 'Hag[a-aa]enberg'**;**

result **:=** RegexUtils**.**IsValidPattern**(**pattern**);**

writeln**(**'pattern -> '**:**20**,** pattern**);**

writeln**(**'result -> '**:**20**,** result**);**

PrintUtils**.**PrintHeader**(**'6.'**,** headerLength**);**

pattern **:=** 'Hag[a-.]enberg'**;**

result **:=** RegexUtils**.**IsValidPattern**(**pattern**);**

writeln**(**'pattern -> '**:**20**,** pattern**);**

writeln**(**'result -> '**:**20**,** result**);**

PrintUtils**.**PrintHeader**(**'7.'**,** headerLength**);**

pattern **:=** 'Hag]a-aa[enberg'**;**

result **:=** RegexUtils**.**IsValidPattern**(**pattern**);**

writeln**(**'pattern -> '**:**20**,** pattern**);**

writeln**(**'result -> '**:**20**,** result**);**

PrintUtils**.**PrintHeader**(**'8.'**,** headerLength**);**

pattern **:=** '^Ha.ga^[a-b]e.nb^e[abcde]rg'**;**

result **:=** RegexUtils**.**IsValidPattern**(**pattern**);**

writeln**(**'pattern -> '**:**20**,** pattern**);**

writeln**(**'result -> '**:**20**,** result**);**

PrintUtils**.**PrintHeader**(**'8.'**,** headerLength**);**

pattern **:=** 'Hagenberg'**;**

result **:=** RegexUtils**.**IsValidPattern**(**pattern**);**

writeln**(**'pattern -> '**:**20**,** pattern**);**

writeln**(**'result -> '**:**20**,** result**);**

**End;**

**Procedure** TestMatches**;**

**Var**

text**,** pattern**:** **String;**

pos**:** Integer**;**

result**:** Error**;**

**Begin**

text **:=** 'Ich bin ein Hagenberger'**;**

// Invalid Pattern

PrintUtils**.**PrintHeader**(**'Invalid pattern'**,** headerLength**);**

pattern **:=** 'Hagen^'**;**

pos **:=** RegexUtils**.**Matches**(**text**,** pattern**);**

writeln**(**'text: '**:**20**,** ' -> '**,** text**);**

writeln**(**'pattern: '**:**20**,** ' -> '**,** pattern**);**

writeln**(**'pos: '**:**20**,** ' -> '**,** pos**);**

// No regex invalid

PrintUtils**.**PrintHeader**(**'No regex invalid'**,** headerLength**);**

pattern **:=** 'Bagen'**;**

pos **:=** RegexUtils**.**Matches**(**text**,** pattern**);**

writeln**(**'text: '**:**20**,** ' -> '**,** text**);**

writeln**(**'pattern: '**:**20**,** ' -> '**,** pattern**);**

writeln**(**'pos: '**:**20**,** ' -> '**,** pos**);**

// Ignore regex invalid

PrintUtils**.**PrintHeader**(**'Ignore regex invalid'**,** headerLength**);**

pattern **:=** RegexUtils**.**CreateIgnoreRegex**(**'Ha'**,** 'gen'**,** 1**,** result**);**

pos **:=** RegexUtils**.**Matches**(**text**,** pattern**);**

writeln**(**'text: '**:**20**,** ' -> '**,** text**);**

writeln**(**'pattern: '**:**20**,** ' -> '**,** pattern**);**

writeln**(**'pos: '**:**20**,** ' -> '**,** pos**);**

// Negation regex invalid

PrintUtils**.**PrintHeader**(**'Negation regex invalid'**,** headerLength**);**

pattern **:=** RegexUtils**.**CreateNegationRegex**(**'Hagenb'**,** 'erger'**,** result**);**

pos **:=** RegexUtils**.**Matches**(**text**,** pattern**);**

writeln**(**'text: '**:**20**,** ' -> '**,** text**);**

writeln**(**'pattern: '**:**20**,** ' -> '**,** pattern**);**

writeln**(**'pos: '**:**20**,** ' -> '**,** pos**);**

// Character sequence invalid

PrintUtils**.**PrintHeader**(**'Character sequence invalid'**,** headerLength**);**

pattern **:=** RegexUtils**.**CreateCharacterSetRegex**(**'Hagen'**,** 'cdef'**,** 'erger'**,** result**);**

pos **:=** RegexUtils**.**Matches**(**text**,** pattern**);**

writeln**(**'text: '**:**20**,** ' -> '**,** text**);**

writeln**(**'pattern: '**:**20**,** ' -> '**,** pattern**);**

writeln**(**'pos: '**:**20**,** ' -> '**,** pos**);**

// Character range invalid

PrintUtils**.**PrintHeader**(**'Character range invalid'**,** headerLength**);**

pattern **:=** RegexUtils**.**CreateCharacterRangeRegex**(**'Hagen'**,** 'erger'**,**'c'**,** 'f'**,** result**);**

pos **:=** RegexUtils**.**Matches**(**text**,** pattern**);**

writeln**(**'text: '**:**20**,** ' -> '**,** text**);**

writeln**(**'pattern: '**:**20**,** ' -> '**,** pattern**);**

writeln**(**'pos: '**:**20**,** ' -> '**,** pos**);**

// Character negated sequence invalid

PrintUtils**.**PrintHeader**(**'Character negated sequence invalid'**,** headerLength**);**

pattern **:=** RegexUtils**.**CreateNegationCharacterSetRegex**(**'Hagen'**,** 'abe'**,** 'erger'**,** result**);**

pos **:=** RegexUtils**.**Matches**(**text**,** pattern**);**

writeln**(**'text: '**:**20**,** ' -> '**,** text**);**

writeln**(**'pattern: '**:**20**,** ' -> '**,** pattern**);**

writeln**(**'pos: '**:**20**,** ' -> '**,** pos**);**

// Character negated range invalid

PrintUtils**.**PrintHeader**(**'Character negated range invalid'**,** headerLength**);**

pattern **:=** RegexUtils**.**CreateNegationCharacterRangeRegex**(**'Hagen'**,** 'erger'**,**'a'**,** 'f'**,** result**);**

pos **:=** RegexUtils**.**Matches**(**text**,** pattern**);**

writeln**(**'text: '**:**20**,** ' -> '**,** text**);**

writeln**(**'pattern: '**:**20**,** ' -> '**,** pattern**);**

writeln**(**'pos: '**:**20**,** ' -> '**,** pos**);**

// No regex valid

PrintUtils**.**PrintHeader**(**'No regex valid'**,** headerLength**);**

pattern **:=** 'bin'**;**

pos **:=** RegexUtils**.**Matches**(**text**,** pattern**);**

writeln**(**'text: '**:**20**,** ' -> '**,** text**);**

writeln**(**'pattern: '**:**20**,** ' -> '**,** pattern**);**

writeln**(**'pos: '**:**20**,** ' -> '**,** pos**);**

// Ignore regex valid

PrintUtils**.**PrintHeader**(**'Ignore regex valid'**,** headerLength**);**

pattern **:=** RegexUtils**.**CreateIgnoreRegex**(**'Ha'**,** 'en'**,** 1**,** result**);**

pos **:=** RegexUtils**.**Matches**(**text**,** pattern**);**

writeln**(**'text: '**:**20**,** ' -> '**,** text**);**

writeln**(**'pattern: '**:**20**,** ' -> '**,** pattern**);**

writeln**(**'pos: '**:**20**,** ' -> '**,** pos**);**

// Negation regex valid

PrintUtils**.**PrintHeader**(**'Negation regex valid'**,** headerLength**);**

pattern **:=** RegexUtils**.**CreateNegationRegex**(**'Hagenb'**,** 'urger'**,** result**);**

pos **:=** RegexUtils**.**Matches**(**text**,** pattern**);**

writeln**(**'text: '**:**20**,** ' -> '**,** text**);**

writeln**(**'pattern: '**:**20**,** ' -> '**,** pattern**);**

writeln**(**'pos: '**:**20**,** ' -> '**,** pos**);**

// Character sequence valid

PrintUtils**.**PrintHeader**(**'Character sequence valid'**,** headerLength**);**

pattern **:=** RegexUtils**.**CreateCharacterSetRegex**(**'Hagen'**,** 'abc'**,** 'erger'**,** result**);**

pos **:=** RegexUtils**.**Matches**(**text**,** pattern**);**

writeln**(**'text: '**:**20**,** ' -> '**,** text**);**

writeln**(**'pattern: '**:**20**,** ' -> '**,** pattern**);**

writeln**(**'pos: '**:**20**,** ' -> '**,** pos**);**

// Character range valid

PrintUtils**.**PrintHeader**(**'Character range valid'**,** headerLength**);**

pattern **:=** RegexUtils**.**CreateCharacterRangeRegex**(**'Hagen'**,** 'erger'**,**'a'**,** 'f'**,** result**);**

pos **:=** RegexUtils**.**Matches**(**text**,** pattern**);**

writeln**(**'text: '**:**20**,** ' -> '**,** text**);**

writeln**(**'pattern: '**:**20**,** ' -> '**,** pattern**);**

writeln**(**'pos: '**:**20**,** ' -> '**,** pos**);**

// Character negated sequence valid

PrintUtils**.**PrintHeader**(**'Character negated sequence valid'**,** headerLength**);**

pattern **:=** RegexUtils**.**CreateNegationCharacterSetRegex**(**'Hagen'**,** 'cde'**,** 'erger'**,** result**);**

pos **:=** RegexUtils**.**Matches**(**text**,** pattern**);**

writeln**(**'text: '**:**20**,** ' -> '**,** text**);**

writeln**(**'pattern: '**:**20**,** ' -> '**,** pattern**);**

writeln**(**'pos: '**:**20**,** ' -> '**,** pos**);**

// Character negated range valid

PrintUtils**.**PrintHeader**(**'Character negated range valid'**,** headerLength**);**

pattern **:=** RegexUtils**.**CreateNegationCharacterRangeRegex**(**'Hagen'**,** 'erger'**,**'d'**,** 'f'**,** result**);**

pos **:=** RegexUtils**.**Matches**(**text**,** pattern**);**

writeln**(**'text: '**:**20**,** ' -> '**,** text**);**

writeln**(**'pattern: '**:**20**,** ' -> '**,** pattern**);**

writeln**(**'pos: '**:**20**,** ' -> '**,** pos**);**

// Multiple regex valid

PrintUtils**.**PrintHeader**(**'Character negated range valid'**,** headerLength**);**

pattern **:=** RegexUtils**.**CreateIgnoreRegex**(**'H'**,** 'ge'**,** 1**,** result**)**

**+** RegexUtils**.**CreateNegationRegex**(**''**,** 'm'**,** result**)**

**+** RegexUtils**.**CreateCharacterRangeRegex**(**''**,** ''**,** 'a'**,** 'c'**,** result**)**

**+** RegexUtils**.**CreateNegationCharacterRangeRegex**(**'er'**,** 'er'**,**'k'**,** 'm'**,** result**);**

pattern **:=** 'H.ge^m[a-c]er^[k-m]er'**;**

pos **:=** RegexUtils**.**Matches**(**text**,** pattern**);**

writeln**(**'text: '**:**20**,** ' -> '**,** text**);**

writeln**(**'pattern: '**:**20**,** ' -> '**,** pattern**);**

writeln**(**'pos: '**:**20**,** ' -> '**,** pos**);**

**End;**

**Begin**

PrintUtils**.**Print**(**'Tests for function IsValidString'**,** Green**,** White**,** headerLength**);**

TestIsValidString**;**

writeln**;**

PrintUtils**.**Print**(**'Tests for function CreateIgnoreRegex'**,** Green**,** White**);**

TestCreateIgnoreRegex**;**

writeln**;**

PrintUtils**.**Print**(**'Tests for procedure CreateNegationRegex'**,** Green**,** White**);**

TestCreateNegationRegex**;**

writeln**;**

PrintUtils**.**Print**(**'Tests for procedure CreateCharacterSetRegex'**,** Green**,** White**);**

TestCreateCharacterSetRegex**;**

writeln**;**

PrintUtils**.**Print**(**'Tests for procedure CreateCharacterRangeRegex'**,** Green**,** White**);**

TestCreateCharacterRangeRegex**;**

writeln**;**

PrintUtils**.**Print**(**'Tests for procedure CreateNegationCharacterSetRegex'**,** Green**,** White**);**

TestCreateNegationCharacterSetRegex**;**

writeln**;**

PrintUtils**.**Print**(**'Tests for procedure CreateNegationCharacterRangeRegex'**,** Green**,** White**);**

TestCreateNegationCharacterRangeRegex**;**

writeln**;**

PrintUtils**.**Print**(**'Tests for procedure IsValidPattern'**,** Green**,** White**);**

TestIsValidPattern**;**

writeln**;**

PrintUtils**.**Print**(**'Tests for procedure Matches'**,** Green**,** White**);**

TestMatches**;**

writeln**;**

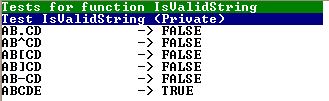
**End.**

# Tests

Folgend sind die Tests der beiden implementierten Units RegexUtils angeführt.

## IsValidString

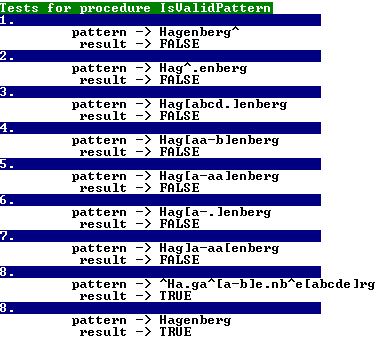
Diese Tests testen die intern verwendete Funktion IsValidString, die prüft ob die Zeichenkette vordefinierte Regex Zeichen enthält.



Sollte die Zeichenkette vordefinierte Regex Zeichen enthalten so wird False zurückgegeben. Sollten keine Regex Zeichen enthalten sein so wird True zurückgegeben.

## IsValidPattern

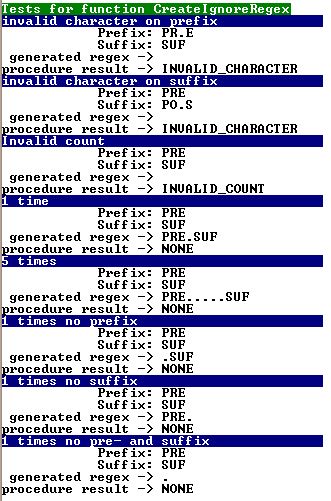
Diese Tests testen die interne Prozedur IsValidPattern, die prüft ob der übergebene Pattern gültige Regular Expressions enthält.



Die enthaltenen Regular Expressions werden korrekt validiert, und es ist zu sehen das auch mehrere Regular Expressions enthalten sein können oder gar keine.

## CreateIgnoreRegex

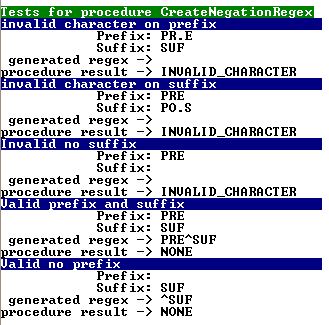
Diese Tests testen die Prozedur CreateIgnorePattern, die eine Regex erstellt, die die Position eine oder mehrerer Zeichen definiert, an denen das Zeichen ignoriert werden kann, also jedes Zeichen erlaubt ist.



Sollten Präfix oder Suffix ein vordefiniertes Regex Zeichen enthalten so wird ein Fehler gesetzt. Ebenso wenn die Anzahl der Wiederholungen des Regex kleiner gleich 0 ist. In Falle eines Fehlers wird immer ein Leerstring zurückgeliefert, ansonsten wird die erstellte Regex zurückgegeben.

## CreateNgationRegex

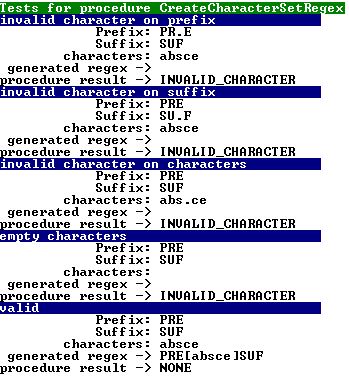
Die folgenden Tests testen die Prozedur CreateNegationPattern, die eine Regex erstellt, die das folgende Zeichen negiert, also jedes Zeichen ist erlaub, nur nicht das darauf Folgende.



Wenn der Präfix oder Suffix ein vordefiniertes Regex Zeichen enthält oder der Suffix eine leere Zeichenkette ist, dann wird ein Fehlern gesetzt, andererseits wird die erstellte Regex zurückgegeben.

## CreateCharacterSetRegex

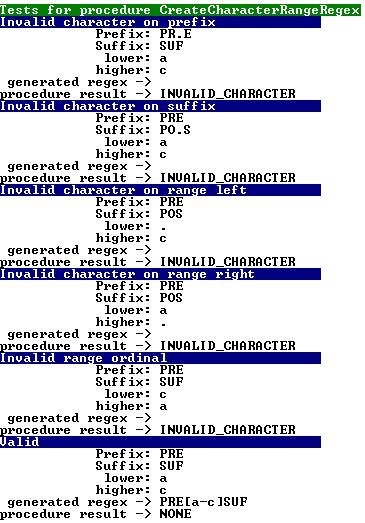
Die Tests testen die Prozedur CreateCharacterSetRegex, die eine Regex erstellt, die eine Menge von Zeichen definiert, welche an der Position im Pattern erlaubt sind.



Sollte der Präfix, Suffix oder die erlaubten Zeichen ein vordefiniertes Regex Zeichen enthalten oder ist die Zeichenkette der erlaubten Zeichen leer, so wird ein Fehler gesetzt andererseits wird die erstellte Regex zurückgegeben.

## CreateCharacterRangeRegex

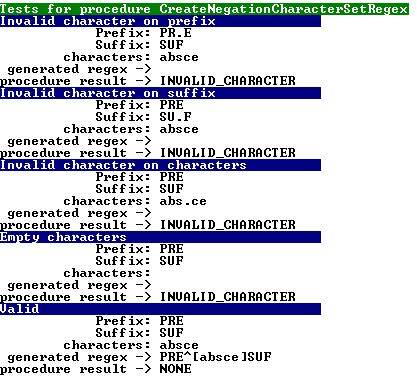
Diese Tests testen die Prozedur CreateCharacterRangeRegex, die eine Regex erstellet, welche einen Wertebereich für den Ordinal Wert des folgenden Zeichens spezifiziert.



Sollten der Präfix, Suffix, die obere oder untere untere Schranke ein vordefiniertes Regex Zeichen enthalten oder die Schranke ungültig sein (Bsp.: 10 - 4), dann wird ein Fehler gesetzt, andererseits wird die erstellte Regex zurückgegeben.

## CreateNegationCharacterSetRegex

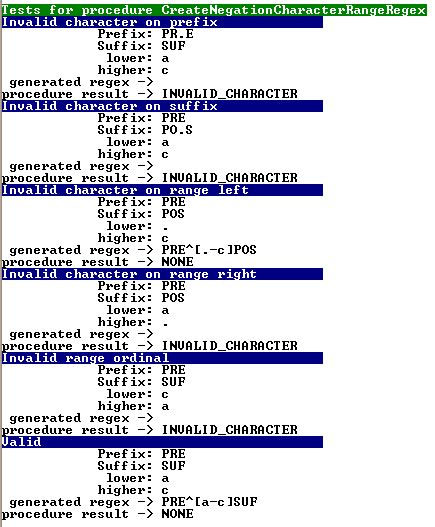
Diese Tests testen die Prozedur CreateNegationCharacterSetRangeRegex, welche eine Regex erstellt welche eine Menge von erlaubten Zeichen negiert und somit sind diese Zeichen für das folgende Zeichen nicht mehr erlaubt.



Sollte der Präfix, Suffix oder die erlaubten Zeichen ein vordefiniertes Regex Zeichen enthalten oder ist die Zeichenkette der erlaubten Zeichen leer, so wird ein Fehler gesetzt andererseits wird die erstellte Regex zurückgegeben.

## CreateNegationCharacterRangeRegex

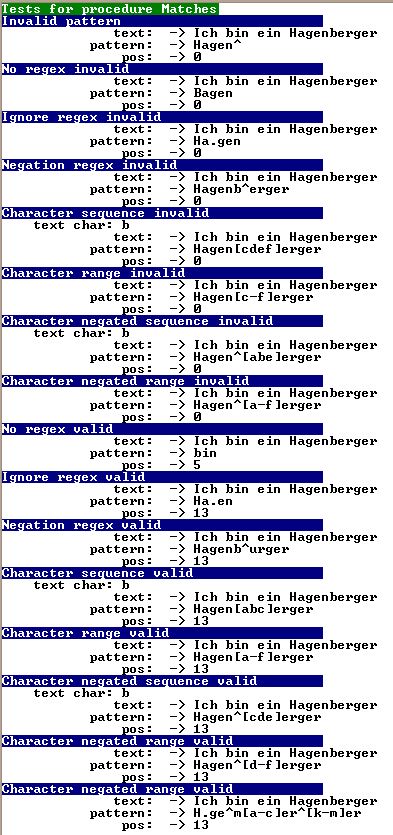
Diese Tests testen die Prozedur CreateNegationCharacterRangeRegex, die eine Regex erstellet, welche einen Wertebereich für den Ordinal Wert des folgenden Zeichens spezifiziert, dass nicht erlaubt ist.



Sollten der Präfix, Suffix, untere Schranke oder die obere Schranke ein vordefiniertes Regex Zeichen enthalten oder die Schranke ungültig sein (Bsp.: 10 - 4), dann wird ein Fehler gesetzt, andererseits wird die erstellte Regex zurückgegeben.

## Matches

Diese Tests testen die Funktion Matches, die einen Pattern in einer Zeichenkette sucht, wobei die Zeichenkette auch Regular Expression enthalten darf.



Es ist zu sehen das sich diese Funktion wie spezifiziert verhält und das auch Pattern ohne Regex sowie Pattern mit mehreren Regex korrekt behandelt werden.