Abgabetermin: Mi, 29. 03. 2023

☒ **Gr. 1,** Dr. S. Wagner **Name** Wurm Elias **Aufwand in h** 2h
☐ **Gr. 2,** Dr. D. Auer
☐ **Gr. 3,** Dr. G. Kronberger **Punkte** _____ **Kurzzeichen Tutor / Übungsleiter*in** _____ / _____

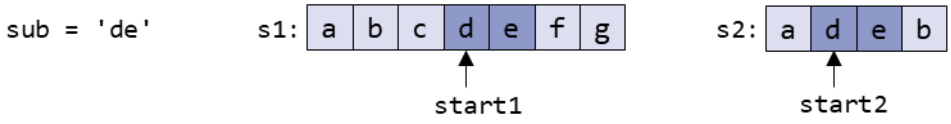
1. Längste gemeinsame Teilkette (12 Punkte)

Entwickeln Sie eine Pascal-Prozedur

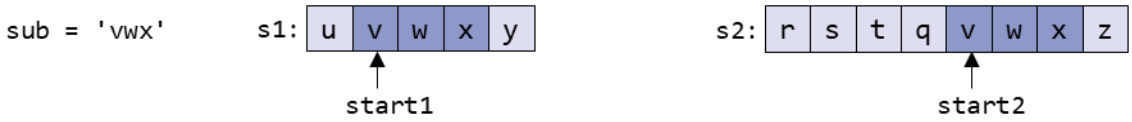
```
PROCEDURE FindLongestMatch(s1, s2: STRING; VAR sub: STRING;
                           VAR start1, start2: INTEGER);
```

die für zwei Zeichenketten `s1` und `s2` jene längste Teilkette (`sub`) findet, die sowohl in `s1` als auch in `s2` vorkommt. Die Anfangsposition der gefundenen längsten Teilkette muss für `s1` in `start1` und für `s2` in `start2` zurückgegeben werden.

Beispiel 1*:*



Beispiel 2*:*



Gibt es keine gemeinsame Teilkette, also nicht einmal ein gemeinsames Zeichen, liefert `sub` eine leere Zeichenkette und `start1` sowie `start2` den Wert 0.

Gibt es mehrere längste Teilketten, dann ist eine davon frei zu wählen und es sind für diese die entsprechenden Positionen in `start1` und `start2` zu liefern.

Für die Lösung von Teilaufgaben in `FindLongestMatch` können natürlich bekannte Verfahren zur Zeichenkettensuche eingesetzt werden, versuchen Sie aber, eine möglichst effiziente Gesamtlösung zu finden, und geben Sie eine Abschätzung der asymptotischen Laufzeitkomplexität dieser Gesamtlösung an.

2. Rotation von Zeichenketten (6 Punkte)

Entwickeln Sie ein Pascal-Programm, das überprüft, ob eine Zeichenkette *a* vom Datentyp `STRING` eine *zyklische Rotation* einer Zeichenkette *b* ist. Beispielsweise ist die Zeichenkette `Hagenberg` eine zyklische Rotation der Zeichenkette `bergHagen` (und umgekehrt).

Eine Zeichenkette *a* ist eine *zyklische Rotation* einer Zeichenkette *b*, wenn beide Zeichenketten die gleiche Länge *n* haben, die gleichen Zeichen enthalten und die Zeichen mit einer Distanz zwischen 0 und *n* – 1 in der gleichen Reihenfolge angeordnet sind.

Hinweis: Diese Aufgabe könnte man lösen, indem man die Konkatenation *a* + *a* (z. B. `Hagen`**`berg-Hagen`**`berg`) bildet und darin die Zeichenkette *b* sucht. Entwickeln Sie für diese Aufgabe eine Lösung, die ohne Konkatenation einer Zeichenkette auskommt. Modifizieren Sie dazu einen in der Übung entwickelten Algorithmus (z.B. `BruteForceSearch`).

3. Suche in Zeichen*strömen* (6 Punkte)

In den ADF2-Übungen haben Sie verschiede Algorithmen zur Suche in Zeichen*ketten* kennengelernt. Gesucht ist ein Algorithmus für die Suche einer Zeichenkette *p* (*pattern*) in einem Zeichen*strom*, der zeichenweise von der Standardeingabe eingelesen wird. Modifizieren Sie hierfür einen in der Übung entwickelten Algorithmus.

Hinweis: Für die Suche einer Zeichenkette *p* mit der Länge *m* ist ein Puffer erforderlich, der die letzten *m* Zeichen der Eingabe speichert.

Hinweise:

1. Geben Sie für alle Ihre Lösungen immer eine „Lösungsidee“ an.
2. Dokumentieren und kommentieren Sie Ihre Algorithmen.
3. Bei Programmen: Geben Sie immer auch Testfälle ab, an denen man erkennen kann, dass Ihr Programm funktioniert, und dass es auch in Fehlersituation entsprechend reagiert.

Aufgabe 1:

Zeitaufwand: 30min

Einfach bruteforce für jeden char in s1 alle chars in s2 durchgehen und jedesmal bei dem man einen größeren gemeinsamen substring gefunden hat den speichern, bis man durch ist.

Laufzeitkomplexität:

worst case:
O(m*n)
m = length(s1)
n = length(s2)

Code:

```
program FindLongestMatchingSubstring;

procedure FindLongestMatch(s1, s2: STRING; var sub: STRING; var start1,
start2: INTEGER);
var
  i, j, s1Len, s2Len, len, maxlen: INTEGER;
begin
  // init values
  s1Len := LENGTH(s1);
  s2Len := LENGTH(s2);
  maxlen := 0;
  sub := '';
  start1 := 0;
  start2 := 0;

  for i := 1 to s1Len do
    for j := 1 to s2Len do
      if s1[i] = s2[j] then
      begin
        len := 1;
        // find matching substring
        while (i + len - 1 <= s1Len) and (j + len - 1 <= s2Len) and (s1[i +
len - 1] = s2[j + len - 1]) do
          INC(len);
        // if found matching substring is longer than current longest found
matching substring save it
        if (len - 1) > maxlen then
        begin
          maxlen := len - 1;
          sub := COPY(s1, i, maxlen);
          start1 := i;
          start2 := j;
        end;
      end;
end;
```

```
procedure TestFindLongestMatch(s1, s2: STRING);
var
  sub: string;
  start1, start2: integer;
begin
  WriteLn('Find longest matching substring for: ''', s1, ''' and ''', s2,
'''');
  FindLongestMatch(s1, s2, sub, start1, start2);
  WriteLn('sub = "', sub, '"');
  WriteLn('start1 = ', start1);
  WriteLn('start2 = ', start2);
  WriteLn;
end;

begin
  // Test Case 1: Both strings are empty
  TestFindLongestMatch('', '');

  // Test Case 2: One string is empty
  TestFindLongestMatch('hello', '');

  // Test Case 3: No common substring
  TestFindLongestMatch('hello', 'world');

  // Test Case 4: Common substring is at the beginning
  TestFindLongestMatch('hello world', 'hello there');

  // Test Case 5: Common substring is in the middle
  TestFindLongestMatch('the quick brown fox jumps over the lazy dog', 'the
quick blue fox jumps over the lazy cat');

  // Test Case 6: Multiple common substrings of the same length
  TestFindLongestMatch('abcdabcd', 'efghefgh');

  // Test Case 7: Multiple common substrings of different lengths
  TestFindLongestMatch('abcdabcde', 'efghefgh');
end.
```

Ergebnis Tests:

```
Find longest matching substring for: '' and ''
sub = ""
start1 = 0
start2 = 0

Find longest matching substring for: 'hello' and ''
sub = ""
start1 = 0
start2 = 0

Find longest matching substring for: 'hello' and 'world'
sub = "l"
start1 = 3
start2 = 4

Find longest matching substring for: 'hello world' and 'hello there'
sub = "hello "
start1 = 1
start2 = 1

Find longest matching substring for: 'the quick brown fox jumps over the lazy dog' and 'the quick blue fox jumps over the lazy cat'
sub = " fox jumps over the lazy "
start1 = 16
start2 = 15

Find longest matching substring for: 'abcdabcd' and 'efghefgh'
sub = ""
start1 = 0
start2 = 0

Find longest matching substring for: 'abcdabcde' and 'efghefgh'
sub = "e"
start1 = 9
start2 = 1
```

Aufgabe 2:

Zeitaufwand: 30min

Über ersten String iterieren und bei jedem i einfach brutesearchpattern matching wo beim ersten string zum index j immer das i dazu gerechnet wird und falls i + j beim pattern searching out of bounds ist wird j auf 1 - i gesetzt. Abbruch bedingung ist chars nicht gleich bzw. j = i

Code:

```
program CyclicRotation;

procedure IsCyclicRotation(s1, s2: STRING; var pos: INTEGER; var ret:
boolean);
var
  i,j,k,s1Len, s2Len: INTEGER;
begin
  s1Len := LENGTH(s1);
  s2Len := LENGTH(s2);

  ret := false;
  pos := 0;

  i := 0;
  if((s1Len = 0) and (s2Len = 0)) then
  begin
    ret := true;
    pos := 1;
  end else if(s1Len = s2Len) then
    while (i < s1Len) and (ret = false) do
    begin
      j := 1;
      k := 1;
      while((s2[i+j] = s1[k]) and (k <= s1Len)) do
      begin
        Inc(j);
        Inc(k);
        if(((i + j) > s1Len)) then
          j := 1 - i// if out of bounds set to start
        ;
      end;
      if(k = s1Len+1) then
      begin
        ret := true;
        pos := i + 1;
      end;
      Inc(i);
    end;
end;
```

```
procedure TestIsCyclicRotation(s1, s2: STRING);
var
  pos: INTEGER;
  ret: BOOLEAN;
begin
  IsCyclicRotation(s1, s2, pos, ret);
  WriteLn('Is: ''', s2, ''' a cyclic rotation of ''', s1, ''': ', ret);
  if(ret) then writeln('On Position: ', pos);
  WriteLn;
end;

begin
  // Test Case 1:
  TestIsCyclicRotation('', ''); // expected: True, On Position: 1

  // Test Case 2:
  TestIsCyclicRotation('abc', ''); // expected: False

  // Test Case 3:
  TestIsCyclicRotation('', 'abc'); // expected: False

  // Test Case 4:
  TestIsCyclicRotation('HelloWorld', 'HelloWorld'); // expected: True, On
Position: 1

  // Test Case 5:
  TestIsCyclicRotation('HelloWorld', 'WorldHello'); // expected: True, On
Position: 6

  // Test Case 6:
  TestIsCyclicRotation('abcd', 'abce'); // expected: False
end.
```

Ergebnis Tests:

```
Is: '' a cyclic rotation of '': TRUE
On Position: 1

Is: '' a cyclic rotation of 'abc': FALSE

Is: 'abc' a cyclic rotation of '': FALSE

Is: 'HelloWorld' a cyclic rotation of 'HelloWorld': TRUE
On Position: 1

Is: 'WorldHello' a cyclic rotation of 'HelloWorld': TRUE
On Position: 6

Is: 'abce' a cyclic rotation of 'abcd': FALSE
```

Aufgabe 3:

Zeitaufwand: 1h

KMP pattern match algorithmus verwenden um pattern zuerst vorzubearbeiten und danach zeichen für zeichen die eingabe durchgehen. Somit ist es egal wenn ein pattern nicht match und man nicht zurückspringen muss zu i – pLen sondern einfach zeichen für zeichen durchiteriert. Falls das pattern nicht gefunden wird aber man trotzdem das program beenden kann hab ich mich für ein terminate char entschieden, in dem Fall !.

Code:

```
program PatternSearch;
var
  pattern: String;
  pLen, i, j, pos: Integer;
  next: array of integer;
  c: Char;

begin
  Write('Enter pattern: ');
  ReadLn(pattern);
  pLen := Length(pattern);

  SetLength(next, pLen + 1);

  i := 1;
  j := 0;
  next[i] := 0;

  while(i < pLen) do
    if((j = 0) or (pattern[i] = pattern[j])) then
    begin
      Inc(i);
      Inc(j);
      if (pattern[i] <> pattern[j]) then next[i] := j else next[i] := next[j];
    end else j := next[j];

  i := 1;
  pos := 1;

  write('char > '); readln(c);
  while ((i <= pLen) and (c <> '!')) do // my terminate char is a !
    if((i = 0) or (c = pattern[i])) then
    begin
      Inc(pos);
      write('char > '); readln(c);
      Inc(i);
    end else i := next[i];
```

```
  WriteLn;
  if (i > pLen) then WriteLn('pattern was found on position: ', pos - pLen)
else WriteLn('not found');
  WriteLn;
end.
```

Tests:

```
Enter pattern: abc
char > a
char > b
char > c
char > !

pattern was found on position: 1
```

```
Enter pattern: abc
char > a
char > b
char > !

not found
```

```
Enter pattern: abcdabc
char > a
char > b
char > c
char > d
char > a
char > b
char > c
char > !

pattern was found on position: 1
```

```
Enter pattern: abcdabc
char > a
char > b
char > c
char > d
char > a
char > b
char > x
char > !

not found
```

```
Enter pattern: abc
char > x
char > y
char > z
char > a
char > b
char > c
char > !

pattern was found on position: 4

Enter pattern: a
char > b
char > c
char > a
char > !

pattern was found on position: 3
```