

☐ Gr. 1, Dr. G. Kronberger

Name Andreas Roither Aufwand in h 6 h

☐ Gr. 2, Dr. H. Gruber



Gr. 3, Dr. D. Auer

Punkte            Kurzzeichen Tutor / Übungsleiter            /           

## 1. *m*-Ketten-Problem

(4 + 6 Punkte)

- a) *Definition*: Eine Zeichenkette ist eine *m*-Kette, wenn sie höchstens *m* unterschiedliche Zeichen enthält.

*Beispiele*: Die drei Zeichenketten *a*, *ab* und *abcbaac* sind 3-Ketten, die Zeichenkette *abcd* ist aber keine 3-Kette mehr, sondern eine 4-Kette. Eine Zeichenkette *s* der Länge *n* ist natürlich eine *n*-Kette, von Interesse ist aber das kleinste *m* für welches die Bedingung aus der Definition oben für *s* noch gilt. Entwickeln Sie daher eine möglichst effiziente Funktion

```
FUNCTION MinM(s: STRING): INTEGER;
```

zur Ermittlung des kleinsten *m* für eine Zeichenkette *s*.

- b) Gegeben sei eine nichtleere Zeichenkette *s* und eine ganze Zahl *m* mit  $1 \leq m \leq \text{Length}(s)$ . Entwickeln Sie eine möglichst effiziente Funktion

```
FUNCTION MaxMStringLen(s: STRING, m: INTEGER): INTEGER;
```

welche die Länge der längsten *m*-Kette, die als Teilkette in *s* enthalten ist, liefert.

## 2. Wildcard Pattern Matching

(4 + (5 + 5) Punkte)

Viele Programme, z. B. Texteditoren und Kommandozeilen-Interpreter diverser Betriebssysteme (engl. *shells*), verwenden eine spezielle Variante der Zeichenkettensuche, die in der Musterkette Platzhalter (auch Jokerzeichen, engl. *wildcards*, genannt) zulässt. Denken Sie z. B. an den MS-DOS/Windows-Befehl *del \*.\** bzw. an das äquivalente UNIX-Kommando *rm \**. Hier muss festgestellt werden, ob die Musterkette (*\*.\** bzw. *\**) zu einem Dateinamen im aktuellen Verzeichnis passt.

Jokerzeichen dürfen nur in Musterketten vorkommen: Dabei steht das Jokerzeichen *?* für *ein* beliebiges Zeichen in der Zeichenkette und das Jokerzeichen *\** für eine *beliebige Anzahl* (null oder mehr) beliebiger Zeichen in der Zeichenkette. Jokerzeichen können auch gemischt und mehrfach in einer Musterkette vorkommen.

Nehmen Sie an, dass sowohl die Muster- als auch die Zeichenkette durch das spezielle Endezeichen *\$* abgeschlossen ist, welches innerhalb der Ketten nicht vorkommt. Folgende Tabelle zeigt einige einfache *Beispiele*:

Musterkette <i>p</i>	Zeichenkette <i>s</i>	<i>p</i> und <i>s</i> passen zusammen?
ABC\$	ABC\$	ja
AB <i>C</i> \$	AB\$	nein
ABC\$	AB <i>C</i> D\$	nein
A? <i>C</i> \$	A <i>X</i> C\$	ja
*\$	<i>beliebige auch leere Kette</i>	ja
A* <i>C</i> \$	AC\$	ja
A* <i>C</i> \$	A <i>X</i> Y <i>Z</i> C\$	ja

- a) Erweitern/ändern Sie den *BruteForce*-Algorithmus für die Zeichenkettensuche so, dass er obige Aufgabenstellung bewältigt, jedoch als Jokerzeichen nur ? (auch mehrfach) in der Musterkette vorkommen darf.
  - b) Die zusätzliche Behandlung des Jokerzeichens \* ist mit den Standard-Algorithmen leider nicht mehr so einfach möglich. Allerdings lässt sich das Problem relativ einfach mittels Rekursion lösen:
    - 1. Definieren Sie zuerst ein rekursives Prädikat *Matching(p, s)*, das *true* liefert, wenn *p* und *s* zusammenpassen, sonst *false*. Zerlegen Sie dabei sowohl *p* als auch *s* "geschickt" in zwei Teile: in das erste Zeichen und den Rest der Kette.
    - 2. Implementieren Sie das Prädikat *Matching* in Form einer rekursiven Funktion und testen Sie diese ausführlich.
- 

**Für besonders Interessierte:** Implementieren Sie eine iterative Variante des Prädikats *Matching* aus 2.b.1, testen Sie diese ausführlich und stellen Sie Laufzeitvergleiche der rekursiven und iterativen Variante für lange Eingabeketten an. Als „**Belohnung**“ gibt es bis zu vier Zusatzpunkte.

# Übung 2

## Aufgabe 1

### Lösungsidee

Bei MinM wird in eine Liste eingefügt falls ein Zeichen darin noch nicht enthalten ist. Am Schluss wird gezählt wie viele Zeichen in der Liste enthalten sind. Diese Anzahl gibt MinM zurück. Bei MaxMStringLen wird solange in eine Liste eingefügt bis m erreicht wird. Falls die Anzahl unterschiedlicher Zeichen in der Liste überschritten wurde, löscht RemoveFirst das erste Zeichen in der Liste. Falls die aktuelle Länge größer ist als die vorher gespeicherte Länge wird die aktuelle Länge gespeichert.

```

1 PROGRAM kette;
  (* Implementation with lists *)
3
4 TYPE
5   nodePtr = ^listElement;
6   listElement = RECORD
7     next: nodePtr;
8     c: Char;
9   END; (* RECORD *)
10
11 (* Creates a new node*)
12 FUNCTION NewNode(c : Char): nodePtr;
13 VAR
14   node : nodePtr;
15 BEGIN
16   New(node);
17   node^.next := NIL;
18   node^.c := c;
19   NewNode := node;
20 END;
21
22 (* Appends a Node to a List *)
23 PROCEDURE Append(var list : nodePtr; element : nodePtr);
24 VAR
25   tmp : nodePtr;
26 BEGIN
27   if list = NIL THEN list := element ELSE
28   BEGIN
29     tmp := list;
30     WHILE tmp^.next <> NIL DO tmp := tmp^.next;
31
32     tmp^.next := element;
33   END;
34 END;
35
36 (* recursive; disposes every node in a list *)
37 PROCEDURE ClearList(var list : nodePtr);
38 BEGIN
39   IF list <> NIL THEN
40   BEGIN
41     IF list^.next = NIL THEN dispose(list) ELSE ClearList(list^.next);
42   END;
43 END;

```

```

45  (* Counts nodes in a list *)
    FUNCTION CountNodes(n : nodePtr) : INTEGER;
47  VAR
    count : INTEGER;
49  BEGIN
    count := 0;
51  WHILE n <> NIL DO BEGIN
    Inc(count);
53  n := n^.next;
    END;
55  CountNodes := count;
    END;
57
    (* Removes the first node of a list *)
59  PROCEDURE RemoveFirst(var list : nodePtr);
    VAR
61  temp : nodePtr;
    BEGIN
63  IF list <> NIL THEN BEGIN
    temp := list;
65  list := list^.next;
    Dispose(temp);
67  END;
    END;
69
    (* Check if char exists in the list;
    Returns TRUE OR FALSE *)
71  FUNCTION CharExists(list : nodePtr; c : Char): Boolean;
    VAR
73  n : nodePtr;
    BEGIN
75  n := list;
77
    WHILE n <> NIL DO BEGIN
79  IF n^.c = c THEN BEGIN
    CharExists := TRUE;
81  break;
    END;
83  n := n^.next;
    END;
85
    IF n = NIL THEN CharExists := FALSE;
87  END;
89
    (* Counts different chars in a list;
    RETURNS 0 if empty*)
91  FUNCTION CountDistinct(list: nodePtr): Integer;
    VAR
93  temp, temp2 : nodePtr;
    BEGIN
95  IF list <> NIL THEN
    BEGIN
97  temp := list;
    temp2 := NIL;
99
    WHILE temp <> NIL DO BEGIN
101  IF NOT CharExists(temp2,temp^.c) THEN Append(temp2, NewNode(temp^.c));

```

```

    temp := temp^.next;
103  END;

    CountDistinct := CountNodes(temp2);
    ClearList(temp2);
107  END
    ELSE
109    CountDistinct := 0;
    END;
111

(* Prints out list *)
113  PROCEDURE PrintList(list : nodePtr);
    VAR
115    n : nodePtr;
    BEGIN
117    n := list;

    WHILE n <> NIL DO BEGIN
119      Write(n^.c);
121      n := n^.next;
    END;
123    WriteLn;
    END;
125

(* Insert to string from a list RETURNS STRING*)
127  FUNCTION InsertinString(list : nodePtr): STRING;
    VAR
129    n : nodePtr;
    s : STRING;
131  BEGIN
    n := list;
133    s := "";

    WHILE n <> NIL DO BEGIN
135      s := Concat(s,n^.c);
137      n := n^.next;
    END;
139    InsertinString := s;
    END;
141

(* Implementation with Single Linked List *)
143  FUNCTION MinM(s: STRING) : Integer;
    VAR
145    i : Integer;
    list : nodePtr;
147  BEGIN
    list := NIL;
149

    FOR i := 1 TO Length(s) DO BEGIN
151      IF NOT CharExists(list,s[i]) THEN Append(list, NewNode(s[i]));
    END;
153

    MinM := CountNodes(list);
155  END;

(* Implementation with Single Linked List *)
157  FUNCTION MaxMStringLen(var longestS: STRING; s: STRING; m: Integer): Integer;
159  VAR

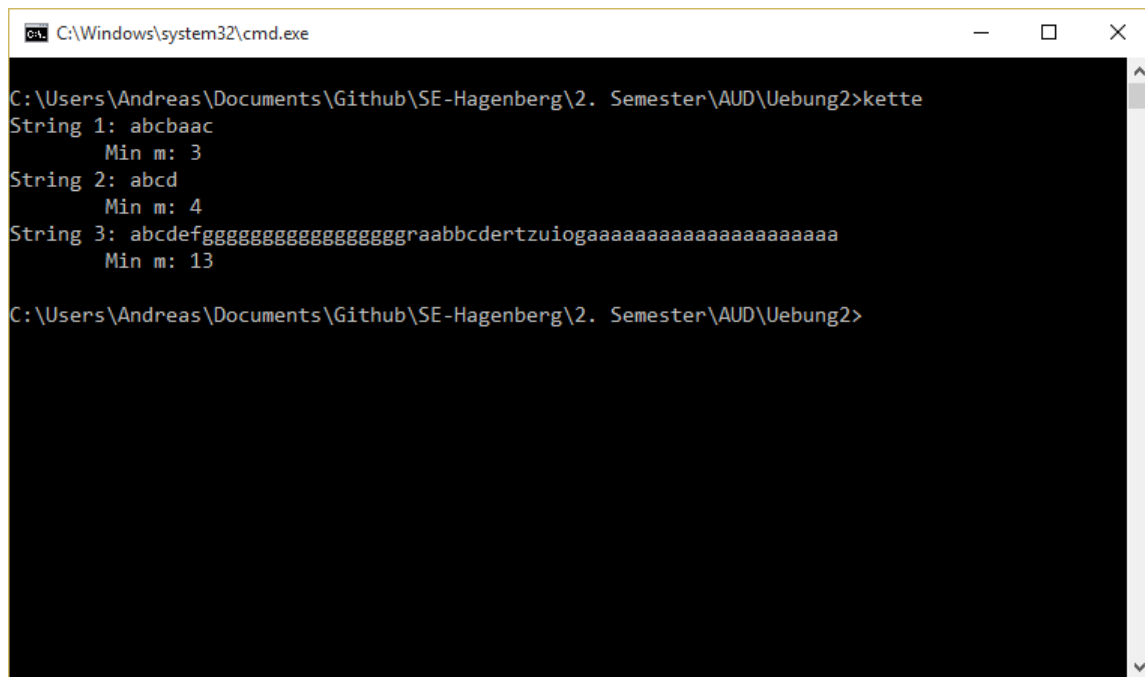
```

```

161   i, count, tempCount, maxLength : Integer;
162   list : nodePtr;
163 BEGIN
164   list := NIL;
165   count := 0;
166   maxLength := 0;
167
168   FOR i := 1 TO Length(s) DO BEGIN
169
170     Append(list, NewNode(s[i]));
171     tempCount := CountDistinct(list);
172
173     IF tempCount > m THEN
174     BEGIN
175       RemoveFirst(list);
176     END
177     ELSE
178       count := count + 1;
179
180     IF count > maxLength THEN BEGIN
181       maxLength := count;
182       longestS := InsertinString(list);
183     END;
184   END;
185   MaxMStringLen := maxLength;
186 END;
187
188 VAR
189   s1, s2, s3, longest : String;
190 BEGIN
191   s1 := 'abcbaac';
192   s2 := 'abcd';
193   s3 := 'abcdefggggggggggggggggraabbcderztuiogaaaaaaaaaaaaaaaaaaaaa';
194   s3 := 'abcdefggggggggggggggggraabbcderztuiogaaaaaaaaaaaaaaaaaaaaa';
195   longest := '';
196
197   WriteLn('String 1: ', s1, #13#10#9, 'Min m: ', MinM(s1));
198   WriteLn('String 2: ', s2, #13#10#9, 'Min m: ', MinM(s2));
199   WriteLn('String 3: ', s3, #13#10#9, 'Min m: ', MinM(s3));
200
201   WriteLn('-----');
202   WriteLn('String 1 mit m 2: ', s1, #13#10#9, 'MaxM: ', MaxMStringLen(longest,s1,2),
203     #13#10#9, 'Longest substring: ', longest, #13#10);
204   WriteLn('String 2 mit m 4: ', s2, #13#10#9, 'MaxM: ', MaxMStringLen(longest,s2,4),
205     #13#10#9, 'Longest substring: ', longest, #13#10);
206   WriteLn('String 3 mit m 5: ', s3, #13#10#9, 'MaxM: ', MaxMStringLen(longest,s3,5),
207     #13#10#9, 'Longest substring: ', longest, #13#10);
208   WriteLn('String 3 mit m 7: ', s3, #13#10#9, 'MaxM: ', MaxMStringLen(longest,s3,7),
209     #13#10#9, 'Longest substring: ', longest, #13#10);
210 END.

```

Kette.pas



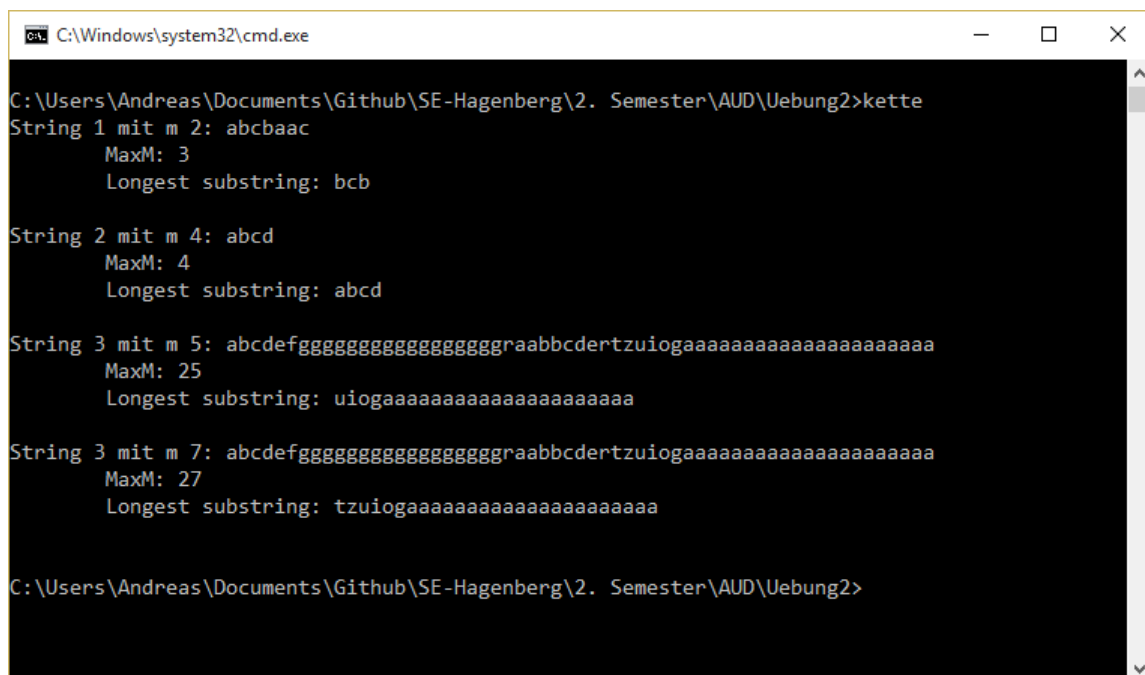
```
C:\Windows\system32\cmd.exe

C:\Users\Andreas\Documents\Github\SE-Hagenberg\2. Semester\AUD\Uebung2>kette
String 1: abcbaac
      Min m: 3
String 2: abcd
      Min m: 4
String 3: abcdefgggggggggggggggggggggraabbcderztzuiogaaaaaaaaaaaaaaaaaaaaa
      Min m: 13

C:\Users\Andreas\Documents\Github\SE-Hagenberg\2. Semester\AUD\Uebung2>
```

Abbildung 1: Ausgabe 1a

Ausgegeben wird die Anzahl der unterschiedlichen Zeichen der oben angegebenen Zeichenketten.



```
C:\Windows\system32\cmd.exe

C:\Users\Andreas\Documents\Github\SE-Hagenberg\2. Semester\AUD\Uebung2>kette
String 1 mit m 2: abcbaac
      MaxM: 3
      Longest substring: bcb

String 2 mit m 4: abcd
      MaxM: 4
      Longest substring: abcd

String 3 mit m 5: abcdefgggggggggggggggggggggraabbcderztzuiogaaaaaaaaaaaaaaaaaaaaa
      MaxM: 25
      Longest substring: uioaaaaaaaaaaaaaaaaaaaaa

String 3 mit m 7: abcdefgggggggggggggggggggggraabbcderztzuiogaaaaaaaaaaaaaaaaaaaaa
      MaxM: 27
      Longest substring: tzuiogaaaaaaaaaaaaaaaaaaaaa

C:\Users\Andreas\Documents\Github\SE-Hagenberg\2. Semester\AUD\Uebung2>
```

Abbildung 2: Ausgabe 1b

Hier wird der längste Substring mit maximalen m eines Strings ausgegeben.

## Aufgabe 2

### Lösungsidee

Das Matching funktioniert ähnlich wie die BruteForce Methode. Es wird von links nach rechts durch den string gegangen. Dabei wird bei jedem Aufruf ein Teil des string nicht mehr mit übergeben, für die Zeichen “\*” und “?” werden dabei spezielle Aktionen durchgeführt. Falls die Zeichenfolge nicht mehr übereinstimmt wird False zurückgegeben, andernfalls wird True zurückgegeben.

```

PROGRAM wildcard;

2  (* Matching Going from the left to right
   recursive *)
4  FUNCTION Matching(p, s : STRING):Boolean;
6  VAR
   i,j:Integer;
8  BEGIN
   i := 1;
10  j := 1;

12  WHILE (p[j] <> '*') AND (j <= length(p)) AND ((s[i] = p[j]) OR
   (p[j] = '?')) DO BEGIN
14     i := i + 1;
     j := j + 1;
16  END;

18  IF (p[j] <> '*') AND (i <= length(s)) THEN BEGIN
     Matching := Matching(p, Copy(s, 2, length(s)))
20  END
   ELSE IF (j <= length(p)) AND (i <= length(s)) THEN BEGIN
22     Matching := Matching(Copy(p, j + 1, Length(p)), Copy(s, 2, length(s)));
   END
   ELSE IF ((j >= length(p)) AND (i >= length(s))) OR
24     ((j = length(p)) AND (p[j] = '*')) THEN BEGIN
26     Matching := True;
   END
   ELSE Begin
28     Matching := False;
30  END;
   END;

32  VAR
34   s,p : STRING;
BEGIN
36   s := '?c?';
   p := 'aca';
38
   IF Matching(s, p) THEN WriteLn(s, ' and ', p, #9#9, ' True')
40  ELSE WriteLn(s, ' and ', p, #9, ' False');

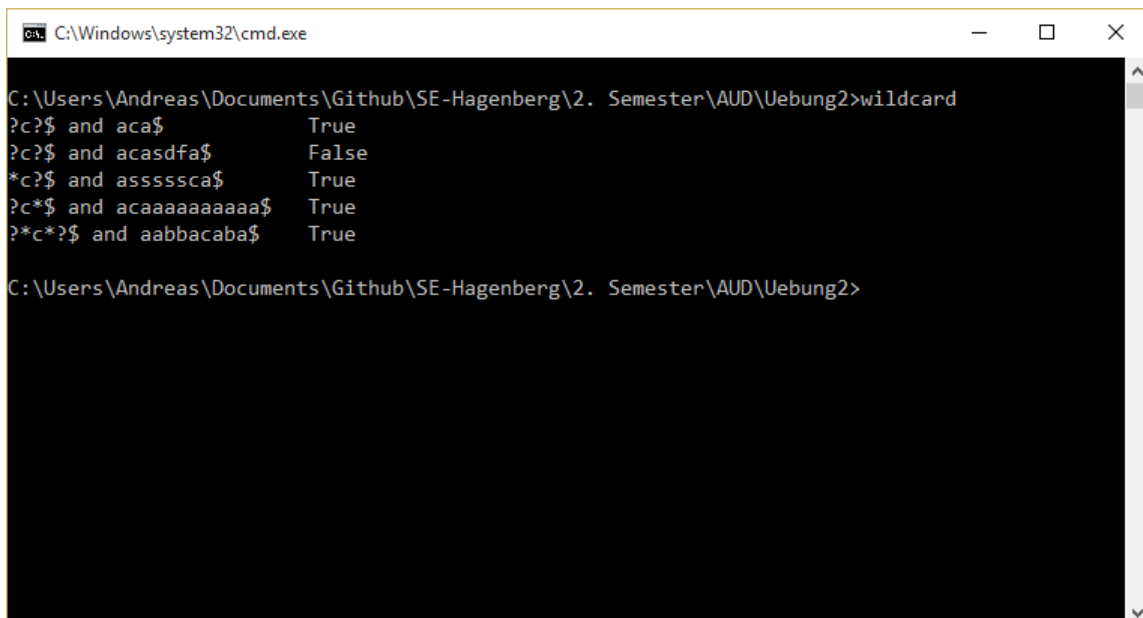
42   s := '?c?';
   p := 'acasdfa';
44   IF Matching(s, p) THEN WriteLn(s, ' and ', p, #9, ' True')
   ELSE WriteLn(s, ' and ', p, #9, ' False');
46
   s := '*c?';

```



```
48 p := 'assssca$';  
   IF Matching(s, p) THEN WriteLn(s, ' and ', p, #9, ' True')  
50 ELSE WriteLn(s, ' and ', p, #9, ' False');  
  
52 s := '?c*$';  
   p := 'aaaaaaaaaaaa$';  
54 IF Matching(s, p) THEN WriteLn(s, ' and ', p, #9, ' True')  
   ELSE WriteLn(s, ' and ', p, #9, ' False');  
  
56 s := '?*c*?$';  
58 p := 'aabbacaba$';  
   IF Matching(s, p) THEN WriteLn(s, ' and ', p, #9, ' True')  
60 ELSE WriteLn(s, ' and ', p, #9, ' False');  
END.
```

wildcard.pas



```
C:\Windows\system32\cmd.exe  
  
C:\Users\Andreas\Documents\Github\SE-Hagenberg\2. Semester\AUD\Uebung2>wildcard  
?c?$ and aca$           True  
?c?$ and acasdfa$       False  
*c?$ and assssca$       True  
?c*$ and aaaaaaaaaaaa$  True  
?*c*?$ and aabbacaba$   True  
  
C:\Users\Andreas\Documents\Github\SE-Hagenberg\2. Semester\AUD\Uebung2>
```

Abbildung 3: Ausgabe Matching

Ausgabe für die jeweiligen Zeichenketten.