

Übung 5

Arbeitsaufwand insgesamt: 12h

Inhaltsverzeichnis

Übung 5	1
Teil 1 – Proteinbiosynthese	2
Lösungsidee:	2
Lösung:	2
Testfälle:	3
Teil 2 – Primfaktorenzerlegung	8
Lösungsidee:	8
Lösung:	8
Testfälle:	9
Teil 3 – Punktmatrix.....	13
Lösungsidee:	13
Lösung:	13
Testfälle:	14

Teil 1 – Proteinbiosynthese

Ziel der Übung ist die Ausführung der Vorgänge der Transkription und Translation in C++ um eine Aminosäurekette zu erhalten.

Lösungsidee:

Es muss eine DNA-Sequenz eingelesen werden und diese auf Richtigkeit überprüft werden bzw. auf Großschreibung genormt werden. Dazu könnte die Funktion „read_DNA()“ aus der letzten Übung (UE04) wiederverwendet werden. Für die Transkription reicht es aus, eine Schleife zu schreiben, in der jedes „T“ in der Sequenz mit einem „U“ ausgetauscht wird.

Für die Translation müssen jeweils 3 Buchstaben aus der RNA-Sequenz einer Aminosäuren-Abkürzung zugeordnet werden. Da jede Kombination aus „A“, „U“, „G“ und „C“ überprüft werden muss, würde ich die verschiedenen Kombinationsmöglichkeiten gemeinsam mit der zugehörigen Abkürzung in einem Vektor speichern. Diese Wertepaare können aus einer Datei eingelesen werden, um ewig lange String-Literale und statischen Code (dutzende IF-Abfragen) zu vermeiden.

Somit muss nur noch nach dem Start-Codon gesucht werden und ab dieser Stelle bis zum Stop-Codon (oder bis zum Ende der Sequenz – das ging aus der Angabe nicht hervor) in einer Schleife alle 3 Zeichen langen RNA-Subsequenzen (welche sich aber nicht überschneiden) im vorher erwähnten Vektor gesucht werden und die zugehörige Abkürzung an einen anfangs leeren String angehängt werden und nach der Schleife ausgegeben werden.

Grundsätzlich werden also I/O, strings, Vektoren und Filestreams benötigt. Das resultiert in folgenden benötigten Bibliotheken: iostream, string, vector und fstream.

Lösung:

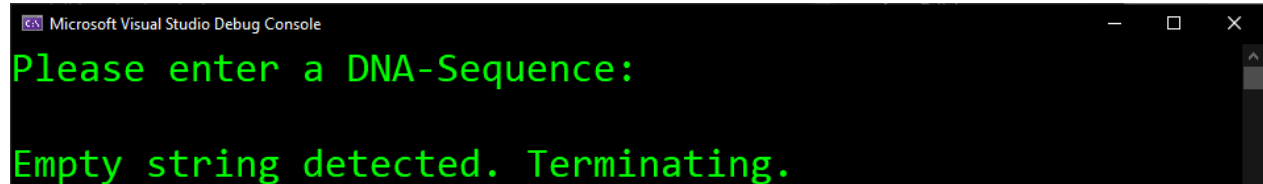
Als C++ Projekt „Teil_1“ im Archiv

Testfälle:

Fehlerhafte Eingaben:

- Leerer String
- Falsche Zeichen
- Nicht leer, aber zu wenig für Translation

Leere Eingabe:

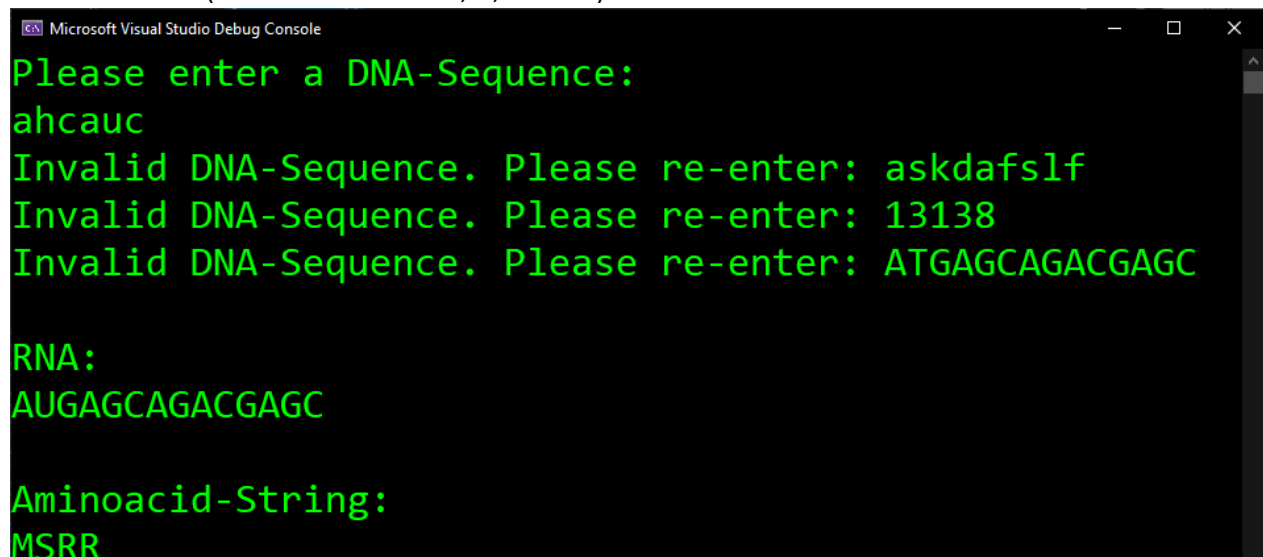


```
Microsoft Visual Studio Debug Console

Please enter a DNA-Sequence:

Empty string detected. Terminating.
```

Falsche Zeichen (andere Zeichen als A, G, C und T):



```
Microsoft Visual Studio Debug Console

Please enter a DNA-Sequence:
ahcauc
Invalid DNA-Sequence. Please re-enter: askdafs1f
Invalid DNA-Sequence. Please re-enter: 13138
Invalid DNA-Sequence. Please re-enter: ATGAGCAGACGAGC

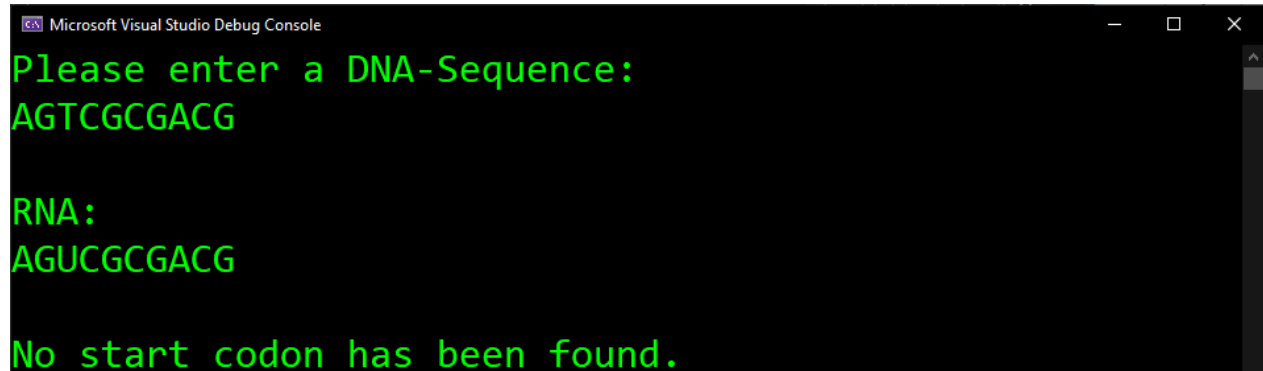
RNA:
AUGAGCAGACGAGC

Aminoacid-String:
MSRR
```

Valide Eingaben

- Beliebig viele Zeichen, kein Start-Codon
- Beliebig viele Zeichen, begrenzt durch Sequenz-Ende
- Beliebig viele Zeichen, begrenzt durch Stop-Codon
- Kleinbuchstaben
- Referenzbeispiel

Beliebig viele Zeichen, kein Start-Codon:



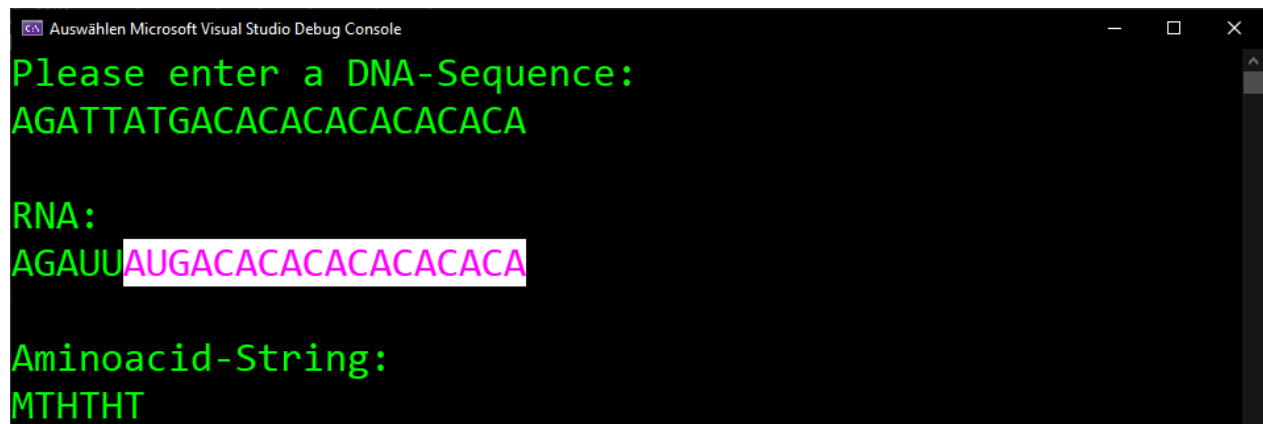
```
Microsoft Visual Studio Debug Console

Please enter a DNA-Sequence:
AGTCGCGACG

RNA:
AGUCGCGACG

No start codon has been found.
```

Beliebig viele Zeichen. begrenzt durch Sequenzende:

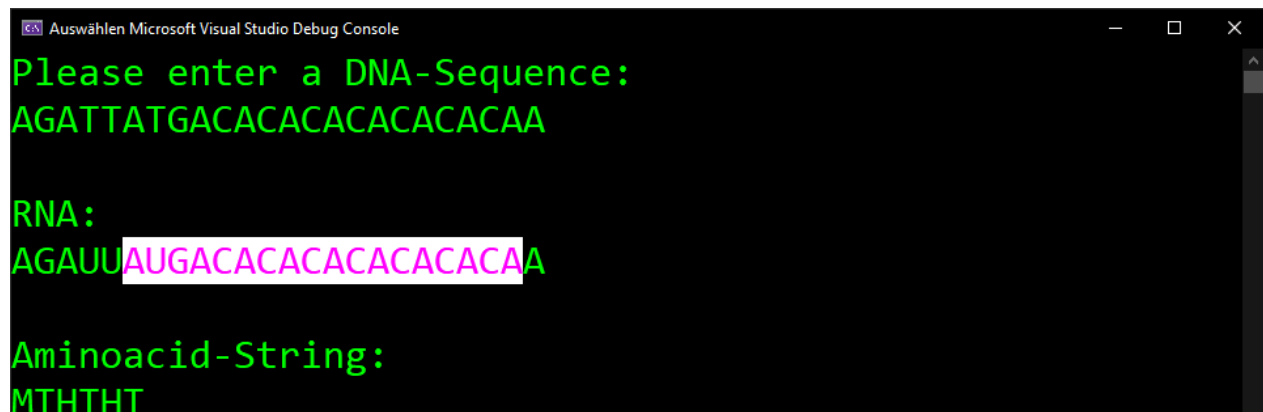


```
Auswählen Microsoft Visual Studio Debug Console

Please enter a DNA-Sequence:
AGATTATGACACACACACACA

RNA:
AGAUUAUGACACACACACACA

Aminoacid-String:
MHTHT
```

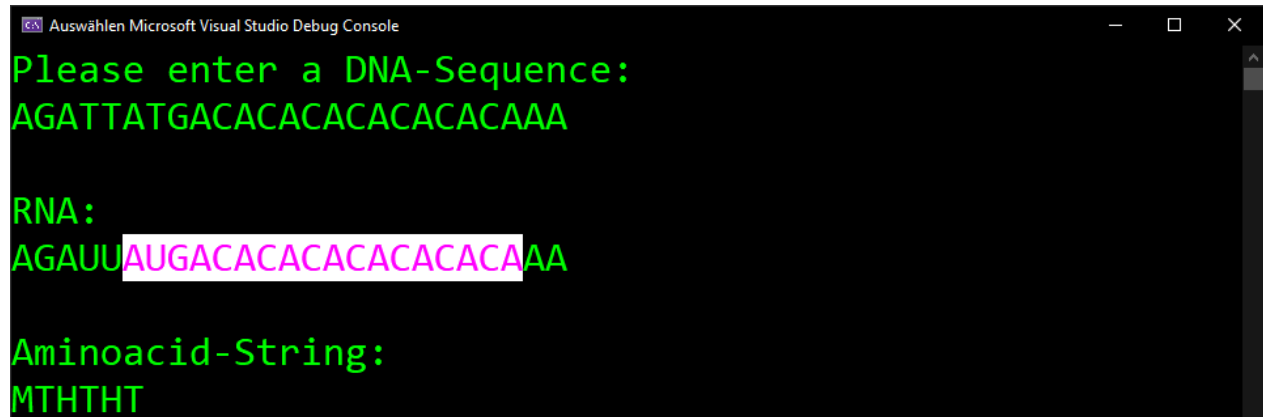


```
Auswählen Microsoft Visual Studio Debug Console

Please enter a DNA-Sequence:
AGATTATGACACACACACACAA

RNA:
AGAUUAUGACACACACACACAA

Aminoacid-String:
MHTHT
```



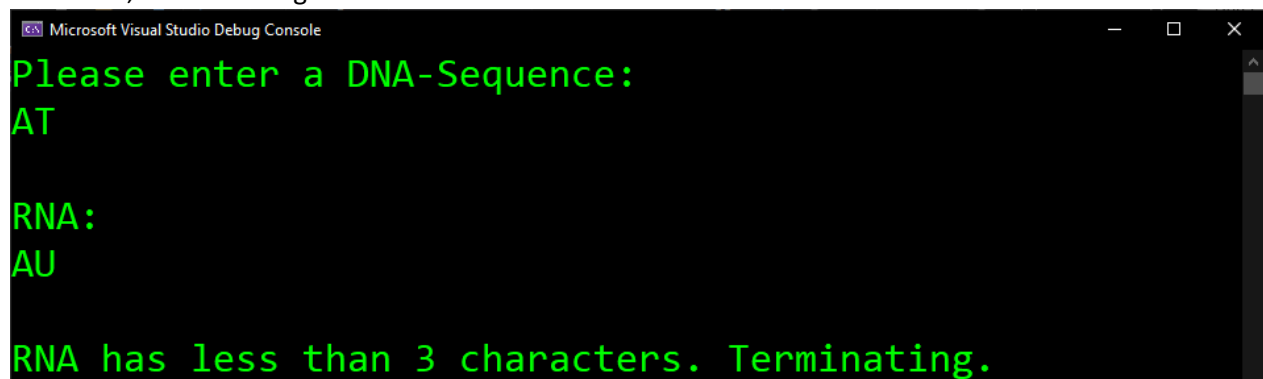
```
Auswählen Microsoft Visual Studio Debug Console

Please enter a DNA-Sequence:
AGATTATGACACACACACACAAA

RNA:
AGAUUAUGACACACACACACAAA

Aminoacid-String:
MTHTHT
```

Nicht leer, aber zu wenig für Translation:



```
Microsoft Visual Studio Debug Console

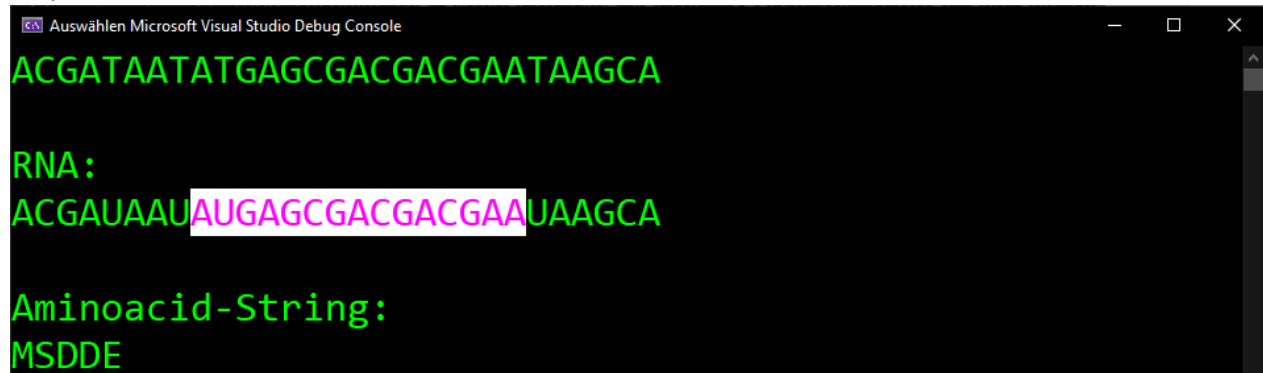
Please enter a DNA-Sequence:
AT

RNA:
AU

RNA has less than 3 characters. Terminating.
```

Beliebig viele Zeichen, begrenzt durch Stop-Codon:

Stop-Codon „UAA“:



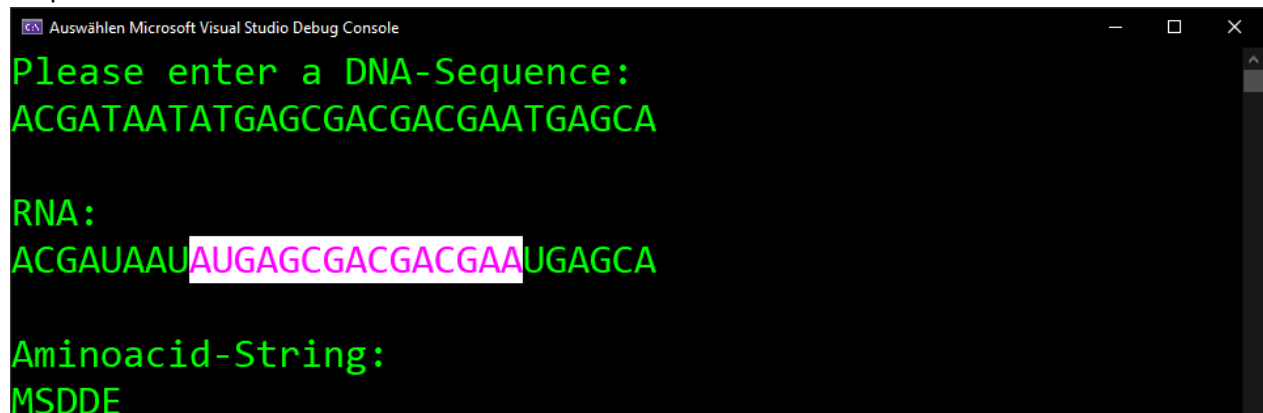
```
Auswählen Microsoft Visual Studio Debug Console

ACGATAATATGAGCGACGACGAATAAGCA

RNA:
ACGAUAAUAUGAGCGACGACGAAUAAGCA

Aminoacid-String:
MSDDE
```

Stop-Codon “UGA”:



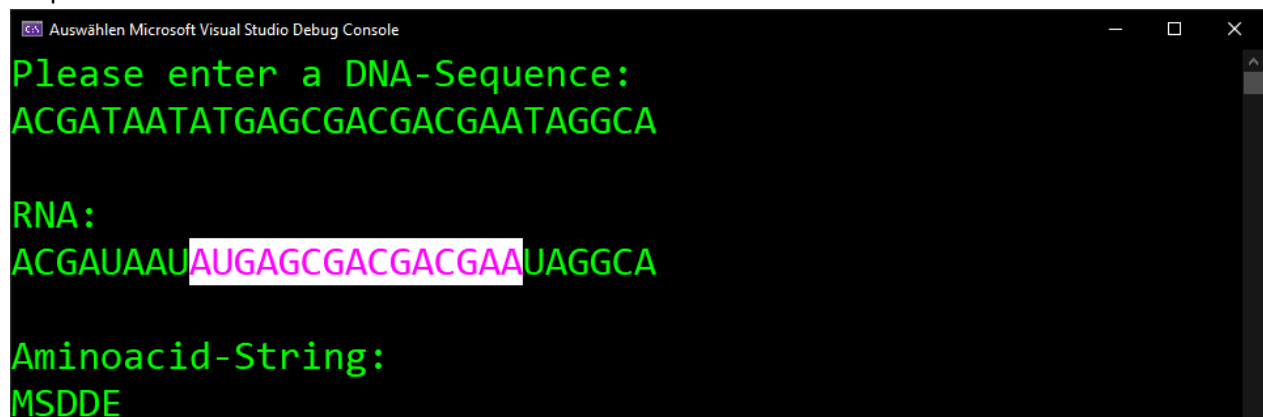
```
Auswählen Microsoft Visual Studio Debug Console

Please enter a DNA-Sequence:
ACGATAATATGAGCGACGACGAATGAGCA

RNA:
ACGAUAAUAUGAGCGACGACGAAUGAGCA

Aminoacid-String:
MSDDE
```

Stop-Codon “UAG”:



```
Auswählen Microsoft Visual Studio Debug Console

Please enter a DNA-Sequence:
ACGATAATATGAGCGACGACGAATAGGCA

RNA:
ACGAUAAUAUGAGCGACGACGAAUAGGCA

Aminoacid-String:
MSDDE
```

Kleinbuchstaben:

```

Auswählen Microsoft Visual Studio Debug Console

Please enter a DNA-Sequence:
atgacgatcacgatagcatcga

RNA:
AUGACGAUCACGAUAGCAUCGA

Aminoacid-String:
MTITIAS
  
```

Referenzbeispiel:

```

Auswählen Microsoft Visual Studio Debug Console

Please enter a DNA-Sequence:
CACAGACTCAGAGAGAACCCACCATGGTGCTGTCTCCTGCCGACAAGACCAACGTCAAGGCCGCTGGGGTAAGGTCGGCGCGCACGCTGGCGAGTATGGTGCGGAGGCC
TGGAGAGGATGTTCTGTCTTCCCCACCACCAAGACCTACTTCCCGCATTTCGACCTGAGCCACGGCTCTGCCAGGTTAAGGGCCACGGCAAGAAGGTGGCCGACGCGC
TGACCAACGCCGTGGCGCACGTGGACGACATGCCCAACGCGCTGTCCGCCCTGAGCGACCTGCACGCGCACAAAGTTTCGGGTGGACCCGGTCAACTTCAAGCTCCTAAGCC
ACTGCTGTGTTGACCTGGCGCCACCTCCCCGCCGAGTTACCCCTGCGGTGCACGCTCCCTGGACAAGTTCCTGGCTTCTGTGAGCACCGTGCTGACCTCCAAAT
ACCGTTAAGCTGGAGCCTCGGTAGCCGTTCCTCCTGCCGCTGGGCCCTCCCAACGGGCCCTCCTCCCTCCTTGAC

RNA:
CACAGACUCAGAGAGAACCCACC AUGGUGCUGUCUCCUGCCGACAAGACCAACGUCAAGGCCGCCUGGGGUAAGGUCGGCGCGCACGUGGCAGUAUGGUGCGGAGGCC
UGGAGAGGAUGUCCUGUCCUJCCCCACCACCAAGACCUACUJCCGCACUJCGACCUAGGCCACGGCUCUGCCAGGUUAAGGGCCACGGCAAGAAGGUGGCCGACGCGC
UGACCAACGCCGUGGCGCACGUGGACGACAUGCCCAACGCGCUGUCGCCUUGAGCGACCUACGCGCACAAAGCUUCGGGUGGACCCGGUCAAUUAAGCUCCUAAGCC
ACUGCCUGCUGGUGACCUUGGCCGCCACCUCCCGCCGAGUUCACCCUGCGGUGACGCUCCUUGGACAAGUUCUGGCUCUCUGUGAGCACCGUGCUGACCUCAAU
ACCGUUAAGCUGGAGCCUCGGUAGCCGUUCCUCCUGCCCGCUGGGCCUCCCAACGGGCCUCCUCCCUCCUUGCAC

Aminoacid-String:
MVLSPADKTNVKAANGKVGAHAGEYGAELERMFLSFPTTKTYFPHFDLSHGSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKLRVDPVNFKLLSHCLLVTLA
AHLPAEFTPAVHASLDKFLASVSTVLTSKYR
  
```

Teil 2 – Primfaktorenzerlegung

Ziel ist die Entwicklung eines C++ Programms, welches Zahlen einliest und diese in auf- und absteigender Reihenfolge in ihre Primfaktoren zerlegen kann und ausgibt.

Lösungsidee:

Die Zahl muss vom User eingelesen werden und somit auch auf Fehler überprüft werden. Bei numerischen Datentypen ist das noch dazu sehr heikel, weil falsche Eingaben, wie z.B. Strings, Fehler im Konsolen-Inputstream erzeugen können. Zusätzlich dazu, soll der User auch entscheiden ob die Faktoren auf- oder absteigend gelistet werden, was auch auf der Konsole eingelesen wird.

Theoretisch könnte der Nutzer jede beliebige Zahl eingeben, Datentypen erlauben aber nur eine gewisse Größe. Ich würde mich aber NICHT dazu entscheiden, den maximal größten Wert zuzulassen und einen signed Integer verwenden. Die Primzahlenfaktorisierung kann bei vielen Stellen nämlich, je nach Wert, sehr lange dauern. Das wird sogar bei manchen Technologien ausgenutzt (Verschlüsselung), ist aber in diesem Fall eher unerwünscht und selbst im Wertebereich eines signed Integers schon mehr als bemerkbar.

Die Faktorisierung läuft dann immer nach folgendem Prinzip ab:

1. Ist die Zahl bereits eine Primzahl, so kann nicht weiterzerlegt werden -> Zahl ausgeben. Ende.
2. Ist die Zahl keine Primzahl, so wird mit jeder Primzahl zwischen 2 und der *Hälfte des eingegebenen Wertes auf Teilbarkeit überprüft.
3. Ist die Teilbarkeit gegeben, so wird der Teiler ausgegeben und die Zahl dadurch dividiert. Es wird wieder ab Schritt 1 wiederholt.

Wir müssen mit Primzahlen nur bis zur Hälfte des Wertes auf Teilbarkeit überprüfen, da dies der letzte Wert ist, bei dem ein Wert ≥ 2 resultieren kann. Der nächste ganzzahlig teilbare Wert wäre erst die Zahl selbst und diese würde bei Teilung „1“ ergeben und 1 ist keine Primzahl. Das würde nur in unnötigem Rechenaufwand resultieren.

Dieses Prinzip lässt sich auch bei der Funktion „is_prime()“ einsetzen. Wir überprüfen für einen gegebenen Wert ob es Zahlen gibt, die größer gleich 2 und kleiner gleich Hälfte des Wertes sind und gleichzeitig die Zahl teilen. Wird die gegebene Zahl durch keine Zahl in diesem Bereich geteilt, so ist sie Prim.

Aufsteigende Reihenfolge erreicht man durch Setzen des Divisors auf den niedrigsten Anfangswert (2) und kontinuierlichen Erhöhen bis Teilbarkeit erreicht ist. Bei absteigender Reihenfolge wird der Divisor auf den Höchsten Wert (Inputwert/2) gesetzt und kontinuierlich verringert.

Es wird mit I/O und Strings gearbeitet. Daraus resultiert, dass folgende Bibliotheken benötigt werden: iostream, string.

Lösung:

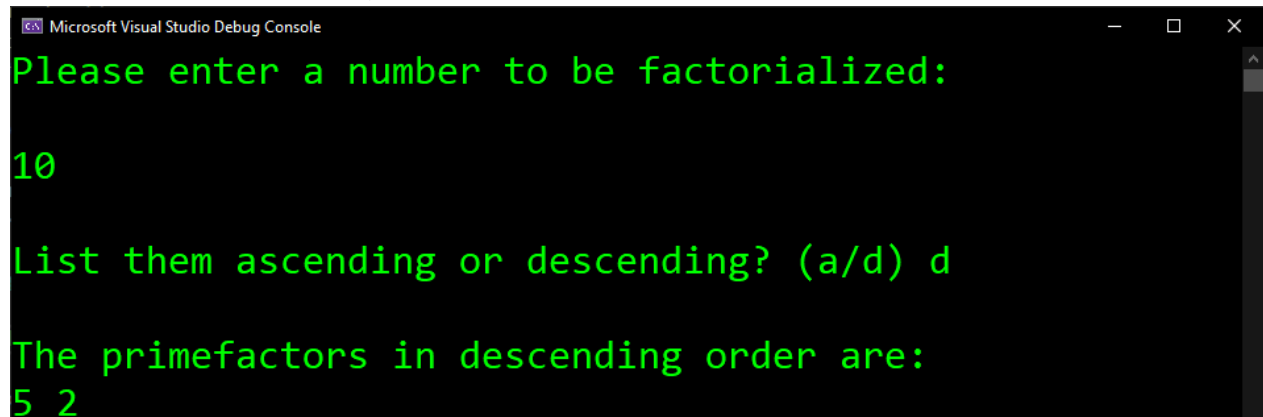
Als C++ Projekt „Teil_2“ im Archiv

Testfälle:

Fehlerhafte Eingaben:

- Leere Eingaben
- Zahl kleiner gleich 1
- Falscher Datentyp
- Zu große Zahl

Leere Eingaben (bei Kombination von „cin“ mit Integern erzeugt dies nur einen sichtbaren Zeilenumbruch in der Konsole):



```
Microsoft Visual Studio Debug Console

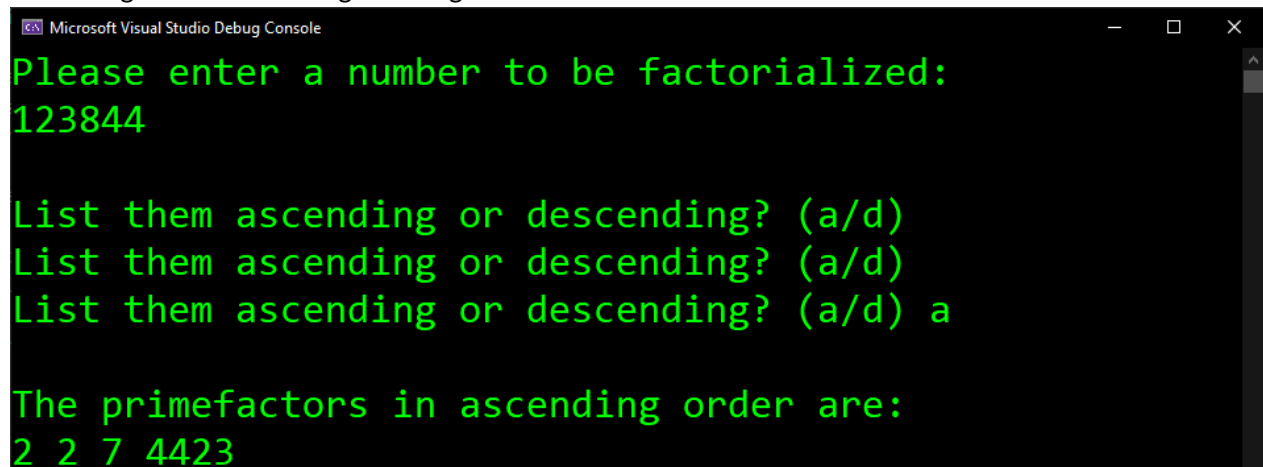
Please enter a number to be factorialized:

10

List them ascending or descending? (a/d) d

The primefactors in descending order are:
5 2
```

Leere Eingabe der Auflistungsrichtung:



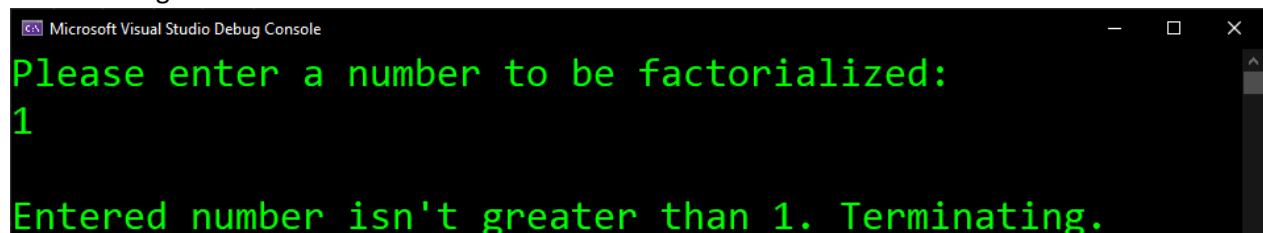
```
Microsoft Visual Studio Debug Console

Please enter a number to be factorialized:
123844

List them ascending or descending? (a/d)
List them ascending or descending? (a/d)
List them ascending or descending? (a/d) a

The primefactors in ascending order are:
2 2 7 4423
```

Zahl kleiner gleich 1:



```
Microsoft Visual Studio Debug Console

Please enter a number to be factorialized:
1

Entered number isn't greater than 1. Terminating.
```

Falsche Datentypen (String bei Zahl, andere Werte als „a“ und „d“ bei der Richtung):

```
Microsoft Visual Studio Debug Console

Please enter a number to be factorialized:
asjdsalfsa1000
The value must be numeric (MAX:2147483647): ,100
The value must be numeric (MAX:2147483647): .200
The value must be numeric (MAX:2147483647): Nein
The value must be numeric (MAX:2147483647): 210000000

List them ascending or descending? (a/d) Nein
List them ascending or descending? (a/d) 29
List them ascending or descending? (a/d) a

The primefactors in ascending order are:
2 2 2 2 2 2 2 3 5 5 5 5 5 5 7
```

Der nächste Fall ist von mir mit momentanen Mitteln noch nicht besser behandelbar, aber er führt zu keinem Absturz.

Zahl mit nachfolgendem String bei Zahl-Input:

```
Microsoft Visual Studio Debug Console

Please enter a number to be factorialized:
1238593sinnloser_string

List them ascending or descending? (a/d) a

The primefactors in ascending order are:
97 113 113
```

Zu große Zahl (größer 2^{31} bzw. ca. 2.15 Mrd):

```
Microsoft Visual Studio Debug Console

Please enter a number to be factorialized:
2300000000
The value must be numeric (MAX:2147483647): 2100000000

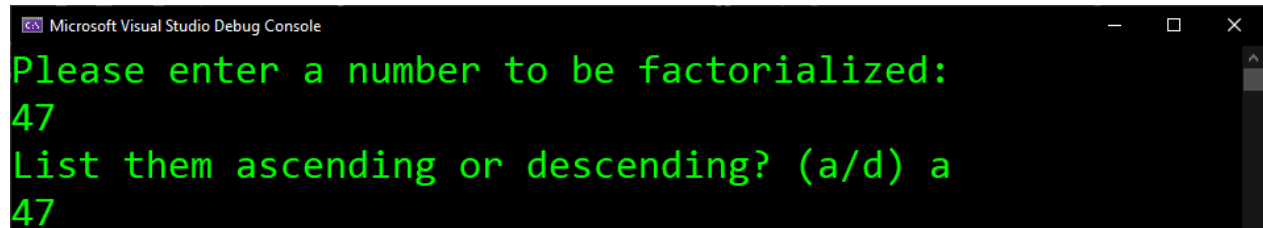
List them ascending or descending? (a/d) a

The primefactors in ascending order are:
2 2 2 2 2 2 2 2 3 5 5 5 5 5 5 5 7
```

Valide Eingaben:

- Primzahlen
- Keine Primzahlen, aufwärts sortiert
- Keine Primzahlen, abwärts abwärts

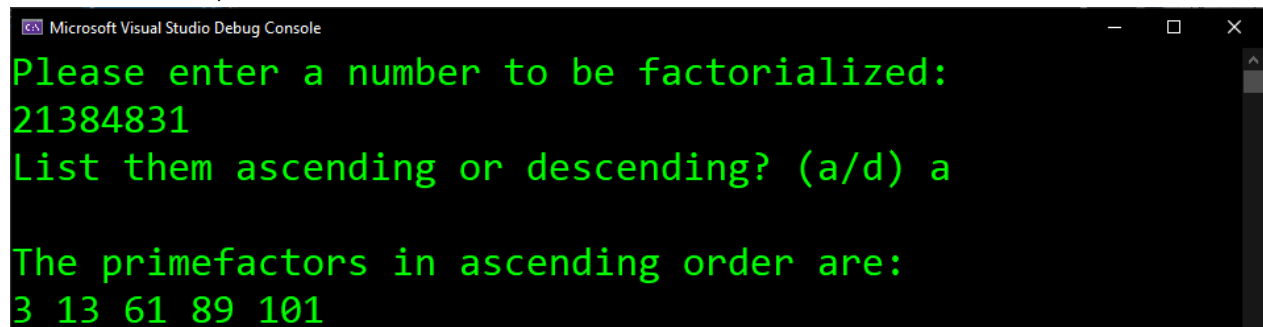
Primzahlen:



```
Microsoft Visual Studio Debug Console

Please enter a number to be factorialized:
47
List them ascending or descending? (a/d) a
47
```

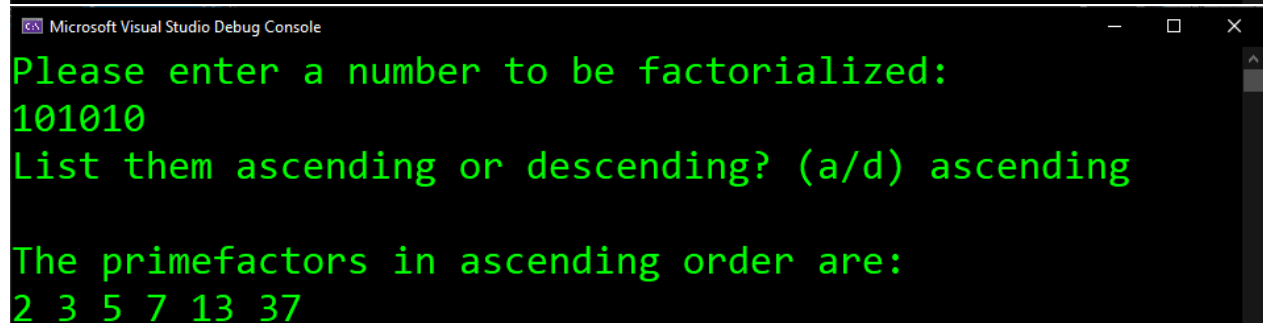
Keine Primzahlen, aufwärts sortiert:



```
Microsoft Visual Studio Debug Console

Please enter a number to be factorialized:
21384831
List them ascending or descending? (a/d) a

The primefactors in ascending order are:
3 13 61 89 101
```

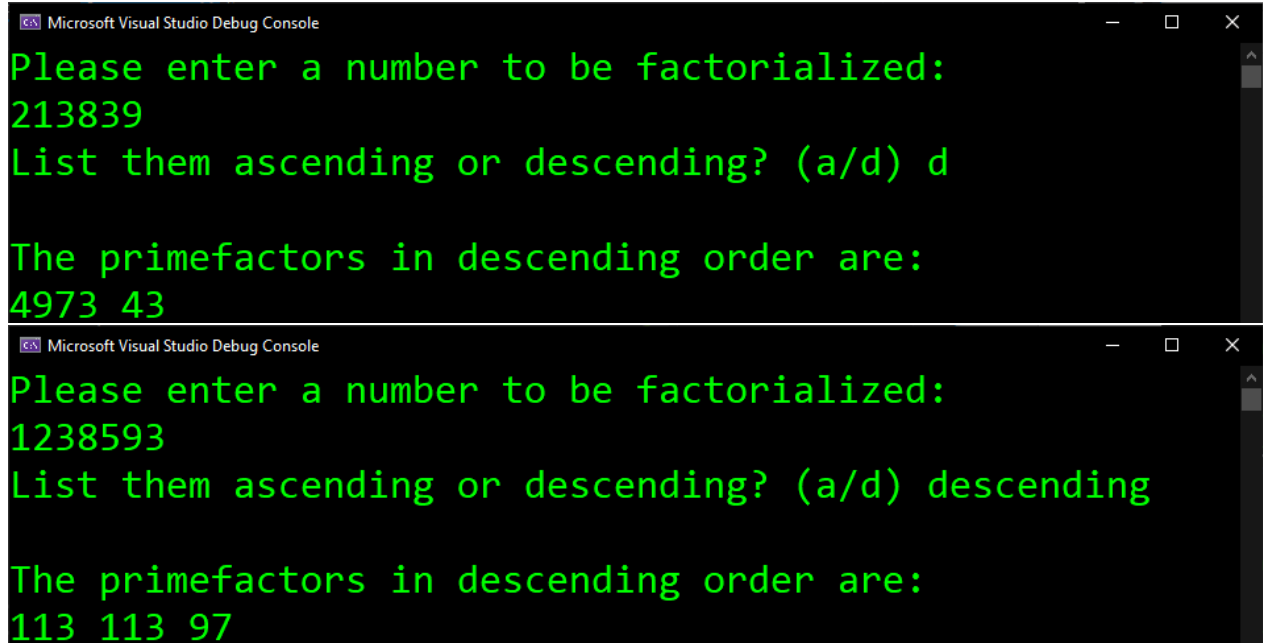


```
Microsoft Visual Studio Debug Console

Please enter a number to be factorialized:
101010
List them ascending or descending? (a/d) ascending

The primefactors in ascending order are:
2 3 5 7 13 37
```

Keine Primzahlen, abwärts sortiert:



The image shows two screenshots of the Microsoft Visual Studio Debug Console. The top screenshot shows the input '213839' and the output '4973 43'. The bottom screenshot shows the input '1238593' and the output '113 113 97'. Both screenshots show the prompt 'Please enter a number to be factorialized:' and the question 'List them ascending or descending? (a/d) d'.

```
Microsoft Visual Studio Debug Console
Please enter a number to be factorialized:
213839
List them ascending or descending? (a/d) d

The primefactors in descending order are:
4973 43

Microsoft Visual Studio Debug Console
Please enter a number to be factorialized:
1238593
List them ascending or descending? (a/d) descending

The primefactors in descending order are:
113 113 97
```

Teil 3 – Punktmatrix

Ziel der Übung ist das Erstellen von Punktmatrizen basierend auf zwei DNA-Sequenzen um deren Ähnlichkeit an gewissen Stellen zu überprüfen.

Lösungsidee:

Zu Beginn müssen zwei DNA-Sequenzen eingelesen werden, welche wiederum auf Validität überprüft werden müssen. Hierfür kann wieder die Funktion aus UE4 verwendet werden. Der Algorithmus könnte zwar für jeden String verwendet werden, aber in der Angabe wird von DNA-Sequenzen gesprochen, weshalb ich die Eingaben auf diese limitiere.

Anschließend kann ein 2-dimensionaler Vektor angelegt werden, welcher mit der Länge des 2. Strings initialisiert wird, um für jeden Buchstaben einen Vergleich zum anderen String zu haben. Die Werte von „0“ und „1“ könnten zwar, weil es nur diese 2 Zustände gibt, symbolisch durch einen Bool realisiert werden und somit Ressourcen gespart werden, aber in der Angabe werden die zu speichernden Werte von „0“ und „1“ fix angesprochen, weshalb ein Integer verwendet werden sollte.

Für jeden Buchstaben im 1. String wird dann überprüft, ob dieser gleich einem anderen Buchstaben ist. Das geschieht mithilfe von 2 For-Schleifen, welche gemeinsam Zeile und Spalte der Matrix durchiterieren. Sind die Buchstaben gleich, so wird eine „1“ an die momentane Zeile, welche selbst ein Vektor ist, angehängt. Sind diese ungleich, wird eine „0“ angehängt.

Für den strikten Modus muss die Anzahl an Iterationen in den Schleifen um 1 verringert werden, weil wir sonst mit „Zeile+1“ und „Spalte+1“ einen Wert außerhalb der Matrix abfragen würden. Das Prinzip ist gleich zur normalen Punktmatrix, nur dass nun zusätzlich zum aktuellen Zeichen auch das Zeichen in der nächsten Reihe und Spalte zwischen den DNA-Sequenzen übereinstimmen muss. Durch die verringerte Anzahl an Iterationen sollten die letzte Zeile und Spalte leer bleiben. Da die Größe aber gleichbleiben soll, müssen wir diese Werte zum Schluss mit „0“ auffüllen, da diese sowieso immer „0“ sind, da es entweder keine nächste Spalte oder keine nächste Zeile gibt.

Für die Ausgabe muss zuerst die Spaltenbeschreibung (der erste String) geschrieben werden und danach wiederum in zwei For-Schleifen über die ganze Matrix zuerst der momentane Buchstabe aus dem zweiten String und danach alle Werte aus der momentanen Zeile der Matrix, wobei die „0“ zu einem Blank und eine „1“ zu einem „X“ wird. Bei der Formatierung muss darauf geachtet werden, dass die Zeilenwerte mit der Spaltenbeschreibung zusammenpassen.

Im Beispiel wird mit I/O, Vektoren und Stringfunktionen gearbeitet. Daraus ergeben sich folgende Bibliotheken: `iostream`, `vector`, `string`.

Lösung:

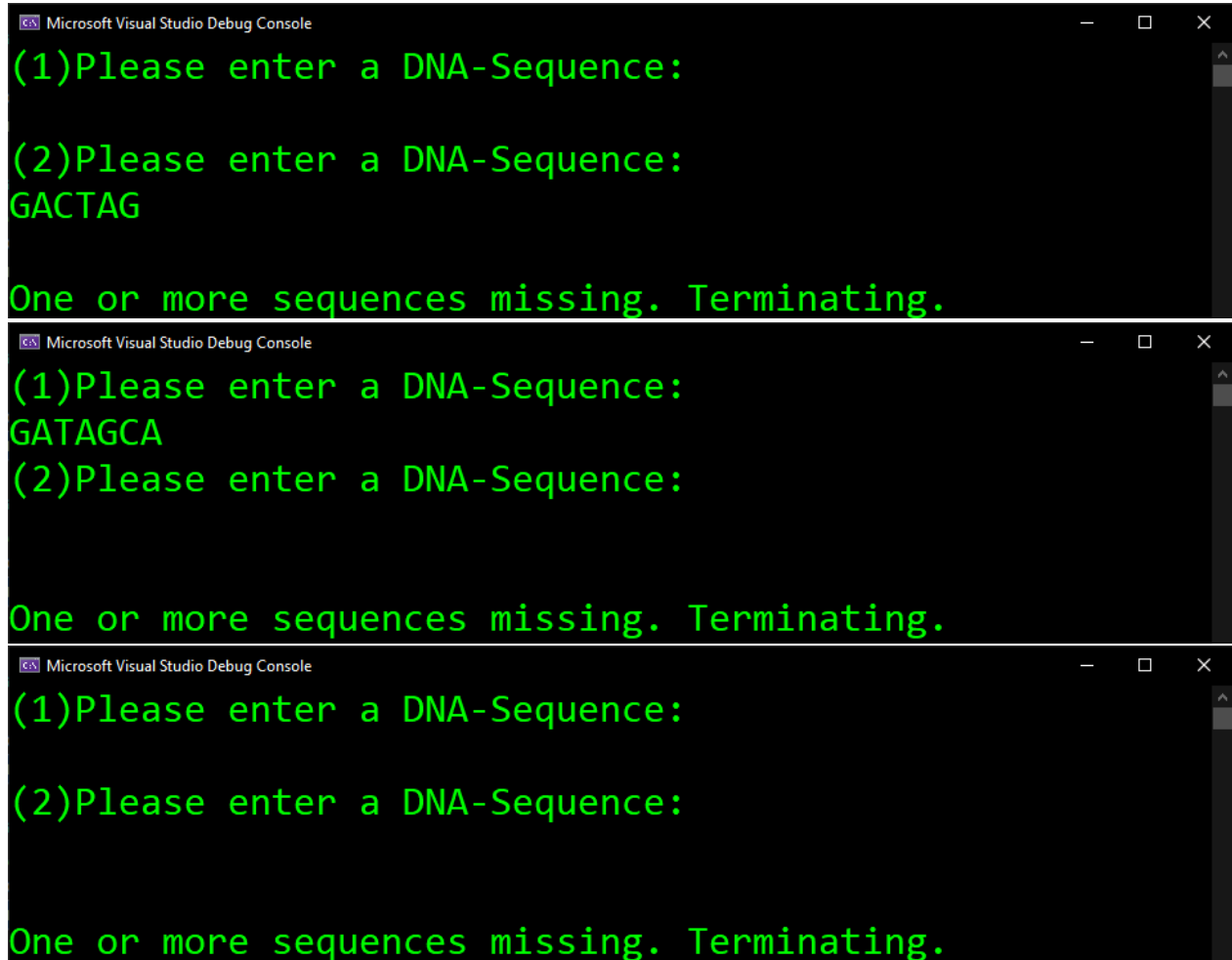
Als C++ Projekt „Teil_3“ im Archiv

Testfälle:

Fehlerhafte Eingaben:

- Mind. 1 Eingabe leer
- Falsche Zeichen

Mind. 1 Eingabe leer:



```
Microsoft Visual Studio Debug Console
(1)Please enter a DNA-Sequence:
(2)Please enter a DNA-Sequence:
GACTAG
One or more sequences missing. Terminating.

Microsoft Visual Studio Debug Console
(1)Please enter a DNA-Sequence:
GATAGCA
(2)Please enter a DNA-Sequence:
One or more sequences missing. Terminating.

Microsoft Visual Studio Debug Console
(1)Please enter a DNA-Sequence:
(2)Please enter a DNA-Sequence:
One or more sequences missing. Terminating.
```

Falsche Zeichen:

```
Microsoft Visual Studio Debug Console
(1)Please enter a DNA-Sequence:
DAS IST EIN STRING!
Invalid DNA-Sequence. Please re-enter: AGCT AGAT
Invalid DNA-Sequence. Please re-enter: AGCT
(2)Please enter a DNA-Sequence:
1000
Invalid DNA-Sequence. Please re-enter: CGTA
```

Normal dot matrix:

	A	G	C	T
C			X	
G		X		
T				X
A	X			

Strict dot matrix:

	A	G	C	T
C				
G				
T				
A				

Valide Eingaben:

- Keine Gleichheiten
- Klein- und Großbuchstaben
- Referenzbeispiel

Keine Gleichheiten:

```
Microsoft Visual Studio Debug Console

(1)Please enter a DNA-Sequence:
AAA
(2)Please enter a DNA-Sequence:
CCC

Normal dot matrix:
  | A | A | A |
-----
C |   |   |   |
C |   |   |   |
C |   |   |   |

Strict dot matrix:
  | A | A | A |
-----
C |   |   |   |
C |   |   |   |
C |   |   |   |
```


Groß- und Kleinschreibung:

Microsoft Visual Studio Debug Console

(1)Please enter a DNA-Sequence:

acgt

(2)Please enter a DNA-Sequence:

ACGT

Normal dot matrix:

	A	C	G	T
A	X			
C		X		
G			X	
T				X

Strict dot matrix:

	A	C	G	T
A	X			
C		X		
G			X	
T				

Microsoft Visual Studio Debug Console

GATTCGCTTAGT

CTGATTCCTTAGTCAG

	G	A	T	T	C	G	C	T	T	A	G	T
C					X		X					
T			X	X				X	X			X
G	X					X					X	
A		X								X		
T			X	X				X	X			X
T			X	X				X	X			X
C					X		X					
C					X		X					
T			X	X				X	X			X
T			X	X				X	X			X
A		X								X		
G	X					X					X	
T			X	X				X	X			X
C					X		X					
A		X								X		
G	X					X					X	

[illegible]