Übungsserie 6

Zykel und Felder

Seminar in KW 46-48

Der Datentyp std::string

Variablen vom Datentyp char werden wie üblich deklariert und initialisiert. Das Literal für den Datentyp ist ein Zeichen in einfachen Anführungszeichen oder eine Zahl im Wertebereich von -128 bis 127.

```
1 char zeichen1  = 'A';                          // Ein Zeichen mit dem Wert des
      Buchstabens A initialisiert
2 char zeichen2  = 65;                           // Ein Zeichen mit dem Wert der Zahl 65
      initialisiert
3 bool istGleich = ( zeichen1 == zeichen2 ); // istGleich hat den Wert 'true', da 'A'
      == 65 ist.
4
5 char zeichen   = '\n';                         // initialisiert zeichen mit dem Newline-
      Zeichen. (Wert 10)
6 char space     = ' ';                          // initialisiert space mit dem
      Leerzeichen (Wert 32)
```

Der Datentyp std::string ist in dem Header #include <string> definiert. Er bietet Manipulationsmöglichkeiten für Zeichenketten, das heißt, Felder aus Buchstaben des Typs char. Im Gegensatz zu char, int, float, double, ... ist std::string kein eingebauter C++-Datentyp sondern in der Standardbibliothek definiert und implementiert.[1]

Eine Variable des Typs std::string wird wie üblich deklariert und initialisiert, wobei das Literal für eine Zeichenkette durch doppelte Anführungszeichen gekennzeichnet ist. Unter C++14 gibt es den Suffix „s“ um Stringliterale explizit zu benennen.

```
1 std::string name = "Mario Hlawitschka"; //< C++ in allen Versionen
2
3 using namespace std::string_literals;
4 std::string fakultaet = "IMN"s;          //< erst ab Version C++14
```

Zeichenketten vom Typ std::string lassen sich wie üblich einlesen und ausgeben, wobei beim Einlesen nur bis zum ersten Leerzeichen gelesen wird:

```
1 std::cin  >> name; //< Bei Eingabe von "Mario Hlawitschka" wird nur "Mario"
      eingelesen.
2 std::cout << name; //< Gibt den Namen auf dem Terminal aus.
```

Eine ganze Zeile kann mit der Funktion std::getline aus <string> eingelesen werden:

```
1 std::string zeile;
2 std::getline( std::cin, zeile ); //< liest von std::cin eine Zeile ein und speichert
       das Ergebnis in der Variablen zeile
```

```
1 std::string meineZeichenkette = "Hallo Freunde";
2
3 // Methode: size_type size() oder size_type length()
4 std::cout << meineZeichenkette.length();          //< gibt die Länge der
      Zeichenkette in der Anzahl an Zeichen aus
5 // Methode: char& at( size_type position )
```

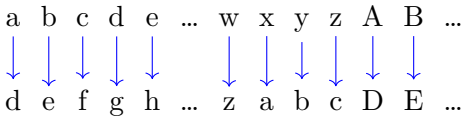
[1] Das gleiche gilt übrigens auch für std::istream, dem Datentyp der Variablen std::cin und std::ostream, dem Datentyp der Variablen std::cout.

```
6 std::cout << meineZeichenkette.at( 2 );          //< gibt das *dritte* Zeichen in
      der Zeichenkette aus
```

1 Caesar-„Verschlüsselung“

Die als Caesar-Verschlüsselung bekannte Technik, Nachrichten unlesbar zu machen, basiert darauf, dass Zeichen in einem vorgegebenen Alphabet vertauscht werden. Im konkreten Fall geht es darum, das Alphabet auf ein um N Zeichen verschobenes Alphabet abzubilden.

Schreiben Sie eine Funktion, die einen Text verschlüsselt. Übergeben Sie der Funktion den Eingabetext als std::string und die Verschiebung der Buchstaben als Schlüssel vom Typ int. Die Verschiebung soll dabei bei dem Schlüssel 3 die Buchstaben wie folgt abbilden:



Alle anderen Zeichen, insbesondere Zahlen und Sonderzeichen, sollen nicht verändert werden. Der Schlüssel soll in dem Programm bei der Ausführung frei wählbar sein.

Hinweis: Der Datentyp char ist intern ein Ganzzahldatentyp. Mit ihm kann man auch rechnen.

2 Histogramme

Rotationsverfahren (wie auch Permutationsverfahren, zu denen Rotationsverfahren zählen) bei der Verschlüsselung sind unsicher und einfach zu knacken. Das zeigt sich vor allem darin, dass die Häufigkeit der Zeichen nicht verändert wird, sondern lediglich auf andere Zeichen abgebildet wird.

Schreiben Sie deshalb ein Programm, das die Häufigkeit von Zeichen bestimmt und diese „graphisch ausgibt“ Die Ausgabe soll dabei (nach geeigneter Skalierung für die Terminalbreite, z.B. maximal 60 Zeichen) in Form von Balkendiagrammen wie folgt erfolgten:

```
1 a: ####### (3.5%)
2 b: ############ (4.0%)
3 c: #### (2.0%)
4 d: ############################## (12.0%)
```

Da C++ nativ keine Funktion zum Einlesen einer ganzen Datei bereitstellt, verwenden Sie dafür folgendes Codefragment.

Quelltext zum Einlesen einer Datei in einen String

```
1  #include <streambuf>
2  #include <fstream>
3
4  std::string leseGanzeDatei( const std::string& dateiname )
5  {
6      std::ifstream ifs( dateiname );
7      return std::string((std::istreambuf_iterator<char>(ifs)),
8                         (std::istreambuf_iterator<char>()));
9  }
10
11 // oder
12
13 std::string leseGanzeDatei2( const std::string& dateiname )
```

```
14 {
15     std::ifstream ifs( dateiname );
16 
17     std::string text;
18     while( ifs )
19     {
20         std::string temp;
21         std::getline( ifs, temp );
22         text.append( temp );
23     }
24     return text;
25 }
```

Schrieben Sie dazu eine Funktion zum Einlesen einer Textdatei, eine Funktion zum Berechnen des Histogramms aller Buchstaben und Zahlen und eine Funktion zur graphischen Ausgabe des Ergebnisses.

Entschlüsselung (Optional)

- Nutzen Sie das Programm aus Aufgabe 2, um einen Schlüssel für die Datei rot.txt im Opal zu erraten. (Hinweis: Im Deutschen ist „e“ mit ca. 17,40% der häufigste Buchstabe, gefolgt von „n“ mit 9,78% und I mit 7,55%)
- Modifizieren Sie das Programm aus Aufgabe 1 so, dass es eine Datei als Eingabe liest.
- Entschlüsseln Sie den Text mit dem modifizierten Programm aus Aufgabe 1.