Übungsblatt 4 20.09.2023

Für diese Aufgaben benötigt ihr die Dateien aus dem Archiv `ueb4.zip`.

Aufgabe 1

Gegeben sei:

```
string s;
Point p;
```

Gebt folgende Deklarationen an:

a) Eine Referenz `r` auf `s`,

b) eine konstante Referenz `cr` auf `p`,

c) ein Array `a` aus 20 vorzeichenlosen Integer-Zahlen,

d) einen Pointer `pp` auf einen `char`-Pointer,

e) einen `vector` aus einzelnen Zeichen (`char`),

f) eine `map` mit Paaren `(Point, double)`,

g) eine `list` aus `unique_ptr<string>`.

Aufgabe 2

Gebt an, wieviel Speicherplatz vom Compiler für folgende Deklarationen auf einem ILP32-System vorgesehen wird.

a) `signed int *b`

b) `char **d`

c) `long e[8]`

d) `short *f[12]`

e) `char j, *k`

f) `char l[] = "hallo"`

g) `const char *s = "welt"`

Aufgabe 3

In `aufgabe_3/rng.cc` findet ihr ein einfaches Programm, in dem ein Vektor mit Zahlenwerten initialisiert werden soll. Implementiert den Operator `Data::operator()` so, dass mit jedem

Aufruf ein neuer Zufallswert im Intervall [0,1] zurückgeliefert wird. Als Zufallszahlengenerator soll die *Mersenne Twister Engine* `mt19937` eingesetzt werden.

Gebt die erzeugten Zufallszahlen auf `cout` aus.

Aufgabe 4

In `aufgabe_4/b64dec.cc` findet ihr ein Gerüst für einen Base64-Dekoder.[1]

Base64 ist eine Kodierung, die jeweils drei aufeinander folgende 8-Bit-Zeichen in vier aufeinander folgende 6-Bit-Sequenzen einteilt. Jede dieser 6-Bit-Sequenzen wird daraufhin mittels eines vorgegebenen Alphabets auf 8-Bit-Zeichen abgebildet. Ist die Anzahl der Eingabezeichen kein Vielfaches von 3, so wird die letzte Dreiergruppe beim Kodieren mit Nullbits aufgefüllt. Zum Zeichen, dass aufgefüllt wurde, fügt der Kodierer dort festgelegte Padding-Symbole ein (in dem gewählten Alphabet ist dies das Zeichen „=“).

Eure Aufgabe ist es nun, die Template-Funktion `decode()` zu implementieren. Die Base64-kodierten Eingabedaten im Intervall [`first`, `last`) sollen dekodiert und das Ergebnis dem Iterator `it` zugewiesen werden. Beim Auftreten des ersten Padding-Zeichens kann die Dekodierung beendet werden (natürlich erst, wenn die letzte angefangene Bitfolge mit Nullbits aufgefüllt und an `it` übergeben wurde.) Zeichen, die nicht in dem übergebenen Alphabet enthalten sind, werden einfach übersprungen (z. B. Newlines).

Im Ordner `aufgabe_4` befinden sich drei Testdateien, die zu sinnvollem Text dekodiert werden können.

Zusatzaufgabe: Wer mag, kann sich ja einmal an der Funktion `encode()` versuchen. Ihr könnt sie im Prinzip genauso aufbauen wie `decode()`, nur dass nun noch das Padding am Ende erzeugt werden muss. Schreibt euch ggf. noch eine Funktion `enc()`, die zu einem (8-Bit-)Werte im Alphabet mit `std::find_if()` den zugehörigen 6-Bit-Schlüssel finden kann. Die Signatur für `enc()` sieht also folgendermaßen aus:

```
Alphabet::key_type enc(Alphabet::mapped_type c);
```

[1] Siehe `https://en.wikipedia.org/wiki/Base64`.