4.2 Enumerationen

Sowohl C als auch C++ bieten die Möglichkeit an, Enumerationen mit Hilfe des Schlüsselwortes `enum` zu definieren. Im Gegensatz zu Java sind die Elemente einer Enumeration aber einfach nur Zahlen, es gibt weder die Möglichkeit, einen `enum`-Wert durch eingebaute Funktionen in seinen Text zu verwandeln noch die Möglichkeit, eine Zeichenkette in den zugeordneten `enum`-Wert zu konvertieren. Ein weiterer Unterschied liegt darin, dass in C und C++ die numerischen Werte der `enum`-Elemente beliebig sein dürfen, solange sie ganzzahlig und eindeutig sind:

```
enum Verpackung {Dutzend = , Gros = , Palette = };
```

Werden keine expliziten Werte angegeben, so startet die Zählung per Definition mit `0` und das nächste Element der Enumeration bekommt jeweils den Wert des vorigen Elementes plus `1`.

4.3 Bitfelder

Sowohl C als auch C++ sind (auch) für die Entwicklung von hardwarenaher Software geeignet. Bei dieser Art von Programmen sind oft bitweise angeordnete Datenstrukturen zu bearbeiten. Ein gutes Beispiel hierfür sind die ATMega-Prozessoren von Atmel, bei dem es ein Register zur Steuerung der Energiesparmodi gibt. Dieses 8 Bit breite Register mit dem Namen `MCUCR` enthält die folgenden Steuerbits:

`SE` Das höchstwertige `SE`-Bit steuert, ob der Prozessor in einen Energiesparmodus wechseln kann

`SM2, SM1, SM0` Die drei folgenden `SM`-Bits steuern, welchen der 8 möglichen Energiesparmodi der Prozessor bei Bedarf aktiviert

`ISC11, ISC10, ISC01, ISC00` Diese niederwertigsten vier Bits, aufgeteilt in zwei Gruppen zu je zwei Bits steuern, aufgrund welcher Bedingungen ein externer Anschluss einen Interrupt verursacht, und damit den Prozessor aufweckt.

In Java wäre eine Programmierung des Energiesparmodus etwas umständlich, da man zunächst den alten Wert auslesen, die nicht `SM`-Bits unter Beibehaltung der anderen Bits auf 0 setzen müsste und dann genau die 3 `SM`-Bits auf den neuen Wert setzen müsste. Zur Einstellung des Energiesparmodus 5 hieße dies:

```
mcucr = mcucr & ~(7«4) | (5«4); // Bis Java 6
mcucr = mcucr & 0b10001111 | 0b01010000; // Ab Java 7
```

Unter C oder C++ lässt sich hierfür eine Struktur (ähnlich wie eine Klasse) definieren, bei der die einzelnen Elemente jeweils auf Bit-Grenzen genau plaziert werden können. Eine solche Definition wäre:

```
struct MCUCR {
    unsigned SE:1;
```

```
        unsigned SM:3;
        unsigned ISC1:2;
        unsigned ISC0:2;
    };
```

Die Anweisung zum Setzen des Energiesparmodus lautet dann einfach (nach Definition einer Variablen `mcucr` vom Typ `MCUCR`): `mcucr.SM=5;`

4.4 Zeichenketten

Eine Zeichenkette ist sowohl in C als auch in C++ ein Array von Zeichen, das durch das spezielle Nullzeichen (in C und C++ geschrieben als `'\0'`) terminiert wird. Damit sind Zeichenketten grundsätzlich änderbar. Jedoch werden Stringkonstanten von einigen Compilern in schreibgeschütztem Speicher angelegt, so dass es beim Schreibzugriff zu einem Speicherfehler kommt. Auch aus diesem Grund sollte die Nutzung von C-Strings in C++ auf diejenigen Fälle beschränkt bleiben, bei denen sie z. B. durch ein API erzwungen wird. In C gibt es leider keine standardisierten Alternativen, allerdings

4.4.1 Regeln für den Umgang mit Zeichenketten

Ansonsten kann man nur die folgenden Regeln empfehlen:

- Keiner Benutzereingabe trauen. Die Benutzung von Arrays mit einer wie auch immer definierten "sinnvollen" Länge ist der erste Schritt auf dem Weg zum Programmabsturz.
- Keine Strings mit `char *gets(char *c)` einlesen, stattdessen immer die Funktion `char *fgets(char *c, int size, FILE *fp)` nutzen. Eine lange Eingabezeile führt bei der Nutzung von `gets` zum Überschreiben des Puffers, bei `fgets` nicht. Zudem schneidet `gets` das Zeilenende ab, `fgets` nicht. So kann man auch erkennen, ob eine überlange Zeile gelesen wurde, indem man prüft, ob die Zeile mit einem Zeilenende endet.
- Beim Verketten von Zeichenketten weder `strncpy` noch `strncat` nutzen, sondern die "einfachen" Funktionen `strcpy` und `strcat`. Vorab ist zu prüfen, ob der Puffer noch lang genug für den anzuhängenden Teil ist. Die Funktionen `strncpy` und `strncat` haben zwei Probleme:
  - Wenn die maximal erlaubte Zahl an Zeichen kopiert bzw. angehängt wurde, wird kein `'\0'` mehr geschrieben, man erhält also statt eines Pufferüberlaufes einen String, der kein Ende hat bzw. dessen Ende durch die nächste zufällig im Speicher liegende `'\0'` bestimmt ist.
  - Die Zahl der Zeichen zu begrenzen ist bei `strncpy` durchaus sinnvoll, bei `strncat` dagegen nicht. Bietet eine Zeichenkette Platz für 1000 Zeichen und enthält gleichzeitig schon 500 Zeichen, müsste man als maximale zu kopierende Bytezahl 499 eingeben, und das `'\0'`-Zeichen noch manuell einfügen:

    ```
    strncpy(target, source, sizeof(target)-strlen(target)-1);
    target[sizeof(target)-1] = '\0';
    ```

In diesem Fall ist es sinnvoller, statt der Methode `strncpy` die Methode `memcpy` zu rufen und die Anzahl der zu kopierenden Zeichen selbst anzugeben.

4.4.2 Zeichenketten-Konstanten

Unter C und C++ sind Zeichenketten-Konstanten wie in Java gleichzeitig auch Strings, nur mit dem Unterschied, dass es keine Objekte, sondern durch `'\0'` terminierte Arrays von Zeichen sind. Aufgrund des Ursprungs beider Sprachen aus den Zeiten vor Unicode sind die Zeichenketten-Konstanten zunächst ASCII- bzw. ANSI-basiert. Seit den 90er Jahren gibt es jedoch auch in C und damit in C++ die Möglichkeit, Zeichenketten-Konstanten als Unicode zu definieren. Hierzu muss vor der Zeichenkette eines der folgenden Präfixe eingetragen werden:

- Das Präfix `L` erzeugt eine Unicode-Zeichenkette, bei der pro Zeichen 16 Bit Speicher verbraucht werden: `short *x = L"Ahren";`.
- Das Präfix `u8` erzeugt eine UTF8-codierte Zeichenkette: `char *x = u8"Ahren";`
- Das Präfix `u` erzeugt eine UTF16-codierte Zeichenkette: `short *x = u"Mohren";`.
- Das Präfix `U` erzeugt eine UTF32-codierte Zeichenkette: `int *x = U"Uberm utig";`.
- Das Präfix `R"delim(` erzeugt eine "rohe" Zeichenkette (nur unter C++), die beliebige Zeichen (auch Zeichenumbrüche) enthalten kann, und die durch die Folge `)delim"` beendet wird. Dieses Konstrukt ist besonders bei der Eingabe regulärer Ausdrücke hilfreich, weil der `\` nicht als Sonderzeichen gilt. Durch Voranstellen eines der vorgenannten Präfixe lässt sich die Codierung der Zeichenkette steuern.

Zu bedenken ist, dass unter C die Nutzung von UTF- oder Unicode-Zeichenketten die Nutzung dafür geeigneter Bibliotheksfunktionen verlangt. Ein spezielles Problem, besonders bei der Nutzung von UTF-Zeichenketten besteht darin, dass man nicht mehr präzise vorhersagen kann, wieviele Zeichen in einen gegebenen Speicherbereich passen.

Unter C++ gibt es jedoch auch eine objektorientierte Variante, die wir allerdings erst später behandeln werden.

4.5 Ausrichtung und Byte-Reihenfolge

Unter Java spielt die Ausrichtung eines Datentyps keine Rolle, es gibt einfach keine Möglichkeit auf die Daten eines Datentyps anders als über seine Elemente zuzugreifen. Unter C und C++ können Speicherbereiche aber mit generischen Funktionen (ähnlich `System.arraycopy` in Java) kopiert werden. Für diesen Fall brauchen die generischen Funktionen Informationen, wieviel Speicher kopiert werden muss. In der Regel (wir werden auch Ausnahmen sehen) wird diese Information zu einer Variablen über den Operator `sizeof` ermittelt.