1 Grundlagen

1.1 Zeichencodierung

Grundsätzlich sind Codierungen nötig, weil Zeichen in Computern als Binärwerte abgebildet werden. Eine Codierung ordnet über Zeichensatztabellen (=Codepage) jedem Binärwert ein Zeichen zu.

ASCII

Die Grundlage moderner Zeichenkodierungen stellt der 7-Bit Standard ASCII dar, der 128 druckbare Zeichen und Steuerzeichen definiert. Die Zeichencodierung erlaubt somit die gleichzeitige Darstellung nur weniger Sprachen im selben Text.

UNICODE

Deswegen wurde 1991 der Unicode-Standard veröffentlicht, der einen Zeichensatz für alle Textelemente aller Schriftkulturen und Zeichensysteme festlegt. Damit ein sollte verhindert werden, dass unterschiedliche Kulturkreise unterschiedliche, zueinander inkompatible, Zeichenkodierungen verwenden. Die ersten 128 Zeichen sind deckungsgleich mit dem ASCII Standard. Es werden 32 Bit pro Zeichen verwendet, somit können über vier Milliarden verschiedene Zeichen unterschieden werden. (Aber es werden nur rund 1 Millionen verwendet)

Aufgrund der großen Anzahl an Zeichen werden die Zeichen in hexadezimale Schreibweise folgendermaßen dargestellt:

A….U+0041
€….U+20AC

UTF (Unicode Transformation Format)
Weil es zu aufwendig und eine Speicherverschwendung wäre, alle Texte in voller Länge (also 32 Bit zu kodieren), ist für viele Anwendungen zum Austauschen von Texten die Zeichenkodierung für Unicode Zeichen UTF-8 zum Standard geworden. Sie ist de-facto Standard im Internet und den damit verbundenen Technologien. Ein Zeichen wird in mindestens einem Byte gespeichert. Wird ein Text mit UTF-8 codiert, der nur ASCII-Zeichen enthält, ist die Datei also identisch mit einer direkt ASCII kodierten Datei, und daher auch kompatibel. Falls aber andere Zeichen wie das € Zeichen oder umlaute verwendet werden, können auch 2,3 oder 4 bytes verwendet werden.

ANSI

Die ANSI- Zeichenkodierung ist eine Erweiterung von ASCII, die eine Umstellung von 7-Bit auf 8-Bit mit sich bringt. Dadurch verdoppelt sich die Zeichenanzahl, in den neuen Zeichen sind Umlaute, Sonderzeichen, etc.. Sie wird meistens in Westeuropäischen Ländern verwendet.

Zeichenkodierung in C

Die Zeichenkodierung in C hängt vom Betriebssystem ab, in dem das Programm ausgeführt wird. Bei Windows ist das oft Codepage 1252 (ANSI)

1.2 Strings in C

Strings in C sind:

- Felder vom Datentyp char. Somit stehen keine eigenen Operatoren, zum Beispiel zum Verbinden von Strings zur Verfügung. Deswegen sind spezielle String-Funktionen notwendig.
- Generell null-terminiert

Sie werden folgendermaßen definiert bzw. intiialisiert:

```
char s[8];
```
oder
```
char s[8] = "Hallo";
```

S ist in diesem Fall ein Pointer (Zeiger) auf das erste Element des Strings (Feld)

ACHTUNG!!!! Direkte Zuweisung ist nur beim Initialisieren möglich. An anderen Stellen im programm muss strcpy() verwendet werden.

```
s = "Hallo";
```


```
s1 = s2;
```


Folgende Stringfunktionen stehen nach dem Einbinden von `#include <string.h>` zur Verfügung:

- `strcpy(char* Ziel, const char* Quelle);`
  Kopiert einen String in einen anderen (Quelle nach Ziel). Hier wird Zeichen für Zeichen kopiert, die beendende '\0' wird automatisch angefügt.

- `strncpy(char* Ziel, const char* Quelle, size_t num);`
  Kopiert num Zeichen von Quelle zu Ziel. Wenn das Ende des Quellen Strings gefunden wird, bevor num Zeichen kopiert wurden, wird bis zur erreichen von num Zeichen der Rest mit '\0' befüllt.
  Wichtig: strncpy() fügt selbst keinen null-Character ('\0') an das Ende des Ziels an. Daher: Ziel wird nur null-terminiert, wenn die Länge des C-Strings Quelle kleiner ist als num.

- `strcat(char* s1, const char* s2);`
  Verbindet zwei Zeichenketten miteinander. Das '\0' von s1 wird überschrieben. Voraussetzung ist, dass der für s1 reservierte Speicher Speicherbereich ausreichend groß zur Aufnahme von s2 ist. Andernfalls ergibt sich ein undefiniertes Verhalten.

- `strncat(char* s1, const char* s2, size_t n);`
  Gleiches Prinzip wie bei strncpy().

- `int strcmp(char* s1, char* s2);`
  Diese Funktion vergleicht zwei Zeichenketten miteinander, wobei Zeichen für Zeichen deren jeweilige ANSI oder ASCII Codes verglichen werden. Wenn die beiden Strings identisch sind, gibt die Funktion 0 zurück. Sind die Strings unterschiedlich gibt die Funktion einen Wert größer oder kleiner 0 aus: Ein Rückgabewert größer / kleiner 0 bedeutet, dass der Code des ersten ungleichen Zeichens in s1 größer / kleiner ist als der des entsprechenden Zeichens in s2.

- `int strlen(cont char *string);`
  Diese Funktion gibt die Länge eines Strings (ohne dem abschließenden Nullzeichen) zurück.

- `char* strstr(const char* s1, const char* s2);`
  Sucht nach dem ersten Vorkommen der Zeichenfolge s2 (ohne dem abschließenden Nullzeichen) in der Zeichenfolge s1 und gibt einen Zeiger auf die gefundene Zeichenfolge (innerhalb s1) zurück. Ist die Länge der Zeichenfolge s2 0, so wird ein Zeiger auf s1 geliefert; war die Suche erfolglos, wird NULL zurückgegeben.

- `char* strtok(char* s1, const char *s2);`
  Diese Funktion zerlegt einen String s1 mit Hilfe der in s2 gegebenen Trennzeichen (token) in einzelne Teil-Strings. S2 kann also eines oder auch mehrere Trennzeichen enthalten. Jeder Aufruf gibt einen Token zurück.

  Beispiel:

```
char text[] = "Das ist ein Beispiel!";
char trennzeichen[] = " ";
char *wort;
int i=1;
wort = strtok(text, trennzeichen);

while(wort != NULL)
{
    printf("Token %d: %s\n", i++, wort);
    wort = strtok(NULL, trennzeichen);
}
```

1.3 Felder in C

Felder in C haben folgende Eigenschaften (die natürlich auch für C-Strings gelten)

- Sie sind statisch, das heißt die Anzahl im Feld gespeicherten Werte kann nach der Definition nicht mehr geändert werden
- Die Länge muss schon beim Schreiben des Programms definiert werden
- Der Programmierer ist dafür verantwortlich, dass nicht über Feldlänge hinausgeschrieben wird
- Als Feldtyp können alle vordefinierten Datentypen außer void sowie alle selbstdefinierten Datentypen verwendet werden
- Das erste Feldelement hat den Index 0, das zweite am Index 1, usw..

Das Definieren von Feldern ist auf mehrere Arten möglich:

- `int zahl [3];`
- `int zahl [] = {1, 2, 3};`
- `int zahl [5] = {1, 2, 3};`

Die Variable zahl ist ein Zeiger auf die Adresse des ersten Feldelements. Die Feldelemente liegen im Speicher direkt hintereinander. Weil die Feldlänge oft an mehreren Stellen im Programm gebraucht wird, wird sie oft am Anfang des Programms als Präprozessordirektive festgelegt.

Die Feldlänge lässt sich auch mit dem Operator sizeof bestimmen.

Einem Feldelement lässt sich ein Wert folgendermaßen zuweisen:

```
zahl [4] = 1;
```

Ein Feld kann einer Funktion folgendermaßen als Parameter übergeben werden. Dabei ist zu beachten, dass hier nur der Zeiger übergeben wird. Somit kann in der Funktion die Feldlänge nicht mit sizeof ermittelt werden. Deswegen wird meistens auch die Feldlänge mitübergeben.

```
void topFunction(double f[], int length) {…}
```

Ist das gleiche wie:

```
void topFunction(double *f, int length) {…}
```

In C sind auch mehrdimensionale Felder möglich. Dabei ist jedes Feldelement selbst wiederum ein Feld. Im Speicher werden mehrdimensionale Felder Reihe für Reihe abgelegt. Deklaration (zuerst Angabe der Zeilen, dann Spalten):

```
int tabelle [2] [3];
```

Intitialisieren:

```
int tabelle [2] [3] = {{7,8,9}, {-3,2,6}}
```

Übergabe als Parameter (es müssen immer alle Feldgrößen aller Dimensionen auer der ersten Angegeben werden):

```
void topFunction (int tabelle [] [3]);
```

Wenn String und Felder mit statischer Länge für Anwendungen unbrauchbar sind, kann auf dynamische Speicherverwaltung zurückgegriffen werden.

1.4 Strukturen in C

Strukturen in C sind neben Feldern wichtig, um große Datenmengen umgehen zu können. Mit Strukturen werden mehrere zusammengehörige Variablen gruppiert und als eigener „Datentyp“ definiert.

Definieren einer Struktur:

```
struct Adresse {
    char name[50];
    char strasse[100];
    short hausnummer;
};
```

Struktur initialisieren und Zugriff auf Strukturen:

```
struct Adresse {
    char name[50];
    char strasse[100];
    short hausnummer;
};

struct Adresse adresseMichael = {"Ruffenacht", "Wehrstegweg", 41};

printf("wir fahren zum %s. Adresse: %s %d", adresseMichael.name,
       adresseMichael.strasse, adresseMichael.hausnummer);
```

1.5 Zufallszahlen in C

In C kann man mit der Funktion rand() eine Pseudo-Zufallszahl erzeugen. Die Zahl liegt zwischen 0 und einer maximalen Zahl welche mit der Symbolischen Konstante RAND_MAX abgefragt wird.

Funktion zur Erzeugung einer ganzzahligen Zufallszahl:

```
int berechneGanzeZZ(int ug, int og)
{
    return rand() % (og-ug+1) + ug;
}
```

Funktion zur Berechnung einer Double Zufallszahl:

```
double berechneFkZZ(double ug, double og)
{
  return (og-ug) * rand() / RAND_MAX + ug;
}
```

Bevor die Zufallszahlen erstellt werden, sollte die Funktion srand() aufgerufen werden. Diese initialisiert den Zufallszahlen-Generator, um die Zufälligkeit der Zahlen zu gewährleisten. Im Parameter wird ein Zeitstempel übergeben, deshalb muss auch time.h eingebunden werden.

```
srand(time (NULL));
```

1.6 Zeichenkodierung in Java

Die Java Virtual Machine verwendet intern die Kodierung UTF-16. Somit nimmt jedes Zeichen 16 Bit im Speicher ein. Wird ein Unicode-Zeichen verwendet, dessen Abbildung mehr als zwei Byte erfordert, wird dieses Zeichen aus zwei char zusammengesetzt. Somit ist es theoretisch möglich an jeder Stelle im Quellcode Unicode-Zeichen im Format \00ea angegeben werden. Trotzdem sollten für Variablen und Klassennamen aber immer englische Zeichen und Wörter verwendet werden.

1.7 Strings in Java

In Java werden Zeichenketten als Objekte der Klasse String gespeichert. Die Klasse ist immutable. Die Klasse ist in zwei Punkten besonders:

- String-Objekte lassen sich direkt aus String-Literalen (Zeichenketten zwischen zwei doppelten Anführungszeichen (Gänsefüßchen)) erzeugen.
- Strings lassen sich mit + aneinanderreihen

Um Strings miteinander vergleichen zu können sollte der folgende Ausdruck verwendet werden:

```
if(String1.equals(string2))
```

Weil die Klasse String immutable ist, sollten Strings in einer Schleife niemals über den „+“-Operator verbunden werden. Dabei müsste in jedem Schleifenaufruf ein neues String -Objekt erzeugt werden. Deswegen ist es sinnvoll die Klasse StringBuilder zu verwenden, die, wie auch String und Stringbuffer, das Interface CharSequence implementiert.

```
StringBuilder builder = new StringBuilder();

while(…)

{

        String s1 = ….

        builder.append(s1).append(„\n”);

}

return builder.toString();
```

1.8 Felder in Java

Felder in Haben folgende Eigenschaften:

- Sind semidynamisch
  Das heißt, die Feldlänge muss beim Erzeugen festgelegt werden und kann nachträglich nicht mehr verändert werden. Anders als in C kann ei Feldlänge aber durch eine Variable, während dem Ausführen des Programms bestimmt werden.

```
Int[] array = new int[arraylength];
```

- Werden ehr selten verwendet, weil Collections in vielen Fällen besser geeignet sind und für den Programmierer weniger Probleme mit sich bringen.
- Haben Ähnlichkeiten mit Objekten. Die Feldlänge ist im Datenelement length und lässt sich folgendermaßen abfragen: `array.length();`
- Können auch mehrdimensional sein
- Wird ein Feld von Strings definiert, sind die Referenten 0 und es werden keine String Objekte erzeugt.

1.9 Collections in Java

Collections sind dynamische Datenstrukturen, die sich der Anzahl der Daten, die sie aufnehmen, anpassen. Das Interface `Collection` implementiert das interface `Iterable`. `Iterable` definiert nur die Methode `Iterator`, die den iterator liefert. Mit dem iterator ist es möglich in einer for-each-Schleife jedes Objekt der Collection abzufragen. In einer Collection können beliebig viele Objekte eines Typs gespeichert werden. Das Auslesen kann auch nur über den Iterator möglich sein. Das Interface `Collection` definiert die unten ersichtlichen Methoden.

```
Collection <T>
```

- `int size()`
- `add (T)`
- `clear()`

Das Interface List ist von Collection abgeleitet. In einer List

- kann ein Objekt mehrmals vorkommen
- gibt es eine definierte Reihenfolge der Objekte

Das Interface Set ist ebenfalls von Collection abgeleitet. Eigenschaften:

- Entspricht dem einer Menge von Objekten eines Typs
- Ein Objekt ist entweder Teil dieser Menge oder nicht Teil dieser Menge
- Ein Objekt kann somit maximal einmal vorkommen
- Es gibt keine definierte Reihenfolge der Objekte

Die wichtigsten beiden Klassen, die von List abgeleitet sind, sind ArrayList und LinkedList.

Bei der ArrayList werden die Objekte intern in einem Feld gespeichert, dass zu Beginn eine Länge hat. Wenn alle Felder belegt sind und ein neues Feldelement hinzugefügt werden soll, werden alle bisherigen Feldelemente in ein neues Feld mit 1.5-facher Länge kopiert und das neue dort hinzugefügt.

ArrayList eignet sich für Anwendungen, bei denen oft auf einzelne Elemente zugegriffen wird.

Bei der LinkedList wird bei jedem Objekt der liste die Referenz zum nachfolgenden und vorherigen Objekt in der Liste, gespeichert. Somit ist der Rechenaufwand verhältnismäßig gering, wenn ein Objekt mitten in der Liste hinzugefügt oder entfernt werden soll.

Die LinkedList sollte verwendet werden, wenn oft Objekte hinzugefügt oder entfernt werden.

Bei Sets kenne ich mich nicht aus.

1.10 Generics

Mit Generics lassen sich Klassen und Methoden mit Typen parametrisieren. Die Klassen/Funktionen passen sich also dem Typ an. Das sit eine Maßnahme zur typsicheren Programmierung.

Ein klassisches Beispiel hierfür sind Listen oder der Swingworker. Zwischen „<“ und „>“ wird der Datentyp angegeben. Weiter hinten in der Zeile ist nur mehr der Diamant Operator nötig, da der Datentyp von vorne übernommen wird.

```
private List<Integer> list = new ArrayList<>();

int wert = list.get(32);
```

Beim Zuweisen des Werts aus der Liste zu dem der Variable wert ist aufgrund der generischen Programmierung keine Typenumwandlung (Cast) nötig.