Übungsserie 3

Abgabe: Dienstag 22.10.2024

Ein zentraler Aspekt beim Programmieren ist die Arbeit mit Variablen, die eine bestimmte Art von Daten speichern können. Beispielsweise haben wir in den letzten Übungen Variablen des Typs `int`, `float` oder `char` verwendet, um einzelne Werte wie Ganzzahlen, Gleitkommazahlen oder Zeichen zu speichern. Doch oft reicht es nicht aus, nur eine einzelne Variable zu speichern. Manchmal möchten wir mehrere Werte desselben Datentyps auf einmal speichern, etwa eine Reihe von Temperaturen oder Noten. Hier kommen Arrays (aka Felder) ins Spiel. Ein statisches Array ist eine Sammlung von Elementen desselben Datentyps, die im Speicher an aufeinanderfolgenden Stellen abgelegt sind. Alle Elemente eines Arrays können durch einen Index angesprochen werden, der bei `0` beginnt und bis zur Größe des Arrays minus eins läuft. Es ist wichtig zu beachten, dass die Größe eines Arrays separat gespeichert werden muss, da C selbst diese Information nicht verwaltet. Statisch Arrays heißen die Arrays, bei denen ihre Größe zur Kompilierungszeit festgelegt ist. Ein einfaches Beispiel: Angenommen, wir möchten die Temperaturen eines bestimmten Tages (in Celsius) über acht Stunden speichern. Statt acht separate Variablen zu deklarieren, können wir ein Array verwenden:

```c
int temperaturen[8];
```

Dieses Array speichert acht Ganzzahlen. Jedes Element im Array kann über seinen Index angesprochen werden, z.B. `temperaturen[0]` für die erste Temperatur oder `temperaturen[7]` für die letzte. Um ein Element des Arrays zu setzen oder zu lesen, nutzen wir den Index. Zum Beispiel, um die Temperatur für die erste Stunde zu setzen:

```c
temperaturen[0] = 22;
```

Ähnlich können wir den Wert eines bestimmten Elements auslesen, z.B.:

```c
int erste_temperatur = temperaturen[0];
```

Wichtig ist, dass der Index eines Arrays nur innerhalb der Grenzen des Arrays verwendet werden darf, also von `0` bis `n-1`, wobei `n` die Größe des Arrays ist. Eine Überschreitung der Indexgrenzen (z.B. `temperaturen[8]` in unserem Beispiel) führt zu undefiniertem Verhalten, da der Zugriff auf Speicherbereiche erfolgt, die nicht für das Array reserviert sind. Dies kann zu unerwarteten Fehlern oder Abstürzen des Programms führen. Arrays können bei ihrer Deklaration direkt initialisiert werden, indem eine sogenannte Initialisierungsliste verwendet wird. Dies geschieht, indem die Werte in geschweiften Klammern angegeben werden. Zum Beispiel könnte man das Array `temperaturen` mit den folgenden Werten initialisieren:

```c
int temperaturen[8] = {22, 21, 19, 18, 20, 23, 24, 22};
```

In diesem Fall werden die acht Ganzzahlen direkt im Array gespeichert, und jedem Element wird der entsprechende Wert aus der Liste zugewiesen, sodass `temperaturen[0] = 22`, `temperaturen[1] = 21` usw. gilt. Falls die Initialisierungsliste weniger Elemente enthält als das Array, werden die nicht explizit initialisierten Elemente automatisch auf `0` gesetzt. Zum Beispiel:

```c
int temperaturen[8] = {22, 21, 19};
```

In diesem Fall werden die ersten drei Elemente mit `22`, `21` und `19` initialisiert, und die restlichen fünf Elemente werden auf `0` gesetzt. Wird ein statisches Array jedoch ohne Initialisierung deklariert, wie zum Beispiel:

```c
int temperaturen[8];
```

so werden auch diese Elemente (nur bei statischen Arrays!!) auf Null gesetzt. Es ist trotzdem eine gute Praxis, Arrays immer entweder durch eine Initialisierungsliste oder durch explizite Zuweisungen zu initialisieren, da es Situationen geben kann, welche sonst uninitialisierte Elemente erzeugen und damit undefined behavior.

	22	21	19	18	20	23	24	22
Index	0	1	2	3	4	5	6	7

Diese Struktur zeigt ein Beispiel eines statischen Arrays mit acht Elementen, wo jeder Speicherplatz durch einen Index zugänglich ist, der bei `0` beginnt und bei `7` endet.

Damit für alle die gleichen Regeln gelten, müssen Ihre Programme standardkonform und ohne Warnungen mit den Compileroptionen (GCC, Clang, MSVS vergleichbar): `-Wall -Wextra -Werror -pedantic -std=c11` ohne Warnungen oder Fehler kompilieren. Vermeiden Sie insbesondere nicht autorisierte Bibliotheken oder Build-Environments. Reichen Sie Ihre Programme (Bitte nur die Quelltext-Dateien) als ZIP Archiv via Opal ein.

In diesem Übungsblatt werden grundlegende Kontrollstrukturen wie Fallunterscheidungen und Verzweigungen behandelt. Diese sind zentrale Konzepte in der Programmierung, um logische Abläufe zu steuern. Insbesondere liegt der Fokus auf der korrekten Anwendung von `if`-Verzweigungen, `else if`, `switch`-Anweisungen sowie der Short-Circuit Evaluation bei logischen Operatoren.

1. **`if`**-Verzweigung:

   Schreiben Sie ein Programm, das eine Zahl von der Tastatur einliest und überprüft, ob diese Zahl positiv, negativ oder null ist.

   - Wenn die Zahl größer als 0 ist, soll die Ausgabe lauten: „Die Zahl ist positiv“.
   - Wenn die Zahl kleiner als 0 ist, soll die Ausgabe lauten: „Die Zahl ist negativ“.
   - Wenn die Zahl gleich 0 ist, soll die Ausgabe lauten: „Die Zahl ist null“.

2. **`if`**-Verzweigung:

   Schreiben Sie ein Programm, das prüft, ob eine Person sowohl volljährig (mindestens 18 Jahre alt) als auch einen gültigen Führerschein besitzt. Das Programm soll zwei Eingaben vom Benutzer einlesen: das Alter und ob ein Führerschein vorhanden ist (als boolescher Wert `true` oder `false`).

   - Geben Sie „Erlaubnis zum Autofahren“ aus, wenn die Person volljährig ist und einen Führerschein besitzt.
   - Geben Sie „Keine Erlaubnis zum Autofahren“ aus, wenn eine der beiden Bedingungen nicht erfüllt ist.

3. **`if`**-Verzweigung:

   Schreiben Sie ein Programm, das eine Temperatur (als Ganzzahl) einliest und basierend auf den folgenden Bedingungen eine entsprechende Nachricht ausgibt:

   - Wenn die Temperatur kleiner als 0 ist, soll die Nachricht „Es ist sehr kalt“ ausgegeben werden.
   - Wenn die Temperatur zwischen 0 und 10 Grad liegt (einschließlich), soll „Es ist kalt“ ausgegeben werden.
   - Wenn die Temperatur zwischen 11 und 20 Grad liegt, soll „Es ist kühl“ ausgegeben werden.
   - Wenn die Temperatur größer als 20 Grad ist, soll „Es ist warm“ ausgegeben werden.

4. **`if`**-Verzweigung:

   Schreiben Sie ein Programm, das eine Ganzzahl einliest und überprüft, ob die Zahl gerade oder ungerade ist, und ob sie positiv oder negativ ist.

   - Wenn die Zahl positiv und gerade ist, soll die Ausgabe lauten: „Die Zahl ist positiv und gerade“.
   - Wenn die Zahl positiv und ungerade ist, soll die Ausgabe lauten: „Die Zahl ist positiv und ungerade“.
   - Wenn die Zahl negativ und gerade ist, soll die Ausgabe lauten: „Die Zahl ist negativ und gerade“.
   - Wenn die Zahl negativ und ungerade ist, soll die Ausgabe lauten: „Die Zahl ist negativ und ungerade“.
   - Wenn die Zahl 0 ist, soll die Ausgabe lauten: „Die Zahl ist null“.

5. Notenberechnung basierend auf erreichter Punktzahl:

   Entwerfen Sie ein Programm, das auf Basis einer maximalen Punktzahl und der erreichten Punktzahl eines Studierenden die entsprechende Note in einem Notensystem mit folgenden Werten berechnet: 5.0, 4.0, 3.7, 3.3, 3.0, 2.7, 2.3, 2.0, 1.7, 1.3, 1.0.

   - Lesen Sie die maximale Punktzahl sowie die erreichte Punktzahl ein.
   - Weisen Sie basierend auf der erreichten Punktzahl die Note zu und geben sie aus

%	<50%	50–54%	55–59%	60–64%	65–69%	70–74%	75–79%	80–84%	85–89%	90-94%	≥95%%
Note	5.0	4.0	3.7	3.3	3.0	2.7	2.3	2.0	1.7	1.3	1.0

   - Geben Sie eine Legende aus, die zeigt, wie viele Punkte für jede Note erforderlich sind.