Programmieren C:
Array-Parameter, Pointer-Arithmetik in Arrays:
Alle Permutationen eines Arrays erzeugen

Klaus Kusche

Wir wollen uns heute einem uralten Problem der Kombinatorik widmen:
Dem *Erzeugen aller möglichen Permutationen* (Vertauschungen, Anordnungen) einer Menge von Objekten. In unserem Fall sind die Objekte einfach die *ganzen Zahlen* von 1 bis n.
Für n=3 wären alle Permutationen beispielsweise 1 2 3, 1 3 2, 2 1 3, 2 3 1, 3 1 2 und 3 2 1.

Wir wollen dazu eine Funktion **`perm`** schreiben, die in einem *übergebenen und schon befüllten Array* bei jedem Aufruf die *nächste* Permutation der im Array enthaltenen Werte erzeugt, und zwar *in sortierter Reihenfolge*.

Die Funktion soll ein Wahr/Falsch-Ergebnis zurückliefern:
"Wahr", wenn sie die nächste Permutation erfolgreich erzeugt hat, und "Falsch", wenn das Array beim Aufruf schon die letzte Permutation (verkehrt sortierte Zahlen) enthält.

Das folgende Verfahren dazu war schon im 14. Jahrhundert bekannt:

- Suche *von hinten* die erste Zahl, die *kleiner als die Zahl dahinter* ist.

  Wenn du *keine* findest, dann enthält das Array schon die letzte Permutation, und du kannst sofort *erfolglos* zurückkehren.

  Merk dir die Position (Index) dieser Zahl als *"links"*.
- Suche *von hinten* die erste Zahl, die *größer als die Zahl an der Stelle "links"* ist.

  Eine solche Zahl findest du immer.

  Merk dir die Position dieser Zahl als *"rechts"*.
- *Vertausche* die Zahlen an den Positionen *"links"* und *"rechts"*.
- Drehe die *Reihenfolge* aller Zahlen zwischen der Position *"links + 1"* und dem *Ende* des Arrays um.
- Kehre erfolgreich zurück: Das Array enthält die nächste Permutation.

Da wir Funktionen üben wollen, teilen wir das Programm gemäß der Programm-Idee in möglichst viele kleine Funktionen auf:

- Eine Funktion **`swap`**: Sie bekommt das *Array und zwei Positionen* übergeben und *vertauscht* die Array-Elemente an den beiden angegebenen Positionen.
  Die Funktion hat keinen Returnwert.
- Eine Funktion **`reverse`**: Sie bekommt das *Array und zwei Positionen* übergeben und bringt die Array-Elemente zwischen diesen beiden Positionen (jeweils einschließlich!) *in umgekehrte Reihenfolge*.
  Auch **`reverse`** hat keinen Returnwert.

  Gehe dazu wie folgt vor:

  Solange die linke Position *kleiner* als die rechte Position ist, vertausche die Elemente an diesen beiden Positionen mittels **`swap`** und gehe dann mit der linken Position eins nach rechts und mit der rechten Position eins nach links.

- Eine Funktion **`find_smaller`**, die mit *einem Array und der Anzahl der Elemente* im Array aufgerufen wird und das Array *von hinten nach vorne* durchläuft, bis sie eine Zahl findet, die *kleiner als die Zahl dahinter* (an der nächsten Position) ist.
  Achtung: Bei welcher Position *beginnt* deine Schleife,
  wenn du mit der Zahl *dahinter* vergleichen musst?

  Ist die Suche erfolgreich, wird die *Position der gefundenen Zahl* zurückgeliefert, gibt es keine solche Zahl, soll das Ergebnis **`-1`** sein.

- Eine Funktion **`find_larger`**, die mit *einem Array, der Anzahl der Elemente* im Array *und einem Zahlenwert* aufgerufen wird und das Array *von hinten nach vorne* durchläuft, bis sie eine Zahl findet, die *größer als die angegebene Zahl* ist.
  Die Funktion liefert die *Position der gefundenen Zahl* als Ergebnis zurück.

  Obwohl die Funktion bei unserem Permutationsprogramm immer so aufgerufen wird, dass eine solche Zahl gefunden wird, sind wir vorsichtig und geben **`-1`** zurück, wenn wir bis zum Array-Anfang keine größere Zahl gefunden haben.

- Unsere *oben beschriebene Funktion* **`perm`** (*Array und Anzahl der Elemente* als Parameter, *Erfolg/Misserfolg* als Returnwert) lässt sich unter Verwendung der vier soeben beschriebenen Funktionen ganz einfach programmieren:
  - **`find_smaller`** liefert uns die Position *"links"* (bei **`-1`** sind wir erfolglos fertig)
  - **`find_larger`** (mit dem Array-Wert an der Stelle "links" als Suchwert) liefert uns die Position *"rechts"*
  - **`swap`** vertauscht die beiden Elemente "links" und "rechts"
  - **`reverse`** dreht die Elemente von "links + 1" bis incl. letztem Element um

- Für das Hauptprogramm schreiben wir noch eine Funktion **`fill`** (ohne Returnwert), die ein übergebenes Array von vorne mit den *Zahlen von 1 bis n in aufsteigender Reihenfolge füllt* (n wird der Funktion ebenfalls übergeben).
  Achtung: Die Zahlen gehen von 1 bis n, die Positionen von 0 bis n-1 !

- Weiters schreiben wir noch eine Funktion **`print`** (auch ohne Returnwert), die mit einem Array und der Anzahl der Elemente aufgerufen wird und die Elemente des Arrays in einer Zeile *ausgibt*.

Schreib dazu ein **Hauptprogramm**, das mit einer *Zahl n auf der Befehlszeile* aufgerufen wird (bei fehlender Zahl, n ‹ 1 oder n › 30 soll eine *Fehlermeldung* kommen!) und *alle Permutationen der Zahlen von 1 bis n ausgibt*.

Mache dazu Folgendes:

- Lege ein Array der Größe n an und fülle es mittels **`fill`** mit den Zahlen von 1 bis n in aufsteigender Reihenfolge.
- Gib es mit **`print`** aus.
- Ruf in einer Schleife immer wieder **`perm`** mit dem Array auf:
  - Wenn **`perm`** "Falsch" geliefert hat: Beende die Schleife und das Programm.
  - Sonst:
    Gib das Array mit **`print`** aus und mach den nächsten Schleifendurchlauf.

Zusatzaufgabe: Pointer-Parameter und Pointer-Schleifen

Gib den Funktionen **`swap`**, **`reverse`**, **`find_smaller`** und **`find_larger`**
als Parameter und Returnwert *Pointer* in das Array *statt Indices*
und baue sie auch intern auf *Schleifen mit Pointern* statt Indices um.

Es ergeben sich folgende *Änderungen*:

- Die Funktionen **`swap`** und **`reverse`** bekommen für die linke und rechte Position *zwei Pointer* statt zwei Indices übergeben (der *Array-Parameter* fällt *weg*).
  Dementsprechend laufen bei der *Schleife* in **`reverse`** *zwei Pointer gegeneinander*.
- Bei den Funktionen **`find_smaller`**, und **`find_larger`** ändert sich am meisten:
  - *Statt der Array-Größe* wird ein *Pointer auf das letzte Element* des Arrays übergeben.
  - Statt einem Index wird ein *Pointer auf das gefundene Element zurückgegeben*.
    Der *"nicht gefunden"*-Wert ändert sich dadurch von **`-1`** auf **`NULL`**.
  - Die Schleifen werden auf *Pointer-Schleifen* umgebaut
    (*Achtung:* Überlege genau, was der *neue Anfangswert* der Schleifen ist!).
- Der *Prototyp* von **`perm`** bleibt *unverändert*, aber intern gibt es Änderungen:
  - Für den Aufruf von **`find_smaller`**, **`find_larger`** und **`reverse`** brauchen wir einen *Pointer auf das letzte Element des Arrays* (mach dafür eine *Hilfsvariable*).
  - **`left`** und **`right`** werden *Pointer statt Indices*.
    Damit ändert sich u.a. die *Prüfung* auf "erfolglos".
- Die Funktion **`fill`** *bleibt*, wie sie ist:
  Da wir in der Schleife nicht nur das **`i`**-te Element,
  sondern auch die Zahl **`i`** selbst brauchen, bringt ein Umbau auf Pointer nichts.
- Der *Prototyp* von **`print`** bleibt *gleich*,
  aber *intern* solltest Du die Index-Schleife durch eine *Pointer-Schleife* ersetzen.
- Auch das *Hauptprogramm* bleibt *unverändert*.