AIK Programmieren 1 Übung: Array- und Referenzparameter

Klaus Kusche

1.) Funktion für Minimum und Maximum

Schreib eine Funktion, die mit einem *Array* von **`double`**-Zahlen (und der *Länge des Arrays*) aufgerufen wird und (in einem einzigen Durchlauf über die Array-Elemente!) *sowohl das Minimum als auch das Maximum* der Zahlen ermittelt.

Als *Returnwert* soll die Funktion anzeigen, ob sie *erfolgreich* war (**`true`**), oder ob als *Länge* des Arrays *Null* oder eine negative Zahl übergeben wurde (**`false`**).

Überlege, wie du Minimum und Maximum zurück zum Aufrufer übermittelst, ohne globale Variablen zu verwenden! (du darfst der Funktion *weitere Parameter* geben!)

Schreib dazu ein **`main`**, das

- zuerst beliebig viele Worte auf der Befehlszeile der Reihe nach in einen **`double`** verwandelt und in einem Array speichert (dimensioniere das Array je nach Anzahl der Worte auf der Befehlszeile!)
- und dann die Minimum-Maximum-Funktion mit diesem Array aufruft. Wenn die Funktion Erfolg meldet, werden Minimum und Maximum ausgegeben, bei Misserfolg (keine Zahlen auf der Befehlszeile / im Array) eine Fehlermeldung.

2.) Sortiertes Einfügen von Zahlen

Gesucht ist ein Programm, das alle auf der Befehlszeile angegebenen (ganzen) Zahlen in ein Array einfügt, und zwar *sortiert und ohne doppelte Zahlen*.

Das Programm soll *in 3 Funktionen + **`main`** unterteilt* sein:

- **`find`** wird mit einem *Array*, der *Anzahl* der Zahlen im Array und der zu suchenden *Zahl* aufgerufen und liefert die (erste) *Position (den Index) der Zahl im Array* zurück (oder **`-1`**, wenn die Zahl nicht gefunden wird bzw. das Array keine Zahlen enthält). Simples Suchen von vorne nach hinten genügt.
- **`insert`** wird mit einem *Array*, der aktuellen *Anzahl* der Zahlen im Array, der dimensionierten *Größe* des Arrays und der einzufügenden *Zahl* aufgerufen und *fügt die Zahl an der richtigen Stelle ein*, sodass das Array sortiert bleibt (**`insert`** darf sich darauf verlassen, dass das Array beim Aufruf *schon sortiert* ist).

  Du musst beim Einfügen *von hinten nach vorne* arbeiten und die bestehenden Elemente so lange *um 1 nach rechts verschieben* (d.h. eine *Lücke von hinten nach vorne wandern* lassen), bis du die richtige Stelle für die neue Zahl gefunden hast.

  Als Returnwert soll die Funktion die *Position* (den Index) liefern, *an der die Zahl eingefügt wurde*. Geht das Einfügen schief (weil das Array schon voll ist oder eine negative Anzahl übergeben wurde), soll der Returnwert **`-1`** sein.

Weiters soll die Funktion (wenn sie die Zahl erfolgreich eingefügt hat) die *Anzahl der Zahlen im Array um 1 erhöhen*
(wie muss die aktuelle Anzahl der Zahlen im Array daher übergeben werden, damit die Funktion die im Aufrufer gespeicherte Anzahl erhöhen kann?).

- **`insertNoDup`** soll eine Zahl *nur dann* in das Array *einfügen*, wenn sie *noch nicht* im Array enthalten ist.

  Es hat genau dieselben Parameter und denselben Returnwert wie **`insert`** und besteht nur aus zwei Funktionsaufrufen und einem **`if`**:

  - Zuerst einmal wird **`find`** für das Array und die Zahl aufgerufen. Findet **`find`** die Zahl, endet **`insertNoDup`** sofort mit Returnwert **`-1`** (*"nicht eingefügt"*).
  - Hat **`find`** die Zahl nicht gefunden, so wird **`insert`** für die Zahl aufgerufen und dessen Returnwert als Returnwert von **`insertNoDup`** zurückgegeben.

- **`main`** besteht im Wesentlichen aus *zwei Schleifen*:

  - Die erste Schleife geht die *Worte auf der Befehlszeile* einzeln durch, verwandelt sie in eine Zahl, und versucht sie mittels **`insertNoDup`** in ein im Hauptprogramm angelegtes, ursprünglich leeres Array einzufügen.

    Ist das Einfügen erfolgreich, soll die Zahl und die von **`insertNoDup`** gelieferte Position ausgegeben werden, sonst die Zahl und "nicht eingefügt".
  - Nach dem Einfügen aller Zahlen soll eine weitere Schleife den *Inhalt des Arrays ausgeben*. Auch die aktuelle Anzahl der Elemente im Array soll ausgegeben werden.

  Die Größe des Arrays im Hauptprogramm soll durch eine **`#define`**-Konstante fix vorgegeben werden (damit man auch testen kann, was passiert, wenn auf der Befehlszeile mehr Zahlen angegeben sind als in das Array passen).

Anmerkung:

Bei größeren Datenmengen ist es *nicht klug*, die Daten *sortiert in einem Array zu speichern*, wenn immer wieder einzelne Elemente eingefügt oder gelöscht werden sollen:
Das Verschieben beim Einfügen und Löschen *dauert viel zu lange!*

Es gibt für diesen Zweck viel besser geeignete Methoden (z.B. binäre Bäume)!