technische universität dortmund

fakultät für informatik

Lutz Oettershagen
Jurij Kuzmic

Dortmund, den 22. November 2018

Übung zur Vorlesung EidP (WS 2018/19)

Blatt 6

Block rot

Es können 4 Punkte erreicht werden.

Abgabedatum: 29. November 2018, 23:59 Uhr

Hinweise

- Bitte beachten Sie die aktuellen Hinweise unter

  `https://ls11-www.cs.tu-dortmund.de/teaching/ep1819uebung/`

- Für die Abgabe sind die Dateien `Aufgabe_06_1.txt`, `Aufgabe_06_2a.h`, `Aufgabe_06_2a.cpp`, `Aufgabe_06_2b.h`, `Aufgabe_06_2b.cpp` und `Aufgabe_06_2c.cpp` zu erstellen.
- Für die Kompilierung muss ebenfalls sichergestellt werden, dass sich Ihre Programme mit den Parametern `-pedantic` und `-Werror` kompilieren lassen. Bei der Kompilierung von der Kommandozeile müsste der Befehl wie folgt aussehen:

  ```
  g++ -pedantic -Werror -o bubblesort Aufgabe_06_2a.cpp
  Aufgabe_06_2b.cpp Aufgabe_06_2c.cpp
  ```

- Stellen Sie sicher, dass alle von Ihnen abgegebene Dateien reine Textdateien im UTF-8-Format sind.
- Am **Donnerstag, 29.11.2018** um **14:15 - 15:45 Uhr** findet eine **Probeklausur** in **HG II, HS 3** statt. Näheres dazu finden Sie auf den Veranstaltungswebseiten.

Aufgaben

Aufgabe 1: Grundlagen (1 Punkt)

Legen Sie für Ihre Antworten eine Textdatei `Aufgabe_06_1.txt` an.

a) Erklären Sie den Unterschied zwischen einer lokalen und einer globalen Variable inklusive der Sichtbarkeitsbereiche. (0.2 Punkte)

b) Wie verhält sich eine statische Variable in einer Funktion? Welche Auswirkung hat dies auf den Gültigkeitsbereich und die Sichtbarkeit? (0.2 Punkte)

c) Welches Problem wird mit Namensräumen gelöst? Geben Sie ein Beispiel für einen beliebigen Namensraum an. (0.2 Punkte)

d) Was sind rekursive Funktionen? (0.1 Punkte)

e) Wie funktioniert das Prinzip der Rekursion in einer Funktion einschließlich Rekursionsverankerung? (0.3 Punkte)

Aufgabe 2: Rekursion (3 Punkte)

In dieser Aufgabe müssen Sie zunächst einen rekursiven Sortieralgorithmus implementieren, um ein vorgegebenes Array sortieren zu können. Danach müssen Sie einen rekursiven Suchalgorithmus implementieren, um innerhalb des vorgegebenen Arrays einen bestimmten Wert zu finden. Eine Art zu sortieren haben Sie bereits im Aufgabenblatt 0 kennengelernt, diese Art des Sortierens wird *Bubblesort* genannt.

a) Rekursives Sortieren
Legen Sie für diese Teilaufgabe die Dateien `Aufgabe_06_2a.h` und `Aufgabe_06_2a.cpp` an. Implementieren Sie dazu den rekursiven Bubblesort-Algorithmus und beschreiben Sie die Header-Dateien korrekt, sodass die `bubblesort`-Funktion in der Teilaufgabe c) über eine Header-Datei eingebunden werden kann. Der Funktionskopf soll dabei wie folgt aussehen:

```
void bubblesort(int *array, unsigned int start, unsigned int end);
```

Benutzen Sie für die Implementierung eine Rekursion. Das Benutzen von Schleifen (egal, ob `do-while`, `while` oder `for`) ist nicht erlaubt. Als Tipp erhalten Sie eine wörtliche Beschreibung des rekursiven Bubblesort-Algorithmus:

- Falls `start` kleiner `end - 1` ist, tue Folgendes.
- Falls Element[`start`] größer als Element[`start+1`] ist, vertausche diese Elemente und wiederhole Bubblesort auf dem gesamten Array.
- Ansonsten (Element[`start`] nicht größer als Element[`start+1`]) erhöhe `start` um 1 und wiederhole Bubblesort.

Für das Vertauschen zweier Elemente wird Ihnen die `swap`-Funktion vorgegeben. Diese Funktion tauscht die Werte der beiden Eingabeparameter `int &a` und `int &b`. Sie können die `swap`-Funktion in Ihre `bubblesort` Implementierung einbauen und sich somit Zeilen sparen.

```
void swap(int &a, int &b) {
  int temp = a;
  a = b;
  b = temp;
}
```

Allerdings müssen Sie die `swap`-Funktion an die richtige Stelle in eine der drei Dateien (`Aufgabe_06_2a.cpp`, `Aufgabe_06_2b.cpp` oder `Aufgabe_06_2c.cpp`) setzen, damit das Programm kompiliert werden kann. Überlegen Sie sich also, welche Funktion `swap` nutzt und sie daher beim Kompilieren kennen sollte.
Ganz wichtig ist jedoch dass die Implementierungen Rekursionen enthalten muss und keine Schleifen genutzt werden dürfen! (1 Punkt)

b) Rekursives Suchen
Legen Sie für diese Teilaufgabe die Dateien `Aufgabe_06_2b.h` und `Aufgabe_06_2b.cpp` an. Implementieren Sie als nächstes den Such-Algorithmus und beschreiben Sie die Header-Dateien korrekt, sodass die `finde`-Funktion in der Teilaufgabe c) über eine Header-Datei eingebunden werden kann. Der Funktionskopf soll dabei wie folgt aussehen:

```
int finde(int wert, int *folge, unsigned int intervallLaenge);
```

Bei der Implementierung teilt man ein sortiertes Array in zwei etwa gleich große Arrays und sucht im jeweils zum Suchwert passenden Array weiter, bis der Wert gefunden wird, oder festgestellt wird, dass der Wert nicht vorhanden ist. Der Algorithmus soll den Wert -1 zurückgeben,

falls der Suchwert im Array nicht vorhanden ist. Ansonsten soll die Position (Index im Array), an dem sich der Suchwert befindet, zurückgegeben werden.
Falls Sie Teilaufgabe a) nicht lösen konnten, können Sie die folgende iterative Implementierung von `bubblesort` verwenden, um Aufgabenteil b) zu lösen.

```
1  void bubblesort(int *array, unsigned int start, unsigned int end) {
2    for (unsigned int i = start; i < (end - 1); ++i) {
3      for (unsigned int j = start; j < (end - i - 1); ++j) {
4        if (array[j] > array[j + 1]) {
5          int tmp = array[j];
6          array[j] = array[j + 1];
7          array[j + 1] = tmp;
8        }
9      }
10   }
11 }
```

Ganz wichtig ist, dass die Implementierung von `finde` Rekursionen enthalten muss und keine Schleifen genutzt werden dürfen! (1.7 Punkte)

c) Testen der Algorithmen
Benutzen Sie für die Teilaufgabe c) die vorgegebene Datei `Aufgabe_06_2c.cpp`. Testen Sie Ihre Implementierungen auf der vorgegebenen `main`-Funktion. Als Hilfe bekommen Sie die Funktion `printArray`. Die Funktion `printArray` gibt ein Array auf dem Terminal aus; damit können Sie überprüfen, ob das Array richtig sortiert wurden. Im Folgenden wird Ihnen die Vorlage für `Aufgabe_06_2c.cpp` gegeben. Nach erfolgreicher Implementierung der Funktionen `bubblesort` und `finde` sollten ab Zeile 28 die Positionen der gesuchten Werte ausgegeben werden. Am Ende sollten Sie 5 Dateien haben (`Aufgabe_06_2a.h`, `Aufgabe_06_2a.cpp`, `Aufgabe_06_2b.h`, `Aufgabe_06_2b.cpp`, `Aufgabe_06_2c.cpp`).
Aufgabe: Binden Sie die Header-Dateien für `bubblesort` und `finde` korrekt ein. Kopieren Sie die Ausgabe des Programms an das Ende dieser Datei.

```
1  /*** Aufgabe_06_2c.cpp ***/
2  #include <iostream>
3  using namespace std;
4
5  //************************************************************
6  // Hier die Header-Dateien fuer bubblesort und finde einbinden
7  //************************************************************
8
9  void printArray(int const *array, unsigned int size);
10
11 int main() {
12   unsigned int const n = 7;
13   int folge_a[n] = { 7, 5, 66, 23, 8, 40, 46 };
14   int folge_b[n] = { 95, 90, 80, 50, 20, 40, 40};
15   int folge_c[n] = { -10, 13, -8, -5, 0, 8, 23};
16
17   /*** Sortiere die Folgen ***/
18   bubblesort(folge_a, 0, n);
19   bubblesort(folge_b, 0, n);
```

```
20    bubblesort(folge_c, 0, n);
21
22    /*** Zeige die Arrays zur Pruefung der korrekten Sortierung ***/
23    printArray(folge_a, n);
24    printArray(folge_b, n);
25    printArray(folge_c, n);
26
27    /*** Suche im ersten Array folge_a die Werte 66, 8, 6 und 70 ***/
28    cout << "66 auf Position: " << finde(66, folge_a, n) << endl;
29    cout << "8 auf Position: " << finde(8, folge_a, n) << endl;
30    cout << "6 auf Position: " << finde(6, folge_a, n) << endl;
31    cout << "70 auf Position: " << finde(70, folge_a, n) << endl;
32
33    /*** Suche im zeiten Array folge_b die Werte -2, 40, 20, 95 ***/
34    cout << "-2 auf Position: " << finde(-2, folge_b, n) << endl;
35    cout << "40 auf Position: " << finde(40, folge_b, n) << endl;
36    cout << "20 auf Position: " << finde(20, folge_b, n) << endl;
37    cout << "95 auf Position: " << finde(95, folge_b, n) << endl;
38
39    /*** Suche im dritten Array folge_c die Werte -8, -14, 8 ,0 ***/
40    cout << "-8 auf Position: " << finde(-8, folge_c, n) << endl;
41    cout << "-14 auf Position: " << finde(-14, folge_c, n) << endl;
42    cout << "8 auf Position: " << finde(8, folge_c, n) << endl;
43    cout << "0 auf Position: " << finde(0, folge_c, n) << endl;
44  }
45
46  void printArray(int const *array, unsigned int size) {
47    for (unsigned int i = 0; i < size; ++i)
48      cout << array[i] << " ";
49    cout << endl;
50  }
51  /********* Ende *********/
```

(0.3 Punkte)

Präsenzaufgabe 3: Rekursionen (0 Punkte)

In dieser Aufgabe sollen drei Probleme **rekursiv** gelöst werden. Für diese Aufgabe wird Ihnen die `main`-Funktion und die Ausgabe vorgegeben. Ihre Aufgabe ist es, die drei folgenden Probleme zu implementieren.

a) Implementieren Sie die `summenformel`-Funktion, so dass diese die n-te Zahl der Summenformel ausgibt. Die Summenformel ist definiert durch:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \frac{n^2+n}{2} = 1+2+3+4...$$

(0 Punkte)

b) Implementieren Sie die `fibonacci`-Funktion, so dass diese Funktion die n-te Fibonaccizahl zurückgibt. Die Fibonaccizahlen sind definiert durch:

$$f(i) = \begin{cases} 0 & \text{falls } i \leq 0 \\ 1 & \text{falls } 0 < i \leq 2 \\ f(n-1) + f(n-2) & \text{falls } i > 2 \end{cases}$$

(0 Punkte)

c) Implementieren Sie die `fakultaet`-Funktion, so dass diese die n-te Fakultät ausgibt. Die Fakultät ist definiert durch:

$$\prod_{i=1}^{n} i = 1 \cdot 2 \cdot 3 \cdot \dots \cdot (n-1) \cdot n$$

(0 Punkte)

Lösen Sie diese Aufgaben mit Hilfe von Rekursionen. Das Verwenden von Schleifen (egal, ob `do-while`, `while` oder `for`) oder eine direkte Berechnung ist nicht erlaubt. Benutzen Sie folgende Vorlage.

```
1  #include <iostream>
2  using namespace std;
3
4  int summenformel(int x) {
5    // hier Ihre Implementierung
6  }
7
8  int fibonacci(int x) {
9    // hier Ihre Implementierung
10 }
11
12 int fakultaet(int x) {
13   // hier Ihre Implementierung
14 }
15
16 int main() {
17   int const n = 10;
18   for (int i = 0; i < n; ++i)
19     cout << fibonacci(i) << " ";
20   cout << endl;
21   for (int i = 0; i < n; ++i)
```

```
22      cout << fakultaet(i) << " ";
23    cout << endl;
24    for (int i = 0; i < n; ++i)
25      cout << summenformel(i) << " ";
26    cout << endl;
27    return 0;
28  }
29  /* Ihre Ausgabe sollte wie folgt aussehen:
30  0 1 1 2 3 5 8 13 21 34
31  0 1 2 6 24 120 720 5040 40320 362880
32  0 1 3 6 10 15 21 28 36 45
33  */
```