Brandenburgische
Technische Universität
Cottbus - Senftenberg

Brandenburgische Technische Universität Cottbus – Senftenberg
Institut für Informatik, Informations- und Medientechnik
PD Dr.-Ing. habil. Thomas Hinze

Wintersemester 2015 / 2016

Abgabe der Lösungen: bis Mittwoch, 20.01.2016

Einführung in die Programmierung
Übungsblatt 5

Hinweise

- Erstellen Sie bitte eine schriftliche Lösung der nachfolgenden Aufgaben und erzeugen Sie daraus eine pdf-Datei. Sie können auch Ihre (lesbare!) handschriftliche Lösung als pdf einscannen. Achten Sie bitte darauf, dass am Beginn der ersten Seite des pdfs Ihr *Name*, Ihre *Matrikelnummer* und der Name Ihres *Tutors* (Laborübungsgruppe) vermerkt ist. Jede Aufgabe, die durch **(Laborübung)** gekennzeichnet ist, beinhaltet als Lösung auch einen C-Quelltext. Bitte packen Sie Ihr pdf gemeinsam mit allen C-Quelltextdateien in eine zip-Datei, die Sie termingemäß im moodle hochladen.
- Sie bekommen eine Bewertung Ihrer Lösung mit der Information, ob sie ausreichend zum Bestehen des Übungsblattes ist. Im Falle einer nichtausreichenden Lösung haben Sie Gelegenheit, eine verbesserte Version zeitnah, aber spätestens bis 01.02.2016 im moodle nachzureichen. Plagiate werden nicht als ausreichende Lösungen anerkannt.

Aufgabe 1

(Laborübung) Der fiktive Lebensmittelhersteller *ProAdipo* produziert Flaschenketchup. Jede Ketchupflasche soll idealerweise eine Inhaltsmasse von 450 Gramm besitzen. Aus technischen Gründen variiert jedoch die Füllmenge in den einzelnen Flaschen. Der Wiegeautomat hat in einem Produktionszeitraum folgende 12 Inhaltsmassen (in Gramm) gemessen: 451.23, 449.71, 449.85, 450.02, 448.79, 450.11, 447.39, 450.28, 450.94, 448.20, 446.98 und 451.19.
Schreiben Sie ein C-Programm, das die einzelnen Massen in einem Feld erfasst sowie die minimale Masse, die maximale Masse und die durchschnittliche Masse (arithmetisches Mittel) bestimmt und ausgibt. Flaschen, deren Inhaltsmasse um mehr als $0,5$ Prozent unter 450 Gramm liegt, dürfen nicht in den Handel gebracht werden. Erweitern Sie Ihr C-Programm um eine Funktion `pruefe`, die die Maßzahlen auf Einhaltung der Mindestfüllmenge prüft und die Inhaltsmassen zu leichter Flaschen identifiziert und anzeigt. Darüber hinaus soll die Funktion `pruefe` die Anzahl zu leichter Flaschen als Funktionswert zurückgeben.

Aufgabe 2

Gegeben ist der folgende Lückenquelltext eines C-Programmes, das zu einem eingegebenen Kalenderdatum aus Tag, Monat und Jahr den zugehörigen Wochentag im Gregorianischen Kalender ab 01.01.1583 ermittelt, wobei die *Gaußsche Wochentagsformel* zur Anwendung kommt.

- **(Laborübung)** Ergänzen Sie bitte die ausgegrauten Stellen im Quelltext entsprechend der jeweiligen Kommentare, ohne darüber hinaus den Quelltext zu modifizieren.
- **(Laborübung)** Testen Sie anschließend Ihr Programm, indem Sie die Wochentage folgender denkwürdiger Daten berechnen und mithilfe einer Internet-Recherche überprüfen:

20.07.1969 ..... Landung des ersten Menschen auf dem Mond
09.11.1989 ..... Fall der Berliner Mauer
03.10.1990 ..... Tag der Deutschen Wiedervereinigung
01.01.2002 ..... Einführung des Euro-Bargeldes
??.??.???? .... Tag Ihrer Geburt (optional – Bitte nur den Wochentag aufschreiben, nicht das Datum)

- Wie lässt sich innerhalb eines C-Programmes ermitteln, wieviel Speicherplatz ein Variablenwert (Datensatzbelegung) vom Typ `struct Kalenderdatum` beansprucht?
- Nennen Sie über das Erfassen von Kalenderdaten hinaus bitte noch *drei* weitere Beispiele für *sinnvolle* Verwendungen zusammengesetzter Datentypen mit `struct`.

```
#include <stdio.h>
/* Zusammengesetzten Datentypen fuer Kalenderdatum global definieren */
struct Kalenderdatum
{
  int tag;
  int monat;
  int jahr;
};

/* Funktion zur Plausibilitaetspruefung und Wochentagsberechnung */

int BestimmeWochentag(struct Kalenderdatum kal)
{
  int d =          ; //Zuweisen des uebergebenen Tages aus der Struktur
  int m =          ; //Zuweisen des uebergebenen Monats aus der Struktur
  int y =          ; //Zuweisen des uebergebenen Jahres aus der Struktur
  int s=0; //Schaltjahresmarker
  /* Feld mit Anzahlen der Tage pro Monat belegen. Index entspricht Monat */
  int mtag[13]={0,31,28,

  if (((y%4)==0) && ((y%100)!=0) || ((y%400)==0)) {s=1;} //Schaltjahresmarker
  if ((y<1583) || (y>9999)) {return -1;}          //Test: sinnvolle Jahresangabe
  if ((m<1) || (m>12)) {return -1;}               //Test: gueltige Monatsangabe
  if ((d<1) || (d>mtag[m]+s*(m==2))) {return -1;} //Test: gueltige Tagesangabe

  /* rechnerischen Jahresbeginn auf 0.3. schieben */
  if (m < 3) {m += 13; y--;} else {m++;}
  s = d + 26*m/10 + y + y/4 - y/100 + y/400 + 6;  // eigentliche Berechnung
  return (s % 7);                 //0: Sonntag, 1: Montag, ..., 6: Samstag
}


/* Hauptfunktion mit Nutzerschnittstelle */

int main(void)
{
  int day, month, year, w;
  struct Kalenderdatum eingabedatum;
  /* Feld aus Zeichenkettenkonstanten mit den einzelnen Wochentagen anlegen */
  /* Erstindex: 0: Sonntag, 1: Montag, ..., 6: Samstag */
  char wtg[  ][11] = {"Sonntag",
```

```
  printf("\nWochentagsbestimmung aus Kalenderdatum. Bitte eingeben\n");
  printf("\nTag: ");
  scanf("%d", &day);
  printf("Monat: ");
  scanf("%d", &month);
  printf("Jahr: ");
  scanf("%d", &year);

  /* Elemente von eingabedatum mit den eingegebenen Werten belegen */
  [redacted]

  w = BestimmeWochentag(eingabedatum);
  if (w < 0) {printf("Fehler\n"); return -1;}
  printf("Wochentag: %s\n", wtg[w]);
  return 0;
}
```

Aufgabe 3

Zwischen etwa 20 und 25 Prozent der Rechenzeit von Großrechenanlagen entfallen nach einer Studie der Stanford University auf das Ausführen von Sortiervorgängen. Sortierverfahren gehörten zudem zu den ersten Algorithmen, die für den Computereinsatz entwickelt wurden. Sortieren dient bei weitem nicht nur dazu, Tabellen oder Ergebnisdaten für den Nutzer leicht erschließbar darzustellen, sondern ist Bestandteil zahlreicher Algorithmen wie beispielsweise der Voraborganisation großer Datenbestände zum zeiteffizienten Durchsuchen oder bei heuristischen Algorithmen, die ein Problem dadurch lösen, dass sie eine Vielzahl potentieller Lösungskandidaten raten und dann jeden dieser Lösungskandidaten ob seiner Tauglichkeit und Qualität als Lösung bewerten.

- **(Laborübung)** Implementieren Sie das Sortierverfahren *Selectionsort* in C. Die zu sortierenden Daten (in unserem Fall ganze Zahlen) werden in einem `long`-Feld bereitgestellt, das entsprechend initialisiert ist. Die Daten sollen *aufsteigend* sortiert werden.
- **(Laborübung)** Testen Sie Ihr Programm in drei verschiedenen Fallstudien:
  Fallstudie 1: `17, 11, 8, 2, 15, 4, 8, 10, 9, 14`
  Fallstudie 2: `17, 15, 14, 11, 10, 9, 8, 8, 4, 2`
  Fallstudie 3: `2, 4, 8, 8, 9, 10, 11, 14, 15, 17`
- **(Laborübung)** Erweitern Sie Ihr Programm, so dass die Anzahl Elementvergleiche sowie die Anzahl Elementvertauschungen ermittelt und gemeinsam mit dem sortierten Feld ausgegeben wird.
- Bei welcher anfänglichen Anordnung der zu sortierenden Feldelemente werden die **wenigsten** Elementvergleiche und Elementvertauschungen benötigt (*best case*)?
- Bei welcher anfänglichen Anordnung der zu sortierenden Feldelemente werden die **meisten** Elementvergleiche und Elementvertauschungen benötigt (*worst case*)?
- **(Laborübung)** In den obigen drei Fallstudien besteht das Feld aus jeweils 10 zu sortierenden Elementen, für die Problemgröße n gilt folglich $n = 10$. Bestimmen Sie für den *worst case* der Problemgrößen 2 bis 10 jeweils die Anzahl Elementvergleiche sowie die Anzahl Elementvertauschungen und protokollieren Sie die Ergebnisse in einer Tabelle. Nutzen Sie für jede der betrachteten Problemgrößen eine geeignete selbstgewählte Anfangsanordnung der zu sortierenden Elemente.

- Entwickeln Sie im *worst case* für die Anzahl Vergleiche eine Formel in Abhängigkeit von n und schätzen Sie die Anzahl Vertauschungen ebenfalls durch eine Formel ab.
- Bis auf wenige Fälle ist bei Selectionsort die Anzahl Vertauschungen bzw. Verschiebungen von Feldelementen deutlich niedriger als bei Insertionsort, hingegen werden aber zumeist mehr Vergleiche vorgenommen. Für welche Anwendungsszenarien bietet sich Selectionsort daher an?
- **(Laborübung)** Erweitern Sie Ihr Programm, so dass *absteigend* statt aufsteigend sortiert wird.

Aufgabe 4

Vergegenwärtigen Sie sich die Sortierverfahren *Insertionsort*, *Selectionsort*, *Mergesort* und *Bucketsort*. Charakterisieren Sie für jedes dieser Verfahren den *best case* und den *worst case*. Wir gehen davon aus, dass mit den Verfahren aufsteigend sortiert wird. Erschließen Sie aus den Verfahren Eigenschaften, Alleinstellungsmerkmale und spezifische Vorzüge im praktischen Einsatz, die einem Softwareentwickler helfen können, für einen konkreten Anwendungsfall das am besten geeignete Sortierverfahren auszuwählen.

Ein Sortierverfahren heißt *stabil*, wenn die Reihenfolge von Elementen mit gleichen Werten während des Sortierens stets erhalten bleibt. Welche der Verfahren sind stabil?