ulm university universität
uulm

Inst. für Angew. Informationsverarbeitung
Prof. Dr. Franz Schweiggert
Michaela Weiss
Wolfgang Kaifler

29.11.2010
Blatt 6

Systemnahe Software I (WS 2010/2011)

Abgabetermin: 07.12.2010

Aufgabe 1: Theorie (5 Punkte)

- Erklären Sie mit eigenen Worten, was ein Zeiger ist.
- Was liefert *&array[0]?
- Wie kann der Ausdruck *(array+i) bei dem int array[10] vereinfacht werden?

Aufgabe 2: Ausgabeverwirrung (5 + 2 + 2 Punkte)

```
1   #include <stdio.h>
2
3   void tu_was (int* p) {
4           *p = *p - 1;
5   }
6
7   int tu_nochmal_was (int a, int b) {
8           tu_was(&a);
9           return a * b;
10  }
11
12  int main () {
13          int array[] = {3,5,7,9,11};
14          int alt;
15
16          for (int i = ((sizeof array) / sizeof (int)); i >0; ) {
17                  tu_was(&i);
18                  alt = array[i];
19                  array[i] = tu_nochmal_was(array[i], i);
20                  printf("%d: alt = %d neu = %d\n", i, alt, array[i]);
21          }
22
23          return 0;
24  }
```

Quellcode 1: verwirrung.c

- Überlegen Sie sich, was das oben angegebene Programm ausgibt. Analysieren Sie hierzu zunächst, was die beiden Funktionen tu_was und tu_noch_was machen.
- Erklären Sie, zu welchem Seiteneffekt der Aufruf der Funktion tu_was führt und warum.
- Erklären Sie, warum der Aufruf der Funktion tu_was innerhalb der Funktion tu_noch_was zu keinen Änderungen an den von der main übergebenen Parametern führt.

Aufgabe 3: Vokabeltrainer (3 + 4 + 6 + 2 + 5 Punkte)

In dieser Aufgabe möchten wir schrittweise einen einfachen Vokabeltrainer bauen.

a.) Entwerfen Sie hierzu das struct „vokabel“, in dem das Wort in Deutsch und in Englisch abgelegt wird. Gehen Sie davon aus, das beide Wörter nicht mehr als 10 Buchstaben haben. Als boolescher Wert soll hier zudem gespeichert werden, ob die Vokabel zuletzt richtig eingegeben wurde.

b.) Schreiben Sie eine Funktion:
void einlesen (int anzahl, struct vokabel v[]) {...}
Diese soll die übergebene Anzahl von Vokabeln in den Vektor v einlesen. Die Anfangsbelegung des booleschen Wertes der „vokabel“ soll *false* sein. Das Einlesen der Vokabeln soll wie folgt aussehen:

```
1. Vokabel:
Deutsch: Hund
Englisch : dog

2. Vokabel:
Deutsch: Katze
Englisch: cat

3. Vokabel:
Deutsch: Maus
Englisch: mouse
```

c.) Schreiben Sie die Funktion
void training (struct vokabel v[], int anzahl, int level) {...},
welche die eingelesenen Vokabeln abfragt.
Der int-Wert *anzahl* stellt hierbei die Gesamtanzahl der eingelesenen und gespeicherten Vokabeln dar. *Level* bezeichnet das aktuelle Abfrage-Level, beginnend bei 1. *V* ist der Vektor, der die Vokabeln enthält.
Die Überprüfung der Antwort erfolgt per Stringvergleich case-sensitive, d.h. die Groß- und Kleinschreibung soll beachtet werden. Wurde ein Wort falsch eingegeben, soll das Programm den Anwender auf diesen Fehler hinweisen und die richtige Antwort ausgeben. War die Antwort richtig, soll auch dies dem Anwender mitgeteilt werden. Zusätzlich soll der boolesche Wert der Vokabel auf *true* gesetzt werden.
Sind alle Vokabeln 1x abgefragt, beginnt das nächste Level. Hier werden nur noch die Vokabeln getestet, die zuvor falsch beantwortet wurden.
Die stufenweise Abfrage endet, wenn im aktuellen Level alle abgefragten Antworten richtig eingegeben wurden. Merken Sie sich hierzu die Anzahl der falschen Antworten im vorherigen Level.

```
Level 1:
Hund: elefant
FALSCH! richtige Eingabe: dog
Katze: cat
RICHTIG!
Maus: Mouse
FALSCH! richtige Eingabe: mouse
- - - - - - - - - - - -
Level 2:
Hund: doog
FALSCH! richtige Eingabe: dog
Maus: mouse
RICHTIG!
- - - - - - - - - - - -
Level 3:
Hund: dogg
FALSCH! richtige Eingabe: dog
- - - - - - - - - - - -
Level 4:
Hund: dog
RICHTIG!
```

d.) Schreiben Sie das Programm trainer.c, das zuerst die Vokabeln einliest und danach abfragt. Nach der Vokabelabfrage soll sich das Programm vom Benutzer verabschieden und sich beenden.

e.) Erweitern Sie das struct *vokabel* um ein int-Feld, in dem abgespeichert wird, in welchem Level, diese Vokabel zuletzt abgefragt wurde. Ändern Sie ihre Trainingsfunktion so ab, dass der Vektor nicht mehr in der Form 0,1,2,3..., sondern zufallsgesteuert durchlaufen wird. Führen Sie hierzu exakt so viele random-gesteuerte Vokabelabfragen durch, wie es falsche Antworten im letzten Level gab. Die Anzahl dieser falschen Antworten können Sie der Funktion als zusätzlichen Parameter übergeben.
Wurde die Vokabel mit dem durch Zufall ausgewählten Index in diesem Level bereits abgefragt oder in einem früheren Level richtig beantwortet, inkrementieren Sie den Index bis Sie eine noch nicht abgefragte Vokabel erreichen. Erreichen Sie das Vektorende, fangen Sie wieder bei Index 0 an.

Hinweis:

- Strings können bei scanf mit Hilfe des Parameters %s eingelesen werden.
- In C existiert direkt kein Datentyp bool. Wahrheitswerte werden mit Hilfe von ganzen Zahlen dargestellt (0: false, alle anderen Zahlen : true). Im C99-Standard wurde jedoch der Datentyp _Bool eingeführt. Wenn Sie die Header-Datei <stdbool.h> einbinden, können Sie die aus anderen Programmiersprachen gewohnten Werte true und false verwenden. Beachten Sie, dass Sie hierzu den C99-Standard bei der Kompilierung einbinden müssen (-std=c99). Bsp.:

```
25  bool b = true, c = false;
26  if (b == true) {...}
```

Quellcode 2: bool.c

- Zufallszahlen können in C mit Hilfe der Funktion *int rand()* aus <stdlib.h> erzeugt werden. Möchten Sie ganzzahlige Zufallszahlen zwischen 0 und N erzeugen, verwenden Sie *rand()%(N+1)*.

Viel Erfolg!