Institut für Informatik

der Ludwig-Maximilians-Universität München

Systempraktikum – Wintersemester 2011/2012

Prof. Dr. Dieter Kranzlmüller
Dr. Nils gentschen Felde, Stephan Reiter, Johannes Watzl

Blatt 3— Grundlagen II: Modularisierung, Strukturen, Adressarithmetik

Abgabedatum theor. Aufgaben	Abgabedatum prakt. Aufgaben	Deadline Projektaufgaben
06.11.	06.11.	—

Theoretische Aufgaben (Blatt 3)

Aufgabe T-3-1

In dieser Aufgabe sollen Sie sich mit den Konzepten Bindung, Speicherdauer und Sichtbarkeit beschäftigen.

a. Welche Arten der Bindung (von Objekten/Variablen) gibt es in C? Schreiben Sie ein Beispielprogramm, an dem Sie die unterschiedlichen Arten von Bindung verdeutlichen.

b. Welche Arten der Speicherdauer kennen Sie? Verdeutlichen Sie wieder an einem selbst gewählten Beispielprogramm.

c. Im Zusammenhang mit Sichtbarkeit (scope) sind lokale und globale Objekte sowie das Konzept der Überschattung von Bedeutung. Denken Sie sich ein Programmbeispiel aus, das alle diese Aspekte beinhaltet.

Aufgabe T-3-2

a. Erklären Sie die Zusammenhänge zwischen der Deklaration und der Definition von Objekten in C.

b. Geben Sie zu jedem der angegebenen C-Fragmente an, ob es sich um eine Definition oder Deklaration handelt:

(a) `int a;`
(b) `int f(int x) { return x+1; }`
(c) `struct S;`
(d) `int f(int);`
(e) `enum { up, down };`
(f) `extern int a;`
(g) `extern const int c = 1;`
(h) `extern const int c;`

[0] Stand: 13. Oktober 2011

Aufgabe T-3-3

Übersetzen Sie das nachfolgende C-Programm, und führen Sie es aus. Erklären Sie für jede der zwölf Zeilen der Ausgabe, wie diese zu Stande kommt.

```
/* Programm für Aufgabe T-2-3 */

#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[]) {

    char buf1[4][10] = {"eins", "zwei", "drei"};
    char buf2[] = "vier";
    char buf3[] = { "Hallo \0 Welt!" };
    int z = 1;

    printf("(1)  Was ist das?: %i\n", argc);
    printf("(2)  Was ist das?: %s\n", argv[0]);
    if (*(buf3+15) == (int) NULL) {
      printf("(3)  Was ist das?: %s\n", buf3);
      buf3[15] = buf3[14];
    }
    printf("(4)  Was ist das?: %s\n", buf3);
    printf("(5)  Was ist das?: %i\n", *buf1[0]);
    printf("(6)  Was ist das?: %i\n", *buf1[1]);
    printf("(7)  Was ist das?: %i\n", *buf1[2]);
    printf("(8)  Was ist das?: %c\n", (int) sizeof(buf2)+50 );
    printf("(9)  Was ist das?: %+ld\n", (long int) sizeof(buf2) );
    printf("(10) Was ist das?: %d\n", 'z');
    printf("(11) Was ist das?: %d\n", (int) "z");
    printf("(12) Was ist das?: %d\n", (int) "helloWorld");
    return EXIT_SUCCESS;
}
```

Praktische Aufgaben (Blatt 3)

Aufgabe P-3-1

Schreiben Sie ein C-Programm, das mathematische Funktionen zur Verfügung stellt und die Ergebnisse der Berechnungen am Bildschirm ausgibt. Implementieren Sie die Funktionen:

a. Fakultät: Berechnung von $n!$

b. Fibonacci-Zahl: Berechnung der n-ten Fibonacci-Zahl

Die Argumente und Ergebnisse der Funktionen sollen dabei die natürlichen Zahlen sein. Die drei Funktionen sind jeweils iterativ und rekursiv zu implementieren. Für die iterative Variante sollen dabei jeweils drei verschiedene Versionen hinsichtlich der Parameterübergabe erstellt werden:

1. by-value
2. by-reference mit `const`-deklarierten Parametern.
3. by-reference **ohne** `const`-deklarierte Parameter.

Bei der dritten Variante könnten Sie dabei, abhängig davon, wie Sie Ihre Funktion implementieren, gefährliche Seiteneffekte erzeugen. Dies tritt auf, wenn Sie über die Referenz (Pointer) die „Original"-Variable in der aufrufenden Routine verändern. Lassen Sie sich deshalb auch nach jedem Aufruf die Werte der Eingabe-Parameter ausgeben. Vermeiden Sie diese Seiteneffekte.

Die Auswahl der zu berechnenden Funktion soll durch Übergabe als Kommandozeilenparameter durch `fak` oder `fib`, gefolgt von dem numerischen Argument erfolgen. Zum Beispiel: `prog-2-1 fib 10`.

Achten Sie beim Ergänzen des Programmrahmens auf eine Fehlerbehandlung. Überlegen Sie sich dazu, welche Fehler (auch hinsichtlich der mathematischen Operationen) auftreten können. **Bitte denken Sie an ein Makefile!**

```
/* Rahmen für Aufgabe P-2-1 */

#include <???>              /* Header-Datei fuer die Ein-/Ausgabe */
#include <???>              /* Funktionen zur Stringbehandlung    */

/* Hier gut kommentieren! */
long fak_rec_bv(const long x) {
   ???
}

/* Hier gut kommentieren! */
long fak_it_bv(const long x) {
   ???
}

/* Hier gut kommentieren! */
long fak_it_br1(const long *x) {
   ???
}

/* Hier gut kommentieren! */
long fak_it_br2(long *x) {
   ???
}


/* Hier gut kommentieren! */
long fib_rec_bv(const long x) {
   ???
}

/* Hier gut kommentieren! */
long fib_it_bv(const long x) {
   ???
}

/* Hier gut kommentieren! */
long fib_it_br1(const long *x) {
   ???
}

/* Hier gut kommentieren! */
long fib_it_br2(long *x) {
   ???
}


int main(int argc, char* argv[]) {
   long zahl = 0;

   if (argc == ??? && strcmp(???, "fak") == 0) {
      sscanf(???, "%ld", ???);
      printf("Vor␣dem␣Funktionsaufruf:␣zahl:␣%ld\n", zahl);

      printf("fak_rec_bv()␣ergibt␣%ld\n", fak_rec_bv(???));
      printf("Nach␣dem␣Funktionsaufruf:␣zahl:␣%ld\n", zahl);
```

```
      printf("fak_it_bv() ergibt %ld\n", fak_it_bv(???));
      printf("Nach dem Funktionsaufruf: zahl: %ld\n", zahl);

      printf("fak_it_br1() ergibt %ld\n", fak_it_br1(???));
      printf("Nach dem Funktionsaufruf: zahl: %ld\n", zahl);

      printf("fak_it_br2() ergibt %ld\n", fak_it_br2(???));
      printf("Nach dem Funktionsaufruf: zahl: %ld\n", zahl);
   }

   else if (argc == ??? && strcmp(???, "fib") == 0) {
      sscanf(???, "%ld", ???);
      printf("Vor dem Funktionsaufruf: zahl: %ld\n", zahl);

      printf("fib_rec_bv() ergibt %ld\n", fib_rec_bv(???));
      printf("Nach dem Funktionsaufruf: zahl: %ld\n", zahl);

      printf("fib_it_bv() ergibt %ld\n", fib_it_bv(???));
      printf("Nach dem Funktionsaufruf: zahl: %ld\n", zahl);

      printf("fib_it_br1() ergibt %ld\n", fib_it_br1(???));
      printf("Nach dem Funktionsaufruf: zahl: %ld\n", zahl);

      printf("fib_it_br2() ergibt %ld\n", fib_it_br2(???));
      printf("Nach dem Funktionsaufruf: zahl: %ld\n", zahl);
   }

   else {
      ???
   }
   return(0);
}
```

Aufgabe P-3-2

a. Ändern Sie Ihr Programm aus Aufgabe P-3-1, und erstellen Sie zwei Module: ein iteratives Modul und ein rekursives Modul. Das iterative-Modul soll die iterativen Funktionen enthalten. Das rekursive-Modul soll die rekursiven Funktionen enthalten. Erstellen Sie für die zwei Module Header-Dateien und Implementierungs-Dateien. Sie benötigen außerdem ein `main`-Modul für die `main()`-Funktion. Modifizieren Sie das Makefile so, dass es das iterative und das rekursive Modul beim Übersetzen automatisch zum Hauptprogramm bindet.

b. Fassen Sie das iterative und das rekursive Modul aus der vorherigen Teilaufgabe jetzt zu einer Bibliothek namens `libMyMath.a` zusammen, und verwenden Sie diese Bibliothek bei der Übersetzung des Hauptprogramms. Informieren Sie sich dazu über das Kommando `ar`. Passen Sie ggf. das Makefile an.

c. Vergleichen Sie die Größe der resultierenden Programme. Um welchen Bibliotheks-Typ handelt es sich?

Aufgabe P-3-3

Aus der Vorlesung kennen Sie C-Strukturen. Sie werden mit dem Schlüsslwort `struct` definiert und haben eine gewisse Ähnlichkeit mit Tupeln in einer relationalen Datenbank.

a. Erstellen Sie ein C-Programm, wie in den folgenden Teilschritten beschrieben:

   - Definieren Sie den Strukturtypen `struct student` und geben Sie ihm (mindestens vier) sinnvolle Attribute (z.B. Matrikelnummer als Integer, Nachname als String, ...).
   - Schreiben Sie eine Funktion `getMatrNr()` mit der folgenden Signatur: `int getMatrNr(struct student s)`. Ihr wird ein Objekt vom Strukturtyp `student` übergeben.

Der Rückgabewert ist die Matrikelnummer des jeweiligen Studenten oder -1 im Fehlerfall. **Verwenden Sie zum Zugriff auf ein Feld die Punkt-Notation, also z.B. `s.matrnr`.**

- Schreiben Sie eine Funktion `setMatrNr()` mit der folgenden Signatur: `int setMatrNr(struct student s, int mnr)`. Ihr wird ebenfalls ein Objekt vom Strukturtyp `student` und ein Integer-Wert übergeben. Die Funktion soll die Matrikelnummer des jeweiligen Studenten auf den übergebenen Wert setzen und im Erfolgsfall 0 zurückgeben. **Verwenden Sie zum Manipulieren eines Feldes wieder die Punkt-Notation, also z.B. `s.matrnr = mnr;`.**
- Schreiben Sie jetzt die `main()`-Funktion. In ihr soll mindestens ein Objekt des Strukturtyps `student` erzeugt werden. Weisen Sie den Attributen dieses Objekts sinnvolle Werte zu, ohne die Funktion `setMatrNr()` zu verwenden. Lassen Sie sich den Wert der Matrikelnummer durch Aufruf der Funktion `getMatrNr()` auf der Konsole ausgeben.
- Erstellen Sie ein Makefile, und übersetzen und testen Sie Ihr Programm.

b. Fügen Sie jetzt weitere Zeilen zur `main()`-Funktion hinzu: Versuchen Sie, den Wert der Matrikelnummer durch Aufruf der Funktion `setMatrNr()` neu zu setzen. Testen Sie durch ein erneutes `getMatrNr()`. Warum funktioniert das so nicht?

c. Beheben Sie dieses Problem. Wie müssen Sie die Funktionssignaturen ändern? Welche weiteren Notationen müssen Sie ändern? Testen Sie Ihr Programm!

Aufgabe P-3-4

Schreiben Sie ein Programm, welches die ersten 255 Bytes einer Datei auf dem Bildschirm ausgibt. Falls die Datei kleiner als 255 Bytes ist, soll ihr vollständiger Inhalt ausgegeben werden. Der Dateiname der auszugebenden Datei soll dabei als Kommandozeilenparameter beim Aufruf des Programms übergeben werden.

Bitte denken Sie an ein Makefile!

Aufgabe P-3-5

Modifizieren Sie das Programm aus Aufgabe P-3-4 so, dass es die ersten 255 Bytes der auszugebenden Datei (bzw. bei kleineren Dateien wieder den vollständigen Inhalt) **in umgekehrter Reihenfolge** auf dem Bildschirm ausgibt. Kopieren Sie dazu den Inhalt des Lesepuffers in ein Array, dessen Inhalt Sie dann beginnend mit dem letzten Element zeichenweise ausgeben (`for`-Schleife). Verwenden Sie **keine** Funktionen aus `string.h`.

Bitte denken Sie an ein Makefile!