Grundlagen der Informatik III

Wintersemester 2010/2011

Wolfgang Heenes, Patrik Schmittat

TECHNISCHE UNIVERSITÄT DARMSTADT

4. Aufgabenblatt mit Lösungsvorschlag

22.11.2010

Hinweis: Der Schnelltest und die Aufgaben sollen in den Übungsgruppen bearbeitet werden. Die Hausaufgaben sind in der Kalenderwoche 48 (29.11. bis 03.12.) bei den Tutoren in **physikalischer Form** (handschriftlich oder gedruckt) abzugeben. Bei allen Abgaben ist der Name des Tutors und die Übungsgruppe deutlich anzugeben. Bei Teamabgaben wird nur eine Lösung eingereicht, die alle Namen der Teammitglieder enthält.

Schicken Sie Ihre Lösungen von Programmieraufgaben *zusätzlich* zur schriftlichen Abgabe per E-Mail an Ihren Tutor. Kommentieren Sie Ihren Quellcode.

Aufgabe 1: Schnelltest

Fragen	***Antworten***
1. *Welche der folgenden Aussagen über die Speicherverwaltung in C sind korrekt?*	☐ *Lokale Variablen in Funktionen müssen explizit wieder freigegeben werden.* ☐ *Mit dem Befehl* `malloc` *wird Speicher auf dem Stack reserviert.* ■ *Der mit* `malloc` *angeforderte Speicher muss wieder explizit freigegeben werden.* ☐ *Der Garbage-Collector gibt periodisch nicht mehr verwendeten Speicher frei.* ☐ *Der Angeforderte Speicher wird automatisch mit Nullen initialisiert.*
2. *Welche Aussagen über Funktionen und Funktionsparameter in C sind richtig?*	■ *Funktionsparameter werden als Kopie übergeben (call-by-value).* ☐ *Eine Referenzübergabe von Parametern (call-by-reference) ist nicht möglich.* ■ *Eine Referenzübergabe von Parametern ist mittels Pointer möglich.* ■ *Die Übergabe von Arrays erfolgt über Pointer.*
3. *Welche der folgenden Behauptungen über Längen von Arrays in C sind zutreffend?*	☐ *Die Funktion* `sizeof` *ermittelt die Länge eines Arrays.* ☐ *Die Funktion* `sizeof` *ermittelt die Größe des Speichers, den ein Array belegt.* *(Bei statischen Arrays ja, sonst nein.)* ☐ *Die Funktion* `length` *ermittelt die Länge eines Arrays.* ■ *Es gibt keine Funktion zur Ermittlung der Länge eines Arrays.* ☐ *Keine dieser Aussagen ist zutreffend.*
4. *Betrachten Sie die folgende C-Deklaration:* `int a[5], *p;`	☐ *Innerhalb ihres Scopes können die Variablen* `a` *und* `p` *beliebig ausgetauscht werden, ohne dass der Compiler einen Fehler meldet.* ■ `a` *und* `p` *sind nicht zuweisungskompatibel (*`a = p` *erzeugt einen Compilerfehler).* ■ *Die Zuweisung* `a[0] = *p;` *wird vom Compiler akzeptiert.* ■ *Die Zuweisung* `*a = p[0];` *wird vom Compiler akzeptiert.* ■ *Die Schreibweise* `*p` *ist äquivalent zu* `p[0]`*.*
	Fortsetzung nächste Seite...

Fragen	*Antworten*
5. *Welche Aussagen zu boole'schen Ausdrücken in C ist korrekt?*	☐ *In C sind der Datentyp* `bool` *sowie die Ausdrücke* `true` *und* `false` *vorhanden.* ■ *Alle Zahlenwerte größer null stellen implizit den logischen Wert true dar.* ☐ *Alle Zahlenwerte kleiner gleich null stellen implizit den logischen Wert false dar.* ■ *Zuweisungen von Zahlenwerten können direkt in logischen Ausdrücken verwendet werden, um die zugewiesene Zahl auf Ungleichheit mit null zu prüfen.*

Aufgabe 2: Informationsdarstellung in C

Schreiben Sie ein C Programm (ANSI-C), welches eine einzugebene Dezimalzahl als Oktal-, Hexadezimalzahl bzw. als ASCII-Zeichen ausgibt. Bevor die Dezimalzahl eingegeben wird, soll der Benutzer auswählen, welche Darstellung gewünscht wird. Das Auswahlmenü verwendet O für Oktal, H für Hexadezimal und A für ASCII. Die Auswahl soll **nicht** *sensitiv auf Groß-/Kleinschreibung reagieren.*

Lösungsvorschlag:

```
#include <stdio.h>
#include <ctype.h>

main()
{
  char eingabe;
  int zahl;

  printf("\nWaehlen␣Sie␣(O)ktal,␣(H)ex␣oder␣(A)SCII␣>␣");
  eingabe = getchar();

  printf("\nBitte␣Dezimalzahl␣eingeben:␣");
  scanf("%i",&zahl);

  switch (toupper(eingabe))
  {
    case 'O':
      printf("Dezimal␣%i␣=␣Oktal␣%o␣\n", zahl, zahl);
    break;
    case 'H':
      printf("Dezimal␣%i␣=␣Hexadezimal␣%x␣\n", zahl, zahl);
    break;
    case 'A':
      if(zahl <= 255)
        printf("ASCII−Code␣%i␣entspricht␣%c␣\n", zahl, zahl);
      else
        printf("Keine␣gueltige␣Zahl\n");
    break;
  }
}
```

Aufgabe 3: Monte-Carlo-Simulation

Monte-Carlo-Simulationen können zur Berechnung des Inhalts regel- und unregelmäßiger Flächen und Körper verwendet werden. Die Algorithmen werden als Monte-Carlo-Algorithmen bezeichnet und gehören in die Klasse der randomisierten Algorithmen. Das Verfahren der Monte-Carlo-Simulation soll anhand der probabilistschen Berechnung der Kreiszahl π verdeutlicht

werden. Hierzu werden zufällige Punkte $\{(x,y)|x \in [-1,1], y \in [-1,1]\}$ *generiert und überprüft, ob diese innerhalb des Einheitskreises liegen. Die sich ergebende Wahrscheinlichkeitsverteilung erlaubt die Berechnung der Kreiszahl* π *nach folgender Formel.*

$$\frac{\textit{Treffer in Kreisflaeche}}{\textit{generierte Punkte im Rechteck}} = \frac{r^2 \cdot \pi}{(2 \cdot r)^2} = \frac{\pi}{4}$$

a) *Schreiben Sie ein C Programm, welches die Kreiszahl* π *berechnet und ausgibt. Die Funktion rand() liefert Zufallszahlen.*[1]

Lösungsvorschlag:

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

main()
{
  long in_circle = 0;
  long schranke = 100000000;
  long i;

  float x, y, pi;

  srand(time(NULL));

  for (i = 0; i < schranke; i++)
  {
    /*
      x und y sollen Zufallszahlen zwischen 0 und 1 sein
    */
    x = rand() / (float)RAND_MAX;
    y = rand() / (float)RAND_MAX;

    /*
      Quadrat des Abstands des Punktes vom Ursprung nach Pytagoras.
      Wenn der Abstand <= 1 ist, liegt der Punkt im Kreis.
    */
    if (x * x + y * y <= 1)
      in_circle++;
  }

  pi = in_circle * 4 / (float) schranke;
  printf("Die Kreiszahl pi ist: %f\n", pi);
}
```

b) *Testen Sie Ihr Programm mit unterschiedlicher Anzahl von Punkten (im Bereich von ca. 100000 bis 100000000). Notieren Sie die Anzahl der Punkte und die Genauigkeit in einer Tabelle.*

Lösungsvorschlag:

Die Tabelle stellt jeweils zwei Ausführungen des Programms dar.

[1] Binden Sie falls notwendig die entsprechenden Librarys ein. Achten Sie bei rand() darauf vorher einen zufälligen Seed zu setzen.

Anzahl der Tests	1. Versuch Wert	Fehler	2. Versuch Wert	Fehler
100000	3.138440	-0.003152	3.143960	0.002367
1000000	3.143972	0.002379	3.143204	0.001611
10000000	3.141396	-0.000196	3.141780	0.000187
100000000	3.141675	0.000082	3.141606	0.000013

c) *Die Berechnung von π kann auch auf ein Flächenintegral zurückgeführt werden.*

$$\int_0^1 \frac{4}{1+x^2} dx$$

Schreiben Sie ein C Programm, welches die Integration (z. B. Trapezintegration) ausführt und vergleichen Sie die Ergebnisse mit b).

Lösungsvorschlag:

```
#include <stdio.h>
#include <math.h>

static long num_steps = 100000;
double step;

main()
{
  int i;
  double x, pi, sum = 0.0;

  step = 1.0 / (double) num_steps;
  for (i = 1; i <= num_steps; i++)
  {
    x = (i - 0.5) * step;
    sum = sum + 4.0 / (1.0 + x * x);
  }

  pi = step * sum;
  printf("PI␣%lf\n", pi);
}
```

Schrittweite	Wert	Fehler
1	3.200000	0.058407
10	3.142426	0.000833
100	3.141601	0.000008
1000	3.141593	0.000000

Der Vergleich der Tabellen zeigt, dass das Ergebnis bei der Integration deutlich schneller zum „richtigen“ Ergebnis führt.

Aufgabe 4: Inline-Assembler-Code in C

Assembler-Programme werden häufig zur Optimierung von Programmteilen eingesetzt, die in einer Hochsprache z. B. C vorliegen. Um den Code nicht vollständig neu zu schreiben, erlauben es moderne Compiler sogenannten Inline-Assembler-Code in ein Programm einzubinden.

Ein einfaches Beispiel zeigt das folgende C Programm:

```
#include <stdio.h>

main()
{
  int n = 33;

  asm("imull␣%%eax;"
      : "=a" (n)
      : "a" (n)
  );

  printf("Das␣Quadrat␣von␣33␣ist␣%i\n", n);
}
```

Das Programm berechnet das Quadrat der Zahl 33 und gibt das Ergebnis auf dem Bildschirm aus.

Inline-Assembler-Code wird mit dem Schlüsselwort **asm** *eingebettet.*

```
asm ( assembler template
           : output operands                    (optional)
           : input operands                     (optional)
           : list of clobbered registers        (optional)
    );
```

Der Assembler-Code (hier als assembler template) bzw. die Befehle werden von " " umschlossen. Für die korrekte Funktionsweise des Inline-Assembler-Code müssen die Variablen von C den zu verwendenden Registern zugeordnet werden. Dies geschieht mit durch die Angabe von output- und input operands.

Die Zuweisung an die Register wird über die Angabe folgender Bezeichner erreicht.

a	*%eax*
b	*%ebx*
c	*%ecx*
d	*%edx*
S	*%esi*
D	*%edi*

Die Angabe (Syntax beachten) im obigen Programm stellt also eine Zuweisung der C Variablen n an das Register %eax da. Achtung die Register werden im Inline-Assembler-Code mit %%eax usw. bezeichnet.

a) *Zur Vorbereitung der Hausaufgabe überzeugen Sie sich von der korrekten Ausführung des C Programms mit Inline-Assembler-Code.*

Hausaufgabe 1: Inline-Assembler-Code in C, Leistungsvergleich (5 Punkte)

Aufbauend auf Aufgabe 3 sollen zwei Implementierungen verglichen werden. Betrachtet wird folgendes C Programm.

```
#include <stdio.h>

main()
{
  int i;
  int result = 1;
  int n = 12;
  for (i = 2; i <= n; i++)
    result *= i; // (result = result * i)

  printf("Fakultaet␣%d\n", result);
}
```

a) *Schreiben Sie ein IA32-Assemblerprogramm welches obiges C Programm realisiert. Testen Sie ihr Programm für n = 12.*

Lösungsvorschlag:

```
.data
intout:  .string "Ergebnis␣%d␣\n"
n:       .long 5

.text
.globl main

main:
     movl n, %ecx  # ecx enthaelt n
     movl $1, %eax # eax entspricht result
.L2:
     imull %ecx
     loop .L2

.ende:
     pushl %eax    # Wert im %eax ausgeben
     pushl $intout
     call printf

# Exit
movl $1, %eax
int $0x80
```

b) *Ersetzen Sie in dem vorgegebenen C Programm die Berechnung durch Ihr IA32-Assemblerprogramm. Dazu soll der in Übung 4 vorgestellte Rahmen für Inline-Assembler verwendet werden.*[2] *Überzeugen Sie sich von der korrekten Funktion des Programms für n = 12.*

Lösungsvorschlag:

```
#include <stdio.h>

main()
{
  int result = 1;
  int n = 12;

  asm(".m1:␣imull␣%%ecx;"
      "loop␣.m1;"
      : "=a" (result)
      : "c" (n), "a" (result)
  );

  printf("Fakultaet␣%d\n", result);
}
```

c) *Vergleichen Sie die Laufzeiten der beiden Programme. Dazu bietet es sich an, die Berechnung in eine for-Schleife einzubetten und z. B. die Berechnung 100000000 zu wiederholen (das Programm sollte schon 3 - 4 Sekunden laufen). Die Zeitmessung können Sie z. B. unter Unix mit* `time` *vornehmen. Sollte Ihr IA32-Assemblerprogramm langsamer laufen, sollten Sie sich noch ein paar Gedanken über Ihre Implementierung machen und diese optimieren.*

[2] Informationen finden sich auch unter `http://www.ibm.com/developerworks/linux/library/l-ia.html`

Lösungsvorschlag:

Das C Programm wurde mit gcc **ohne** Optimierungen übersetzt und beide Berechnungsmethoden 100000000 mal aufgerufen.

C Programm	IA32 Assemblerprogramm
3.131 s	2.380 s

Hausaufgabe 2: Reverse Engineering (5 Punkte)

Gegeben ist folgender Object Dump:

```
080483a4 <up>:
 80483a4: 55                      push   %ebp
 80483a5: 89 e5                   mov    %esp,%ebp
 80483a7: 83 ec 18                sub    $0x18,%esp
 80483aa: 83 7d 08 00             cmpl   $0x0,0x8(%ebp)
 80483ae: 75 09                   jne    80483b9 <up+0x15>
 80483b0: c7 45 ec 00 00 00 00    movl   $0x0,-0x14(%ebp)
 80483b7: eb 36                   jmp    80483ef <up+0x4b>
 80483b9: 83 7d 08 01             cmpl   $0x1,0x8(%ebp)
 80483bd: 75 09                   jne    80483c8 <up+0x24>
 80483bf: c7 45 ec 01 00 00 00    movl   $0x1,-0x14(%ebp)
 80483c6: eb 27                   jmp    80483ef <up+0x4b>
 80483c8: 8b 45 08                mov    0x8(%ebp),%eax
 80483cb: 83 e8 02                sub    $0x2,%eax
 80483ce: 89 04 24                mov    %eax,(%esp)
 80483d1: e8 ce ff ff ff          call   80483a4 <up>
 80483d6: 89 45 fc                mov    %eax,-0x4(%ebp)
 80483d9: 8b 45 08                mov    0x8(%ebp),%eax
 80483dc: 83 e8 01                sub    $0x1,%eax
 80483df: 89 04 24                mov    %eax,(%esp)
 80483e2: e8 bd ff ff ff          call   80483a4 <up>
 80483e7: 8b 55 fc                mov    -0x4(%ebp),%edx
 80483ea: 01 c2                   add    %eax,%edx
 80483ec: 89 55 ec                mov    %edx,-0x14(%ebp)
 80483ef: 8b 45 ec                mov    -0x14(%ebp),%eax
 80483f2: c9                      leave
 80483f3: c3                      ret

080483f4 <main>:
 ...
 80483fe: 55                      push   %ebp
 80483ff: 89 e5                   mov    %esp,%ebp
 8048401: 51                      push   %ecx
 8048402: 83 ec 24                sub    $0x24,%esp
 8048405: c7 45 f4 14 00 00 00    movl   $0x14,-0xc(%ebp)
 804840c: 8b 45 f4                mov    -0xc(%ebp),%eax
 804840f: 89 04 24                mov    %eax,(%esp)
 8048412: e8 8d ff ff ff          call   80483a4 <up>
 ...
```

a) *Analysieren Sie den Object Dump. Was für eine Funktion wird realisiert.*

Lösungsvorschlag:

Rekursive Berechnung der Fibonacci-Zahlen.

b) *Implementieren Sie das Programm in C und testen Sie die Funktionsweise.*

Lösungsvorschlag:

```
#include <stdio.h>

int up(int n)
{
  int x;
  if(n == 0)
  {
    return 0;
  }
  else if(n == 1)
  {
    return 1;
  }
  else
  {
    x = up(n - 2);
    return x + up(n - 1);
  }
}

int main(void)
{
  int n = 20;
  int erg;

  erg = up(n);

  printf("Ergebnis␣%d\n", erg);

  return 0;
}
```