Prof. Dr. K. Wüst
FH Gießen–Friedberg, FB MNI

WS2010/11
Studiengang Informatik

Klausur zu Konzepte Systemnaher Programmierung
Musterlösung mit Erläuterungen

10.2.2011

Nachname:	
Vorname:	
Matrikelnummer:	Kennung:

Bitte die Lösungen zwischen den Aufgabenstellungen einfügen. Wenn ein zusätzliches Blatt nötig ist, bitte leserlich Namen und Matrikelnummer darauf schreiben!

Bitte ankreuzen/ausfüllen: Ich habe die beiden Hausübungen

- in diesem Semester (WS 2010/11) abgegeben und bestanden.
- in Semester ..... bei Prof. ..... abgegeben und bestanden.

Punkteverteilung

Aufgabe	Punkte	erreicht
1	18	
2	16	
3	18	
4	14	
5	16	
6	18	
Summe	100	

Aufg.1)
Schreiben Sie die nachfolgenden Definitionen in C-Syntax:

1) Eine ganze Zahl

```
int i;
```

2) Ein Array von fünf vorzeichenlosen ganzen Zahlen.

```
unsigned int j[5]
```

3) Ein Array von 5 Zeigern auf 8-Bit-Zeichen.

```
char *pc[5]
```

4) Ein Array von 5 Zeigern auf Strukturen, die aus einer doppelt genauen Gleitkommazahl und einer Ganzzahl bestehen. (Definition darf auch aus mehreren Statements bestehen.)

```
struct zweizahlen {
double dzahl;
int izahl;
};

struct zweizahlen *zahlenfeld[5];
```

Es sind auch andere Lösungen möglich, z.B. mit „typedef“.

Aufg.2)
a) Welchen Wert hat die Variable i nach der Ausführung der folgenden Codezeile?

```
i = ((8.5==17/2)? 4 : 5);
```

Bei der Ganzzahldivision wird der Rest ignoriert, ebenso wie in der NinjaVM beim div-Befehl. Das Ergebnis von 17/2 ist also 8. Der nachfolgende Vergleich mit 8.5 ergibt also 0 (false). Deshalb gibt der bedingte Ausdruck („Fragezeichen-Doppelpunkt-Operator") den zweiten Antwortteil (hinter dem Doppelpunkt) zurück. Die richtige Antwort ist also „5".

b) Betrachten Sie das folgende Programmstück und geben Sie an, wie groß der Wert von z nach diesem Programmabschnitt ist. Machen Sie einen Änderungsvorschlag, so dass das Makro in allen Situationen funktioniert.

```
#define einsmehr(x) x+1

y=3;
z=einsmehr(y)*3;
```

Der Präprozessor macht eine einfache Textersetzung, statt `einsmehr(y)` *wird dann an der gleichen Stelle* `y+1` *stehen. Das Statement lautet also* `z=y+1*3;` *und da die Multiplikation Vorrang vor der Addition hat, weist es der Variablen z den Wert 6 zu;*

Wenn man in dem Makro Klammern setzt funktioniert es so, wie der Name vermuten lässt: `#define einsmehr(x) (x+1)`

c) Betrachten Sie das folgende Programmstück. Geben Sie an welches Ergebnis für result man erwarten kann. Wo liegt die Problematik einer solchen Programmierung?

```
value=1;
result = (value++ * 5) + (value++ *3);
```

*Da der Inkrementierungsoperator „++" nachgestellt ist (Postfix), wird das Inkrement (value=value+1) erst ausgeführt, nachdem value verwendet wurde. Das Ergebnis für result ist (zumindest bei manchen Compilern) also schlicht „(1*5)+(1*3)=8". Die Problematik liegt darin, dass das Postinkrement eventuell nach Auswertung des ersten Teilausdrucks schon ausgeführt wird. Dann wäre das Ergebnis „(1*5)+(2*3)=11". Tauscht der Compiler die beiden Teilausdrücke, wird das Ergebnis „(2*5)+(1*3)=13". Diese Überlegungen zeigen, dass man so nicht programmieren sollte. Klarer wäre:*

```
value=1;
result = (value * 5) + (value *3);
value++;
```

Aufg.3)
In einem C-Programm wird der folgende Codeabschnitt ausgeführt:

```
int A[5];

int *p;
p=A;

*p++=1;
*p=4;
*(p+2)=*p-2;
*p--=3;
A[2]=5;
```

a) Geben Sie den Inhalt des Arrays A an (alle Elemente).

A = [1, 3, 5 ,2 ,unbelegt(Zufallswert)]

b) Auf welches Element verweist der Pointer p nun?

Auf Element A[0];

Aufg.4)
Welche Fehler entdecken Sie in den folgenden Programm? Welche Art sind die Fehler, wann und wie wirken sie sich aus?

```
struct person {
char Name[80];
int Mnummer;
};
struct person *zf;

int main(int argc, char *argv[]) {
  zf->Mnummer=123.0;
  (zf->Name)[8]="x";
  printf("%c \n",zf->Name);
  return 0;
}
```

- *Auf das Feld Mnummer wird eine Fließkommazahl zugewiesen, der Compiler warnt vor dem möglichen Datenverlust bei der Typumwandlung. Richtig wäre* `zf->Mnummer=123;`
- *Es wird versucht, auf das achte Element eines arrays vom typ char einen Zeiger auf einen String zuzuweisen, der Compiler warnt wegen der bedenklichen Referenzierung. Richtig wäre* `(zf->Name)[8]='x';`.
- *Bei der Ausgabe mit printf wird kein String ausgegeben. Für eine Stringausgabe müsste der Formatbezeichner „%s“ sein. Dann müsste aber „Name“ zunächst mit einem Zero-terminierten String beschrieben werden. (Kein Warning aber unerwartetes Ergebnis))*
- *Der schwerwiegendste Fehler: Es ist nur eine Variable zf definiert, die einen Zeiger auf eine Struktur vom Typ person hält. Diesem Zeiger wurde kein gültiger Wert zugewiesen. Für die Struktur selbst ist übrigens auch keine Variable angelegt und somit kein Speicherplatz reserviert. Das Ergebnis ist ein sofortiger „Programmabsturz“, das Betriebssystem beendet den Task wegen Speicherschutzverletzung.*

Aufg.5)
Betrachten Sie die folgenden beiden Funktionsdeklarationen:

```
void f1(int a, int b);
void f2(int *a, int *b);
```

Wo liegt der Unterschied der beiden Deklarationen? Wovon hängt es ab, welche der beiden Formen man wählen sollte?

Die erste Form übergibt die Kopien zweier int-Werte. Die Funktion f1 kann damit diese nach Belieben verändern, es erfolgt keine Veränderung der Originalwerte der Argumente (call bei value).

Die zweite Form übergibt Zeiger auf die Originalwerte der Argumente. Damit hat die Funktion f2 Zugriffe auf die Originalwerte der übergebenen Argumente und kann diese auch verändern. (call by reference)

Man wird die zweite Form nur wählen, wenn es notwendig ist, die übergebenen Argumente verändert (auf Ihre Originale) zurückzuschreiben.

Aufg.6)
Schreiben Sie ein Codestück in Ninja-Assembler, das auf dem Heap ein Objekt mit zwei Instanzvariablen (Datenfeldern) anlegt und in die beiden Instanzvariablen Verweise auf die primitiven Datenobjekte „12“ (Instanzvariable Nr.0) und „13“ (Instanzvariable Nr.1) legt. Da jedes putf eine Referenz auf das Objekt und eine auf den primitiven Datentyp „verbraucht“, wird die Objektreferenz zunächst dupliziert.

```
new 2
dup
pushc 12
putf 0
pushc 13
putf 1
```