Lösung Klausur 1a sengT

max. Punkte: 26 Note: (5/24) * P + 1

1. Begriffe (6 Pkte.)
Erklären Sie folgende Begriffe im Zusammenhang mit der Programmiersprache C:

a) Promotion Bei Operationen mit Werten unterschiedlichen Typs wird implizite in den genausten beteiligten Typ gecastet. Zudem werden nicht in allen modifizierten Typen Berechnungen vorgenommen. Reine short- und char-Berechnungen werden z.B. nach int promoviert. (1 P.)

b) Escapesequenz Eine Escapesequenz wird durch Backslash eingeleitet und ermöglicht ein Zeichen (normal oder spezial) mit Hilfe von Ziffern darzustellen. Bsp.: '\n' "hell\02" '\xab' '\056'. (1 P.)

c) Header-File h-Files enthalten die (öffentliche) Schnittstelle in C-Programmen: in ihnen werden Funktionsprototypen, Konstanten sowie nötige Variablen deklariert/definiert. h-Files enthalten mitunter auch etwas Code. h-/c-File-Aufteilung ist Voraussetzung, soll Code in Bibliotheken abgelegt werden. In einem solchen Fall werden h-Files als Quellcode , die c-Files in der Bibliothek (objekt-Code ohne Quelle) abgegeben. Opensource legt auch c-Files offen. (1 P.)

d) case-sensitive Der C-Compiler unterscheidet im Quelltext zwischen Gross- und Kleinschreibung. Dies gilt nicht gezwungenermassen für alle Sprachen und deren Compiler (oder Interpreter). (1 P.)

e) short-cut boolean evaluation Bei der Auswertung von bool'schen Verknüpfungen von Ausdrücken werden nachfolgende Teilausdrücke nicht mehr ausgewertet, falls ein Zwischenresultat nicht mehr beeinflusst werden kann. (1 P.)

f) passing by value Funktionsargumente werden nicht als Originale in eine Funktion hineingereicht, sondern eine Kopie davon wird übergeben. Dasselbe gilt für Returnwerte. (1 P.)

2. Kontrollstruktur (4 Pkte.)
Gegeben ist folgendes Codefragment:

```
int x = 5;        // x = 0, 1, 2...
int a = 4, b = 3;

if (x == 0) {
  a = a + b++;
}
else {
  while (x-- > 0) {
    a = a + b++;
  }
}
```

```
int x = 5;        // x = 0, 1, 2...
int a = 4, b = 3;

do {
  a = a + b++;
} while (--x > 0);
```

Weitere Lösungen möglich.

kleinerer Fehler: - 1 P.
gröberer Fehler: -2 P.
...

Vereinfachen Sie die obigen Zeilen mit Verwenden einer einzigen Kontrollstruktur. Für alle zu Beginn initialisierten x >= 0 (hier als Beispiel x = 5) sollen a und b am Schluss Werte aufweisen wie im gegebenen Codefragment.

3. `struct` (2 Pkte.)
Kann **`struct Sample`** ein Member des Typs **`struct Sample`** enthalten? Begründen Sie Ihr Ja oder Nein.

Eine Struktur kann sich selbst nicht enthalten, da dies einer rekursiven Deklaration gleichkäme. Bei Realisation der Struktur wäre unendlich viel Speicher nötig. (2 P.)

4. Portierung (10 Pkte.)
Portieren Sie das folgende Java-Programm *möglichst 1:1* in C-Code (ohne Verwendung von Pointern!).

Java	C
`public class Tar {`	`/* tar.c */`
`  public static String fooBar =`	`#include <stdio.h>`
`    "FOOBAR";`	
	`static char fooBar[] = "FOOBAR";` 1 P.
`  public int foo;`	`typedef enum _boolean {false, true}`
	`  boolean;`
`  public boolean bar;`	`typedef struct _Tar {`
	`  int foo;`
	`  boolean bar;` 1 P.
`  public Tar(int foo, boolean bar) {`	`} Tar;`
`    this.foo = foo;`	
`    setBar(bar);`	
`  }`	
	`Tar myTar;` 1 P.
`  public int getFoo() {`	`int getFoo() {` 1 P.
`    return foo;`	`  return myTar.foo;`
`  }`	`}`
`  public void setBar(boolean bar) {`	`void setBar(boolean bar) {`
`    this.bar = bar;`	`  myTar.bar = bar;` 1 P.
`  }`	`}`
	`void initTar(int foo, boolean bar) {`
	`  myTar.foo = foo;`
	`  setBar(bar);` 1 P.
	`}`
`  public static void main(`	`int main() {`
`    String[] args)`	`  initTar(88, false);` 1 P.
`  {`	
`    System.out.println(`	`  printf("fooBar: %s\n", fooBar);` 1 P.
`      "fooBar: " + fooBar);`	
`    Tar myTar = new Tar(88, false);`	
`    System.out.println(`	`  printf("myTar.foo: %d\n", getFoo());`
`      "myTar.foo: " + myTar.getFoo());`	
`    System.out.println(`	`  printf("myTar.bar: %s\n",` 1 P.
`      "myTar.bar: " + myTar.bar);`	`    myTar.bar ? "true" : "false");`
`    myTar.setBar(true);`	`  setBar(true);`
`    System.out.println(`	`  printf("myTar.foo: %d\n", getFoo());` 1 P.
`      "myTar.foo: " + myTar.getFoo());`	
`    System.out.println(`	`  printf("myTar.bar: %s\n",`
`      "myTar.bar: " + myTar.bar);`	`    myTar.bar ? "true" : "false");`
`  }`	
`}`	`  return 0;`
	`}`

5. Codevergleich (4 Pkte.)
Gegeben sind die beiden C-Code-Fragmente. Beschreiben Sie alle wesentlichen Gemeinsamkeiten und Differenzen.

```
/* frag1 */
static double arr[] = {2.0, 4.0, 8.0};

double getElem(int i) {
  return arr[i];
}

// ...
```

Unterschiede:
arr hat File Scope, kann also überall aus dem File angesprochen werden. 1 P.

```
/* frag2 */

double getElem(int i) {
  static double arr[] = {2.0, 4.0, 8.0};
  return arr[i];
}

// ...
```

Unterschiede:
arr ist nur in der Funktion **getElem** bekannt, von ausserhalb der Funktion ist kein Zugriff möglich.

Gemeinsamkeiten:
- Für **arr** wird Platz auf dem Datensegment reserviert (nicht auf dem Stack).
- Bei Programmeintritt (in **main**) sind diese Daten initialisiert. 3 P.
- Die Werte sind während des Programms persistent. Dies gilt insbesondere auch für **arr** in **frag2**.