Lösung Klausur 1c sengT

max. Punkte: 27 Note: (5/25) * P + 1

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

a) Preprocessor Bei Kompilation eines C-Programms wird in einem ersten Durchgang der Präprozessor alle #-Direktiven in den C-Dateien bearbeiten, z.B #include, #define, #ifndef etc. Erst danach wird C-kompiliert. (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) file scope Themenbereich: Sichtbarkeit von Variablen/Funktionen. File scope: Sichtbarkeit einer Variablen ab ihrer Deklaration (und Initalisierung) bis zum Datei-Ende. Ausserhalb des Files ist eine solche Variable nicht ansprechbar.

e) signature Die Signatur kennzeichnet den Namen einer Funktion, die Anzahl verlangter Parameter durch Erwähnung ihres Typs. Damit ist auch die Reihenfolge der Typen dieser Parameter festgelegt. Die Namen der Parameter sind im Prototypen nicht erforderlich. Der Rückgabewert einer Funktion gehört nicht zur Signatur. (1 P.)

f) function prototype Deklaration einer Funktion ohne Implementation derselben. Die Deklaration wird mit Rückgabewert, Name der Funktion und der Parameterliste vorgenommen. Der Compiler erhält so vor der Funktionsimplementation Kenntnis dieser Funktion. Bei Aufrufen im Code kann der Compiler Typ-Checking vornehmen.

2. Interface und Implementation (4 Pkte.)
Gegeben ist folgender Code:

```
int x[][Ψ] = {{17, 1, -4}, {0, -3, 2}};

int getXY(int xCoord, int yCoord) {
  return x[xCoord][yCoord];
}

main() {
  printf("(x, y) = %i", getXY(1, 2));
}
```

```
int x[] = {17, 1, -4, 0, -3, 2};

int getXY(int xCoord, int yCoord) {
  return x[xCoord* 3 + yCoord];
}

main() {
  printf("(x, y) = %i", getXY(1, 2));
}
```

a) Durch welchen minimalen Wert ist Ψ im obigen Code zu ersetzen?
Ψ = 3 1 P.

b) Ergänzen Sie das rechte Codefragment so, dass für alle gültigen **xCoord** und **yCoord** der Funktion **getXY** im linken und rechten Fragment dieselben Ausgaben resultieren. 3 P.

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

Java	C

`public class Tar {`
`  public static String fooBar =`
`  "FOOBAR";`
	`/* tar.c */`
	`#include <stdio.h>`
	
	`static char fooBar[] = "FOOBAR";` 1 P.

`  public int foo;`
 	
`  public boolean bar;`
	`typedef enum _boolean {false, true}`
	`  boolean;`
	
	`typedef struct _Tar {`
	`  int foo;`
	`  boolean bar;` 1 P.
	`} Tar;`

`  public Tar(int foo, boolean bar) {`
`  this.foo = foo;`
`  setBar(bar);`
`  }`
	`Tar myTar;` 1 P.

`  public int getFoo() {`
`  return foo;`
`  }`
	`int getFoo() {` 1 P.
	`  return myTar.foo;`
	`}`

`  public void setBar(boolean bar) {`
`  this.bar = bar;`
`  }`
	`void setBar(boolean bar) {`
	`  myTar.bar = bar;` 1 P.
	`}`

	`void initTar(int foo, boolean bar) {`
	`  myTar.foo = foo;`
	`  setBar(bar);` 1 P.
	`}`

`  public static void main(`
`  String[] args)`
`  {`
`  System.out.println(`
`  "fooBar: " + fooBar);`
	`int main() {`
	`  initTar(88, false);` 1 P.
	
	`  printf("fooBar: %s\n", fooBar);` 1 P.

`  Tar myTar = new Tar(88, false);`
`  System.out.println(`
`  "myTar.foo: " + myTar.getFoo());`
`  System.out.println(`
`  "myTar.bar: " + myTar.bar);`
	`  printf("myTar.foo: %d\n", getFoo());`
	
	`  printf("myTar.bar: %s\n",` 1 P.
	`  myTar.bar ? "true" : "false");`

`  myTar.setBar(true);`
`  System.out.println(`
`  "myTar.foo: " + myTar.getFoo());`
`  System.out.println(`
`  "myTar.bar: " + myTar.bar);`
`  }`
`}`
	`  setBar(true);` 1 P.
	`  printf("myTar.foo: %d\n", getFoo());`
	
	`  printf("myTar.bar: %s\n",`
	`  myTar.bar ? "true" : "false");`
	
	`  return 0;`
	`}`

4. Programm-Modularisierung (7 Pkte.)
Schreiben Sie C-Code, der aus drei Einzeldateien besteht: aus *faces.h*, *faces.c* und *testFaces.c*.
faces.c implementiere drei Funktionen, die *Gesichter* an das stdout ausgeben, z.B. **;-) :-(:-|** .
Schreiben Sie *faces.** so, dass sie für eine Library verwendet werden könnten. Mit *testFaces.c* sollen die Funktionen alle getestet werden. Schreiben Sie auch diesen Code.
Wie sieht der Kommandozeilen-Aufruf zur Kompilation/zum Linken dieser drei Dateien mit dem *gcc* aus?

```
/* faces.h */
#ifndef FACES_H
#define FACES_H

void printSmiley();
void printWheepy();
void printIndiffy();

#endif
```

2 P.

```
/* faces.c */
#include "faces.h"
#include <stdio.h>

void printSmiley() {
  printf(";-)");
}


void printWheepy() {
  printf(":-(");
}


void printIndiffy() {
  printf(":-|");
}
```

2 P.

```
/* testFaces.c */
#include "faces.h"
#include <stdio.h>


int main() {
  printSmiley();
  printf("  ");
  printWheepy();
  printf("  ");
  printIndiffy();
  printf("  ");
}
```

2 P.

`gcc -o testFaces testFaces.c faces.c` liefert das Executable **`testFaces`**. 1 P.