1. Function Pointer

```
A: 10
B: 11
C: 12
```

2. Const

- `int * a`: Zeiger auf int
- `int const * b`: Zeiger auf konstanten int
- `int * const c`: Konstanter Zeiger auf int
- `int const * const d`: Konstanten Zeiger auf konstanten int

3. Arrays

Korrekt müsste Zeile 3 heissen: `for (i=0; i<10; i++) {`.
Der Array hat 10 Elemente: Das Problem mit der Schleife der Aufgabe ist, dass bei der letzten Iteration auf Element `array[10]` zugegriffen wird, welches ein ungültiger Zugriff ist (der Array wurde nur von `array[0]` bis `array[9]` deklariert).

4. MIPS Branch Instructions

```
int i;
for (i = 11; i != 0; i--) {
  #do something
}
```

5. MIPS More Branch Instructions

```
slt $at, $s2, $s1
beq $at, $zero, Label
```

Zuerst wird, wenn, wenn $s2$ ("set on less than") kleiner ist als $s1$, *at* auf 1 gesetzt. Dann wird per *beq* ("branch on equal") geprüft, ob *at* 0 ist, was in jedem anderen Fall zutrifft (also wenn $s1 \geq s2$).

6. Endianness

Speicheradresse	**10004**	**10005**	**10006**	**10007**
Wert (hex)	6d	8c	24	00

(a) **Big Endian** (*most significant byte* an der niedrigsten Speicheradresse):
`10007` ist das niederwertigste Byte
Wert: $6d8c2400_{16} = 1837900800_{10}$

(b) **Little Endian** (*least significant byte* an der niedrigsten Speicheradresse):
`10004` ist das niederwertigste Byte
Wert: $00248c6d_{16} = 2395245_{10}$

7. Array-Zugriff

C-Programm:

```
void mul(short x[], int index, int mul) {
  x[index] *= mul;
}
```

Adresse des ersten Arrayelements: $t2
`index`: $t1
`mul`: $t0

Assembler:

```
mul: add $t3, $t1, $t1  // index*2 berechnen (da short-Array)
     add $t4, $t2, $t3  // ...und damit Adresse von x[index] berechnen
                        // $t4 enthält jetzt genaue Adresse)

     mult $t4, $t0      // multiplizieren
     mflo $t4           // Resultat holen und unter $t4 speichern

     jr $ra             // zurückspringen (temporäre $tX-Register müssen nicht
                        // restored werden)
```

8. MIPS Register/Hauptspeicher

```
lw $s2, 40($s1)
```

Angenommen, der Array ist ein Integer-Array, multiplizieren wir den Speicherbedarf der einzelnen Werte des Arrays (4 Byte) mit der Anzahl zu überspringenden Wörter (10) `= 40`.