Lösungshinweise zur Teilprüfung Software- und Internettechnologie Programmierkurs 2 Sommersemester 2004

Aufgabe 1: Verständnisfragen

a)
- `if (a=b)` semantischer Fehler, z.B. für $a = 1$ und $b = -1$, richtig: `if (a==b)`
- `return b` syntaktischer Fehler (Semikolon fehlt)
- `if (a<b) return b; else return a;` berechnet das Maximum anstatt des Minimums von `a` und `b`. Richtig: `if (a>b) return b; else return a;`

b) Es gilt

$$\begin{aligned} b &= a|b = (00011010)_2|(00000011)_2 = (00011011)_2 = (27)_{10} \\ a &= c\&\&b = (c \neq 0) \wedge (b \neq 0) = 1 \\ c &= b >> a = b >> 1 = (00001101)_2 = (13)_{10} \end{aligned}$$

c) Bildschirmausgabe für `switchCase(4)`:

```
knapp bestanden
nicht bestanden
bestanden
```

Bildschirmausgabe für `switchCase(5)`:

```
nicht bestanden
bestanden
```

korrigierte `switch`-Anweisung:

```
switch (note){
  case 4 : printf("knapp bestanden\n");
           break;
  case 5 : printf("nicht bestanden\n");
           break;
  default: printf("bestanden\n");
}
```

d) Beispiele für Adressierungsarten:

- *Register*, z.B. `mov r10,r11` kopiert Inhalt von Register 10 nach Register 11.
- *absolute Adressierung*, z.B. `mov &0x0029,r10` kopiert Inhalt des Words an der Speicheradresse `0x0029` nach Register 10.
- *indirekte Adressierung*, z.B. `mov @r10,r11` kopiert das Word an der in Register 10 abgelegten Speicheradresse nach Register 11.

Aufgabe 2: C-Programmierung

a)
```
int istPrimzahl(unsigned int x){
  unsigned int y;

  /* 0 und 1 sind keine Primzahlen */
  if (x<=1) return 0;

  /* teste fuer alle Zahlen zwischen 2 und floor(sqrt(x)), ob sie Teiler von x sind */
  for (y=2 ; y < floor(sqrt(x)) ; y++)
    if (x%y == 0){
      /* y ist Teiler von x => x ist keine Primzahl */
      return 0;
    } /* if */
  /* kein Teiler >= 2 von x gefunden => x ist Primzahl */
  return 1;
} /* istPrimzahl */
```

b)
```
void printKandidaten(unsigned int x){

  char z,y;
  char zwisch[3]; /* speichert Zwischenbuchstaben als Strings */

  if (x<100 || x>999){
    /* Eingabe ungueltig */
    return;
  } /* if */

  zwisch[2]='\0';
  /* Nimm an, dass die Zeichen 'A' bis 'Z' zusammenhaengend im Zeichensatz
     der Maschine angeordnet sind
  */
  for (z='A' ; z<='Z' ; z++){
    for (y='A' ; y<='Z' ; y++){
      zwisch[0] = z;
      zwisch[1] = y;
      if (!strcmp(zwisch,"NS") || !strcmp(zwisch,"SA") || !strcmp(zwisch,"SS")
          || !strcmp(zwisch,"HJ") || !strcmp(zwisch,"KZ"))
        continue;
      printf("MA-%s %i\n",zwisch,x);
    } /* for y */
  } /* for x */
} /* printKandidaten */
```

Aufgabe 3: Dynamische Datenstrukturen

a)
```
knoten* neuerKnoten(unsigned int neu, knoten* links, knoten* rechts){
  knoten *new;
  new = (knoten*) malloc(sizeof(knoten));
  if (new == NULL) return NULL;
  new->wert = neu;
  new->linkerSohn = links;
  new->rechterSohn = rechts;
  return new;
} /* neuer Knoten */
```

b)
```
int enthaeltBlatt(knoten* v, int b_wert){
  if (v == NULL){
    return 0;
  } /* if */
  if ((v->linkerSohn == v->rechterSohn) && (v->linkerSohn == NULL))
    /* v ist Blattknoten */
    return (v->wert == b_wert);
  return enthaeltBlatt(v->linkerSohn, b_wert) ||
    enthaeltBlatt(v->rechterSohn, b_wert);
} /* enthaeltBlatt */
```

Aufgabe 4: Sensorknotensteuerung

```
.blink: bis.b   #7,&0x0029              ; Alle LEDs loeschen
.Loop:  bic.b   #4,&0x0029              ; gelb einschalten
        mov     #0xFFFF,r14             ; warten
        mov     #0,r15
        call    #wait
        bis.b   #4,&0x0029              ; gelb ausschalten
        mov     #0xFFFF,r14             ; warten
        mov     #0,r15
        call    #wait
        jmp     .Loop
```

Aufgabe 5: MSP430-Assembler

```
.marks: push    r10             ; Registerinhalte sichern
        push    r11
        push    r12
        push    r13
        push    r14
        push    r15

        mov     r12,r13         ; Basisadresse des Notenspiegels nach r13
        ;; Notenspiegel initialisieren
        mov     #1,r14
iloop:  mov     r14,0(r13)      ; aktuellen Notenindex nach 0(r13)
        mov     #0,2(r13)       ; Anzahl initialisieren
        add     #4,r13          ; zum naechsten Listeneintrag springen
        add     #1,r14
        mov     #5,r15          ; i <= 5 bzw. 5-i >= 0 ?
        sub     r14,r15
        jge     iloop

loop:   sub     #1,r11          ; Laufvariable dekrementieren
        jge     cont            ; i >=0 ?
        pop     r15             ; Registerinhalte wiederherstellen
        pop     r14
        pop     r13
        pop     r12
        pop     r11
        pop     r10
        ret
cont:
        mov     2(r10),r13      ; berechne durch Note gegebenen offset
        sub     #1,r13
        add     r13,r13         ; r13 = 4* r13
        add     r13,r13

        add     r12,r13         ; Basisadresse zum offset addieren
        add     #1,2(r13)       ; anzahl inkrementieren
        add     #4,r10          ; zum naechsten Ergebnislisteneintrag springen
        jmp     loop
```