Lösungshinweise zur Teilprüfung Software- und Internettechnologie Programmierkurs 2 Herbstsemester 2006/2007

Aufgabe 1:

a) Die Ausgabe lautet 1. Mögliche korrigierte Funktion:

```
int occurs(int a[], int n){
  int i=n-1;
  while (i >= 0){
   if (a[i--]==42) return 1;
  }
  return 0;
}
```

b) mögliche iterative Version:

```
int fibonacci(int n){
  int old1=0, old2=1, new=1, i=2;
  if (n==0) return 0;
  while (i <= n){
    new = old1+old2;
    old1=old2;
    old2=new;
    i++;
  }
  return new;
}
```

c) Die rekursive Version hat eine Laufzeit von $\Theta(2^n)$, die iterative Version benötigt $\Theta(n)$ Zeit.

d) Ersetze `int a[]` durch `int *a` und `a[i]` durch `*(a+i)`.

Aufgabe 2:

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv){
  int start_h, start_min, end_h, end_min;
  int start_in_mins, end_in_mins;
  int diff, diff_h, diff_min;

  if (argc < 5){
    printf("Benutzung: diff start_h start_min end_h end_min\n");
    return 1;
  } /* if */

  start_h = (int)atoi(argv[1]);
  start_min = (int)atoi(argv[2]);
  end_h = (int)atoi(argv[3]);
  end_min = (int)atoi(argv[4]);

  if (start_h < 0 || start_h > 23 || end_h < 0 || end_h > 23 ||
      start_min < 0 || start_min > 59 || end_min < 0 || end_min > 59){
    printf("Ungueltige Eingabe.\n");
    return 1;
  } /* if */
```

```
  start_in_mins = start_h * 60 + start_min;
  end_in_mins = end_h * 60 + end_min;

  diff = end_in_mins - start_in_mins;
  diff_h = diff / 60;
  diff_min = diff % 60;

  printf("Die Veranstaltung dauert %i Stunde(n) und %i Minute(n).\n",
         diff_h, diff_min);

  return 0;
} /* main */
```

Aufgabe 3:

a)
```
int** create(int l, int n){
  int i;
  int **C = (int**)malloc(l * sizeof(int*));

  if (C==NULL) return NULL;
  for (i=0 ; i < l ; i++){
    C[i] = (int*) calloc(n,sizeof(int*));
    if (C[i]==NULL){
      int j;
      for (j=0 ; j < i ; j++){
        free(C[j]);
      } /* for j */
      free(C);
      return NULL;
    } /* if */
  } /* for */
  return C;
}
```

b)
```
int isNull(int** A, int l, int n){
  int i,j;
  if (A==NULL) return 0;
  for (i=0 ; i < l ; i++){
    for (j=0 ; j < n ; j++){
      if (A[i][j]!=0) return 0;
    } /* for j */
  } /* for i */
  return 1;
}
```

c)
```
int** mult(int** A, int** B, int l, int m, int n){
  int i,j,k;
  int **C;

  if (A==NULL || B==NULL) return NULL;

  C = create(l,n);
  if (C==NULL) return NULL;

  if (isNull(A,l,m) || isNull(B,m,n)) return C;

  for (i=0 ; i < l ; i++){
    for (j=0 ; j < n ; j++){
      for (k=0 ; k < m ; k++){
        C[i][j] += A[i][k] * B[k][j];
      } /* for k */
    } /* for j */
  } /* for i */
  return C;
}
```

d)
```
void destroy(int** C, int l, int n){
  int i;
  if (C==NULL) return;
  for (i=0 ; i < l ; i++) free(C[i]);
  free(C);
}
```

Aufgabe 4:

a) Nein, `mov @r10+,r11` inkrementiert `r10` um 2.

b) `r10` hat den Wert `0xf`, `r11` den Wert `0x8`

c) Nein, weil ein Befehl zusammen mit seinen Argumenten auch länger als 2 Bytes sein kann.

Aufgabe 5:

```
binarySearch:
        ;; Parameter:
        ;; r8 Startadresse Feld A
        ;; r9 Laenge des Felds
        ;; r10 gesuchtes Element x
        ;; r11 Ergebnis

        ;; lokale Variablen
        ;; r9 r
        ;; r12 l
        ;; r13 m
        ;; r14 &(A[m]);

        push r9                 ; Registerinhalte sichern
        push r12
        push r13
        push r14

        mov #0,r12              ; l=0
        sub #1,r9               ; r=n-1
        mov #0,r11
loop:   cmp r12,r9              ; while (l<=r)
        jl end

        mov r9,r13              ; m = (l+r)/2
        add r12,r13
        rra r13

        mov r13,r14             ; r14 = &(A[m])
        add r13,r14
        add r8,r14

        cmp @r14,r10            ; if (A[m]==x)
        jeq found
        jl  less
        mov r13,r12             ; l=m+1
        add #1,r12
        jmp loop
less:   mov r13,r9              ; r=m-1
        sub #1,r9
        jmp loop

found:  mov #1,r11
end:
        pop r14                 ; Registerinhalte wiederherstellen
        pop r13
        pop r12
        pop r9
        ret
```