Aufgabe 1: Gegeben sei der Typ eines Listenelements einer einfach verketteten Liste:
struct el { int value; *// Wert des Listenelements*
struct el *next; *// Zeiger auf nächstes Listenelement* ***}***

Weiterhin sind formuliert:
struct el *start = 0, *p = (struct el *) malloc(sizeof(struct el)); p → value =5; (∗***p).next=0;***

Die folgenden Funktionen sind zu implementieren (optional können auch C-Funktionen formuliert werden, die im Rahmen einer main()-Funktion getestet werden können), vgl. Programmablaufpläne in ***Algorithmen_ueber_Listen.pdf*** :

a.) Es ist eine Funktion ***void showpos*** als **Programmablaufplan** zu implementieren, die für das Listenelemente mit Position i (***i*** ≥***0***) den Wert von ***value*** auf der Konsole ausgibt. Der Funktion sind die Parameter ***start*** und ***i*** mit der Angabe als ***in-***, ***out-*** oder ***inout-***Parameter zu übergeben. Der Test ***i*** ≥***0*** soll in der Funktion erfolgen. Sollte ***i*** ≥ Anzahl der Listenelemente, dann wird eine Fehlermeldung ausgegeben.

b.) Es ist eine Funktion ***void removepos*** als **Programmablaufplan** zu implementieren, die das Listenelement an der Position ***i*** (***i*** ≥***0***) aus der Liste löscht. Sollte die Liste leer sein, dann wird nichts gelöscht. Der Funktion sind ***start*** und ***i*** mit der Angabe als ***in-***, ***out-*** oder ***inout-***Parameter zu übergeben. Der Test ***i*** ≥***0*** soll in der Funktion erfolgen. Sollte ***i*** ≥ Anzahl der Listenelemente, dann wird nichts gelöscht.

c.) Es ist eine Funktion ***struct el *searchvalue*** als **Programmablaufplan** zu implementieren, die die Adresse des ersten Listenelementes mit Wert ***value*** zurückgibt. Sollte kein derartiges Element gefunden worden sein, dann wird ***0*** zurückgegeben. Der Funktion sind ***start*** und ***value*** mit der Angabe als ***in-***, ***out-*** oder ***inout-***Parameter zu übergeben.

Aufgabe 2: Gegeben sei die Definition eines Stacks ***stack*** für ***int***-Werte und ein Stack-Index ***sp***:
int *stack = 0, sp = 0; // ***sp*** ist immer der Index des ersten freien Elements von ***stack[]***
Diese beiden Variablen ***stack[]*** und ***sp*** sollen bzgl. der folgenden Funktionen **global** sein.

Folgende Funktionen sind als **Programmablaufpläne** zu formulieren:
a.) Funktion ***void stackinit*** mit Parameter ***n>0*** zum Anlegen von ***stack[]*** mit ***n*** Elementen, dabei soll ***n*** begründet als ***in-***, ***out-*** oder ***inout-***Parameter übergeben werden.
b.) Funktion ***int isEmpty*** ohne Parameter, Rückgabe ≠***0***, falls ***stack[]*** leer, sonst ***0***
c.) Funktion ***int isFull*** ohne Parameter, Rückgabe ≠***0***, falls ***stack[]*** voll, sonst ***0***
d.) Funktion ***void push*** mit Parameter ***wert***, die ***wert*** in ***stack[]*** speichert, Parameter als ***in-***, ***out-*** oder ***inout-***Parameter übergeben.
e.) Funktion ***void pop*** ohne Parameter zum Entfernen des obersten Elementes aus ***stack[]***
f.) Funktion ***int top*** ohne Parameter zur Rückgabe des obersten Elementes aus ***stack[]***, ohne das Element zu löschen
g.) Funktion ***popTop*** ohne Parameter zur Rückgabe des obersten Elementes aus ***stack[]*** und zum Entfernen des obersten Elementes

Aufgabe 3: Gegeben sei **queue/queue/queue.c** (Vorlesung 5). In ***main()*** soll für ***queue q*** die Funktion ***init_queue(5, &q)*** gerufen worden sein. Danach soll ***enqueue(&q,1); enqueue(&q,2); enqueue(&q,3); enqueue(&q,4); enqueue(&q,5); dequeue(&q); dequeue(&q); enqueue(&q,6); enqueue(&q,5); dequeue(&q);*** gerufen werden.
Man skizziere den Inhalt von ***q*** und gebe ***q→ first*** und ***(*q).last*** und ***q→ count*** an.

Aufgabe 4: Folgende Zahlenfolge wird beim Aufbau eines Binärbaumes eingegeben:
40, 60, 20, 30, 50, 70, 54, 52, 57, 80, 51, 53, 44, 66, 68, 64, 58, 56, 59, 55, 62, 65, 67, 69, 16, 12, 8, 4, 2, 1, 0, 18, 19, 17, 38, 24, 36, 39, 32, 37, 31, 33, 34, 6, 7, 5, 3
Man baue den Binärbaum auf und gebe die Zahlenfolgen an, die beim ***preorder-***, ***inorder-*** und ***postorder-*** Durchlauf ausgegeben werden.

Aufgabe 5: Man erweitere die Funktion ***void insert(int zahl, struct node *k)*** aus ***binbaum.c*** derart, dass die wiederholte Eingabe eines bereits gespeicherten Wertes ***zahl*** abgewiesen wird.

Aufgabe 6: Gegeben sei ein **Binärbaum** mit Wurzel ***u***. Schreiben Sie einen Algorithmus ***delete(u, w)*** in Pseudocode, der für einen Binärbaum den Knoten ***w*** löscht. Erweitern Sie das C-Programm **binbaum.c** aus um eine Funktion int delete(struct node *u, struct node *k), der Rückgabewert 0 beschreibt die Unauffindbarkeit von Knoten k im Baum, der Wert 1 beschreibt das erfolgreiche Löschen des Knotens ***w***.

Aufgabe 7: Man schreibe eine Funktion in C, die **von allen** mehrfach in einem Baum befindlichen Werten genau nur einen im Baum beläßt.

Aufgabe 8: Die Zahlenfolge ***30, 28, 26, 24, 22, 20, 10, 12, 14, 16, 18, 8, 6, 4, 2, 0*** ist mit ***Bubblesort***, ***InsertionSort***, ***SelectionSort*** manuell aufsteigend zu sortieren. Für jeden Schleifendurchlauf ist dabei die neu geordnete Zahlenfolge anzugeben.

Aufgabe 9: Die Zahlenfolge ***9, 8, 7, 6, 5, 4, 3, 2, 1, 0*** ist mit **Mergesort** und mit unten angegebenem **quick_sort** sowohl mit **partition** mit **x = z[r]** als auch mit **x = z[l]** manuell zu berechnen, wobei jeweils die aktuellen Parametern **l** und **r**, die Rückgabe für **pivot**, die Folge der Werte für **i**, **j** und **x** und das Tauschen von Elementen der Zahlenfolge **z** mit Indizes zu notieren ist.

```
void swap(int *a, int *b) {
   int h = *a; *a = *b; *b = h;
}

int partition(int z[], int l, int r) { // Vergleich mit x=z[r]
    int x = z[r], i = l-1, j = r;
    while (1) {
       while (z[++i] < x);
       while ((l<j) && (z[--j] > x));
       if (i < j) swap(z+i, z+j);
       else { swap(z+i, z+r); return i;}
    }
}

int partition(int z[], int l, int r) { // Vergleich mit x=z[l]
    int x = z[l], i = l, j = r+1;
    while (1) {
       while ((i<r) && (z[++i] < x));
       while (z[--j] > x);
       if (i < j) swap(z+i, z+j);
       else { swap(z+j, z+l); return j;}
    }
}

void quick_sort(int z[], int l, int r) {
   if (l < r) {
      int pivot = partition(z, l, r);
      quick_sort(z, l, pivot-1);
      quick_sort(z, pivot+1, r);
   }
}
```