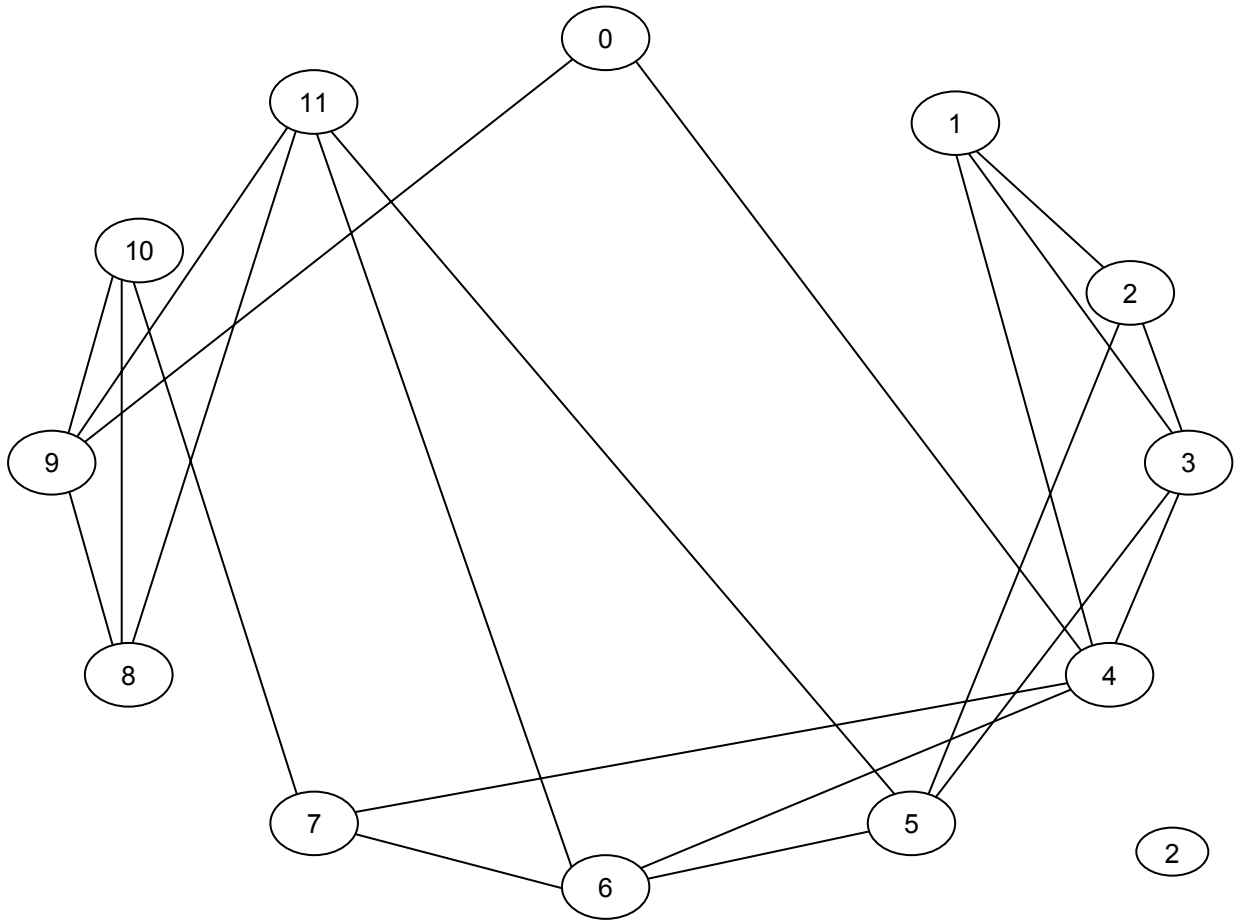
Lösung Klausur 2 infT

Entwicklung/Erweiterung einer Applikation in C
Die zu erstellende C-Applikation umfasst drei Dateien, nämlich *netNode.h*, *netNode.c* und *netNodeTest.c*. Diese Dateien erhalten Sie vorbereitet. Die **`#include`**s verändern/löschen Sie, so wie dies Ihre Projektumgebung verlangt. Am Schluss geben Sie **nur** die Datei *netNode.c* ab - achten Sie also darauf, dass diese stets mit der abgegebenen Header- und Test-Datei kompilierbar ist.

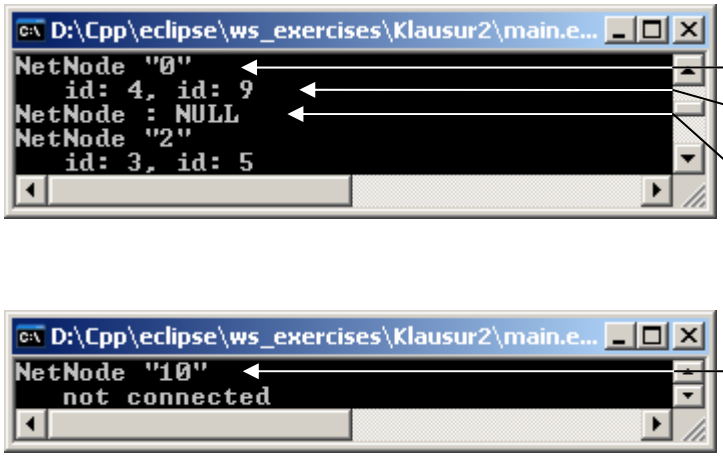
Aufgabenstellung:
Die 3 Dateien sollen ein Netz aus NetNodes realisieren und testen. Sie haben die Funktionalität in der Datei *netNode.c* zu realisieren. Beachten Sie, dass die NetNodes dynamisch anzulegen sind.

1. (2 Pkte.)
Verschaffen Sie sich einen Code-Überblick in der Datei *netNodeTest.c*. Dort werden 12 NetNodes mit Zeigern zu einem Netz verknüpft (ähnlich wie mit Zeigern verkettete Listen erstellt werden können). Verbinden Sie die durch nummerierte Ovale bezeichneten NetNodes mit geraden (!) Linien gemäss dem Testprogramm-Abschnitt "**`// Verknuepfe die NetNodes zu einem Netz:`**".



2. (2 Pkte.)
Implementieren Sie die Funktion **`NetNode *createNetNode()`** in der Datei *netNode.c*. Lesen Sie die Anforderungen an die Funktion, wie sie im Kommentar in der Datei *netNode.c* formuliert sind.
Hinweis:
Die vorbereitete Datei *netNodeTest.c* verändern Sie nach Ihren Bedürfnissen, da Sie diese ja nicht abgeben müssen.

3. (2 Pkte.)
Realisieren Sie die Funktion **`void printNetNode(NetNode *pnn)`**. Die Ausgaben sollten wie unten gezeigt aussehen.
Hinweis:
Diese Funktion vervollständigen Sie allenfalls erst nach Lösen nachfolgender Aufgaben.



4. (3 Pkte.)
Implementieren Sie die private Funktion **`static boolean addNetNode(NetNode *psnn, NetNode *ptnn)`**. Diese Funktion stellt eine Verbindung vom Quell-NetNode zu Ziel-NetNode her (einfache Verkettung). Sie wird in Aufgabe **6.** benötigt.

5. (5 Pkte.)
Implementieren Sie die private Funktion **`static NetNode *removeNetNode(NetNode *psnn, NetNode *ptnn)`**. Diese Funktion entfernt eine Verbindung vom Quell-NetNode zu Ziel-NetNode (einfache Verkettung). Sie wird ab Aufgabe **6.** benötigt.

6. (3 Pkte.)
Realisieren Sie die Funktion **`boolean connectNetNode(NetNode *psnn, NetNode *ptnn)`**. Diese soll sich der beiden in Aufgaben **4.** und **5.** bereitgestellten privaten Funktionen bedienen.

7. (2 Pkte.)
Realisieren Sie die Funktion **`void disconnectNetNode(NetNode *pnn)`**. Diese soll sich der in Aufgaben **5.** bereitgestellten privaten Funktion bedienen.

8. (2 Pkte.)
Implementieren Sie die Funktion **`boolean connectWithSeveralNetNodes(NetNode *pnn, NetNode **nnArr, int nnArrLen)`**.

9. (2 Pkte.)
Implementieren Sie die Funktion **`boolean deleteNetNode(NetNode *pnn)`**.

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


/* Kreiere einen NetNode auf dem Heap.
 *
 * Params:
 *   --               --
 * Returns:
 *   NetNode *        Zeiger auf NetNode auf dem Heap oder
 *                    NULL, falls Kreieren nicht erfolgreich
 */
NetNode *createNetNode() {
   static int i = 0;

   NetNode *pnn = (NetNode *)malloc(sizeof(NetNode));
   if (!pnn) return NULL;                                          1

   pnn->id = i++;
   pnn->pNnArr = NULL;
   pnn->nnArrLen = 0;                                              1
   return pnn;
}


/* Stelle einen NetNode am stdout wie folgt dar (am Beispiel von NetNode
 * mit id = 0 gezeigt, der mit NetNodes 4 und 9 verbunden ist):
 * NetNode "0"
 *    id: 4, id: 9
 * Falls mit keinem NetNode verbunden:
 * NetNode "1"
 *    not connected
 * Falls Parameter selbst NULL:
 * NetNode : NULL
 *
 * Params:
 *   NetNode *pnn    Zeiger auf darzustellenden NetNode, darf auch NULL sein
 * Returns:
 *   --              --
 */
void printNetNode(NetNode *pnn) {
   int i;
   if (!pnn) {
       printf("NetNode : NULL\n");
       return;
   }

   printf("NetNode \"%d\"\n", pnn->id);
   printf("   ");
   for (i = 0; i < pnn->nnArrLen; ++i) {                           1
       if (i != 0) printf(", ");
       printf("id: %d", pnn->pNnArr[i]->id);
   }

   if (!pnn->nnArrLen) printf("not connected");                     1
   putchar('\n');
}
```

```
/* PRIVATE
 * Fuege dem Array des Quell-NetNodes (auf den psnn zeigt) den Zeiger ptnn
 * (der auf den Ziel-NetNode zeigt) hinzu.
 *
 * Params:
 *   NetNode *psnn   Zeiger auf Quell-NetNode (source)
 *   NetNode *ptnn   Zeiger auf Ziel-NetNode (target)
 * Returns:
 *   boolean         TRUE: Hinzufuegen ok,
 *                   FALSE: kein Speicher verfuegbar
 */
static boolean addNetNode(NetNode *psnn, NetNode *ptnn) {
   // Alloziere neuen Speicherplatz:
   int nnArrLen = psnn->nnArrLen + 1;
   NetNode **pNnArr = (NetNode **)realloc(psnn->pNnArr, sizeof(NetNode *) * nnArrLen);
   if (!pNnArr) return FALSE;

   // Speichere Zeiger in Array:
   pNnArr[psnn->nnArrLen] = ptnn;

   // Aktualisiere NetNode:
   psnn->pNnArr = pNnArr;
   psnn->nnArrLen++;
  return TRUE;
}


/* PRIVATE
 * Entferne aus dem Array des Quell-NetNodes den Ziel-NetNode-Zeiger.
 *
 * Params:
 *   NetNode *psnn   Zeiger auf Quell-NetNode (source)
 *   NetNode *ptnn   Zeiger auf Ziel-NetNode (target)
 * Returns:
 *   NetNode *       Zeiger auf Ziel-NetNode: Entfernen ok,
 *                   NULL: Ziel-NetNode nicht gefunden
 */
static NetNode *removeNetNode(NetNode *psnn, NetNode *ptnn) {
   int i;
   boolean found = FALSE;
   if (!psnn->nnArrLen) return NULL;

   // Suche Zeiger im Array und verschiebe diesen ggf.:
   for (i = 0; i < psnn->nnArrLen; ++i) {
       if (!found) {
              if (psnn->pNnArr[i] == ptnn) found = TRUE;
       }
       else {
              psnn->pNnArr[i - 1] = psnn->pNnArr[i];
       }
   }

   // Zeiger nicht vorhanden?
   if (!found) return NULL;

   psnn->nnArrLen--;

   // Keine Zeiger mehr im Array?
   if (!psnn->nnArrLen) {
       free(psnn->pNnArr);
       psnn->pNnArr = NULL;
       return ptnn;
   }

   // Array verkuerzen:
   psnn->pNnArr = (NetNode **)realloc(psnn->pNnArr, sizeof(NetNode *) * psnn->nnArrLen);
   return ptnn;
}
```

```
/* Verbinde einen Quell-NetNode mit einem Ziel-NetNode und umgekehrt
 * (doppelt verkettet).
 * Hat das Verbinden geklappt, antworte TRUE, sonst FALSE.
 *
 * Params:
 *   NetNode *psnn   Zeiger auf Quell-NetNode (source)
 *   NetNode *ptnn   Zeiger auf Ziel-NetNode (target)
 * Returns:
 *   boolean         TRUE: Hinzufuegen ok,
 *                   FALSE: kein Speicher verfuegbar
 */
boolean connectNetNode(NetNode *psnn, NetNode *ptnn) {
   if (addNetNode(psnn, ptnn)) {
       if (addNetNode(ptnn, psnn)) return TRUE;

       removeNetNode(psnn, ptnn);
   }
   return FALSE;
}


/* Loese NetNode aus allen Verbindungen. Stelle sicher, dass alle
 * andern NetNodes diesen NetNode aus ihrer Verbindung entlassen haben.
 *
 * Params:
 *   NetNode *pnn    Zeiger auf NetNode, der aus Netz entfernt werden soll
 * Returns:
 *   --              --
 */
void disconnectNetNode(NetNode *pnn) {
  while (pnn->nnArrLen) {
       NetNode *ptnn = removeNetNode(pnn, pnn->pNnArr[0]);
       removeNetNode(ptnn, pnn);
   }
}


/* Verbinde einen NetNode mit einer Anzahl weiterer NetNodes. Konnten alle
 * Verbindungen hergestellt werden, so antworte TRUE, sonst FALSE.
 *
 * Params:
 *   NetNode *pnn    Zeiger auf NetNode, der mit andern NetNodes verbunden
 *                   werden soll
 *   NetNode **nnArr Zeiger aud Array mit NetNode-Zeigern
 *   int nnArrLen    Laenge des Arrays
 * Returns:
 *   boolean         TRUE, falls alle Verbindungen zustande gekommen sind,
 *                   sonst FALSE
 */
boolean connectWithSeveralNetNodes(NetNode *pnn, NetNode **nnArr, int nnArrLen) {
   int i;

  for (i = 0; i < nnArrLen; ++i) {
   if (!connectNetNode(pnn, nnArr[i])) break;
  }

  if (i == nnArrLen) return TRUE;

  for (; i >= 0; --i) {
   disconnectNetNode(pnn);
  }
  return FALSE;
}
```

```
/* Entferne den NetNode vollstaendig vom Heap. Der NetNode kann nur vom Heap
 * entfernt werden, wenn er vorgaengig aus dem Netz entfernt wurde (mit
 * disconnectNetNode), bzw. keine Verbindungen mehr aufweist.
 *
 * Params:
 *   NetNode *pnn    Zeiger auf NetNode, der vom Heap entfernt werden soll
 * Returns:
 *   boolean         TRUE, falls NetNode entfernt werden konnte,
 *                   sonst FALSE
 */
boolean deleteNetNode(NetNode *pnn) {
   if (!pnn) return FALSE;
   if (pnn->nnArrLen) return FALSE;

   free(pnn);
   return TRUE;
}
```

1

1