```
/* The header file (personal) for this project (circular_list.h)
 * Declarations, type definitions, functions' prototypes etc.
 */

#ifndef __CIRCULAR_LIST__
#define __CIRCULAR_LIST__

typedef struct Persoana
{
   unsigned int ziNastere;
   unsigned int anNastere;
   char Nume[16];
   char Adresa[24];
   struct Persoana *pNext; /* vlad: only one pointer for now: to the next node */
} NODE, *pNODE, **ppNODE;

typedef enum boolean{False, True} BOOL;

void  circ_addNode(ppNODE, pNODE);   /* vlad: the convention we make for
                                      * the 2nd argument here is to preserve the link
                                      * (pointer) to the node before the one to be
                                      * inserted.
                                      */
void  circ_delNode(ppNODE, pNODE);   /* vlad: the convention we make here
                                      * for the 2nd argument tells us that we preserve
                                      * the pointer to some node,
                                      * the one before the deleted one.
                                      */
void  circ_listTraversal(pNODE, unsigned int*);  /* vlad: start with the 1st node
                                      * and move onward.
                                      * Not necessary to take double pointer (!)
                                      */
BOOL  isEmpty(pNODE);    /* vlad: this checks if the list is empty */

#endif
```

```
/* Functions' implementations. (circular_list.c)
 * This is another module within this project.
 * This is where all the functions' implementation should belong.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

#include "circular_list.h"    /* vlad: the new header file */


void circ_addNode(ppNODE ppPrim, pNODE p) /* vlad: we insert anywhere in the
                                * list, but AFTER 'p' (!)
                                */
{
   /* vlad: the new node is dynamically allocated. It doesn't exist otherwise */
   pNODE q = (pNODE) malloc(sizeof(NODE));
   assert(q != NULL);
   fprintf(stdout, "New node's address: %p\n", q);

   /* The information of the new node. It should be asked again each time */
   q->anNastere = 2002;  /* vlad: how can we automate the process here
                          * (and below)?
                          */
   q->ziNastere = 2;     /* vlad: I mean we have several calls to this function.
                          * So some inter-calls data must be preserved...
                          */
   strcpy(q->Adresa, "Adr1");
   strcpy(q->Nume, "N1");

   /* The link related to the new node */
   if( isEmpty(*ppPrim)) {
      q->pNext = q;      /* no NULL links in a circular list (!) */
      *ppPrim = q;       /* mark the new first node in the list.
                          * Such a node is required in traversal
                          */
   } else {
      q->pNext = p->pNext;     /* Preserve the link, in the first place… */
      p->pNext = q;            /* … after which you can alter it.
                                * The new node (here, q) clearly is the successor
                                * of p.
                                */
      }
   return;
}


void circ_delNode(ppNODE ppPrim, pNODE p)
{
   pNODE q;

   q = p->pNext;
   p->pNext = q->pNext;        /* vlad: no further condition required,
                                since no NULL link exists */

   if( q == *ppPrim )          /* vlad: q is the node about to be deleted. What
                                if it is the 1st node? */
   {
      *ppPrim = (*ppPrim)->pNext;/* q will be deleted. The 1st node must change*/
      if( q->pNext == *ppPrim )     /* the list contains only one node. It will
                                     become empty(!) */
```

```
            *ppPrim = NULL;         /* vlad: the list is empty now */
   }

   free(q);       /* delete q */
   return;
}

void circ_listTraversal(pNODE pPrim, unsigned int *pLen)
{
   if(!isEmpty(pPrim))
   {
      *pLen = 0; /* just in case. What if the outside variable is not set to 0?*/
      pNODE tmp = pPrim;      /* vlad: start with the 1st node */
      do {
            puts(tmp->Nume); puts(tmp->Adresa);  /* data processing */
            fprintf(stdout, "Node: %u: year: %u, zi: %u\n",
                                          *pLen, tmp->anNastere, tmp->ziNastere);
            tmp = tmp->pNext;
            (*pLen)++;        /* Pay attention to the operators' precedence.
                               * Use parentheses!
                               * Otherwise side effects could happen.
                               * Update the variable, at each step
                               */
      } while(tmp != pPrim);  /* end with... the 1st node (this is roundness,
                               * after all)
                               * + No NULL links here (!)
                               */
   } else {
      *pLen = 0;  /* vlad: just in case.
                     It is normal to be that way (if the list is empty) */
      fprintf(stdout, "The list is empty. Cannot traverse it!\n");
   }
   return;
}

BOOL isEmpty(pNODE pPrim)
{
    return ( pPrim == NULL ) ? True : False;
}
```

```
/* The main module (main.c).
* It will (directly or not) interact with all other modules belonging to this
project.
*/
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

#include "circular_list.h"    /* vlad: my header stays within the same folder
                               * as the source-code files...
                               */

int main()
{
   unsigned int len;
   pNODE prim = NULL;   /* vlad: empty list, at the very beginning */

   /* vlad: check and prove that the list is empty, at the very beginning */
   fprintf(stdout, "Is the list empty? %s",
                                    (isEmpty(prim) == True) ? "Yes\n" : "No\n");

   /* Add nodes to the list */
   puts("");
   fprintf(stdout, "Adding nodes...\n");
   circ_addNode(&prim, NULL);
   assert(prim != NULL);      /* vlad: double check that we inserted the 1st
                               * node, in the first place.
                               * The list here should not be empty anymore (!)
                               */
   circ_addNode(&prim, prim);
   circ_addNode(&prim, prim);
   circ_addNode(&prim, prim);

   /* List traversal */
   puts("");
   fprintf(stdout, "Nodes' values...\n");
   circ_listTraversal(prim, &len); /* vlad: I only need the 1st node's address*/
   fprintf(stdout, "List's length: %u\n", len);

   /* Delete the list (node by node). At the end, it should be empty */
   puts(""); /* outputs a newline */
   fprintf(stdout, "Is the list empty? %s",
                                    (isEmpty(prim) == True) ? "Yes\n" : "No\n");
   fprintf(stdout, "Deleting the nodes...\n");
   for( ; !isEmpty(prim); --len)    /* vlad: the length here assumes
                                     a previous non-zero computed value */
    {
      fprintf(stdout, "Actual first node's address: %p\n", prim);
      circ_delNode(&prim, prim);
    }

   /* A final traversal should report a 0-length value */
   fprintf(stdout, "Length after deletion: %u\n", len);
   fprintf(stdout, "First node's address: %p\n", prim);      /* should be NULL */
   circ_listTraversal(prim, &len); /* vlad: I only need the 1st node reference */
   fprintf(stdout, "List's length: %u\n", len);
   fprintf(stdout, "Is the list empty? %s",
                                    (isEmpty(prim) == True) ? "Yes\n" : "No\n");

   return 0;
}
```