

```

/* The header file (personal) for this project (list_example.h)
* Declarations, type definitions, functions' prototypes etc.
*/
#ifndef __LIST_EXAMPLE__
#define __LIST_EXAMPLE__

typedef struct Persoana
{
    unsigned int ziNastere;
    unsigned int anNastere;
    char Nume[16];
    char Adresa[24];
    struct Persoana *pNext; /* vlad: only one link here: to the next node */
} NODE, *pNODE, **ppNODE; /* The new user defined data types, useful here */

typedef enum boolean{False, True} BOOL; /* Note: BOOL != unsigned int */

void addNode(ppNODE, pNODE); /* vlad: the convention made for the 2nd
                           * argument here is to preserve the link (pointer)
                           * to the node preceding (before) the one to be inserted.
                           */
Void delNode(ppNODE, pNODE); /* vlad: the convention made here for the 2nd
                           * argument tells us that we preserve the pointer to
                           * some node, the one before the deleted one.
                           */
void listTraversal(pNODE, unsigned int*); /* vlad: start with the 1st node
                                         * and move onward.
                                         */
BOOL isEmpty(pNODE); /* vlad: this checks if the list is empty */

#endif

```

```

/* Functions' implementations. (list_example.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 "list_example.h"      /* vlad: the new designed header file is
                                * required here.
                                * Otherwise the functions below are not
                                * known at all.
                                */
                                */

void addNode(ppNODE ppPrim, pNODE p) /* vlad: we insert anywhere in the list,
                                         * but AFTER 'p' (!)
                                         */
{
    /* vlad: from arguments' perspective, a function call is by default
     * performed 'by value'. This also stands for addresses!
     * Here the address of the first node in the list is passed to the function.
     * The changes within this function may relate to this address. Therefore,
     * they are required outside of this fct as well...
     * That's why I should have a double pointer for prim (ppPrim = &prim)
     * so that I can see when it changes from outside the function!
     */
    pNODE q = (pNODE) malloc(sizeof(NODE));
    assert(q != NULL);
    fprintf(stdout, "New node's address: %p\n", q);

    /* The information (data) of the new node. It should be asked each time */
    q->anNastere = 2001; /* vlad: how can we automate the process here
                           * (and below)? Some inter-calls data must
                           * be preserved...
                           */
    q->ziNastere = 1;
    strcpy(q->Adresa,      "Adr1");
    strcpy(q->Nume,         "N1");

    /* The links of the new node */
    if(p == NULL) /* vlad: insert at the beginning */
    {
        q->pNext = *ppPrim; /* At the very first call of addNode(), the
                               * list has only one node!
                               * It doesn't point to anything after it!
                               */
        *ppPrim = q; /* prim is now NOT NULL */
        fprintf(stdout, "prim's address: %p\n", *ppPrim);
    } else { /* vlad: insert anywhere in the list, after p (!) */
        q->pNext = p->pNext; /* Preserve the link, in the first place...
                               * ... after which you can alter it.
                               * The new node (here, q) clearly comes
                               * after p.
                               */
    }
}

return;
}

```

```

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

    if( p == NULL )      /* vlad: we're about to delete the 1st node */
    {
        q = *ppPrim;      /* q retains the address of the node to be deleted.
                               * Here, q holds the 1st node address.
                               */
        *ppPrim = (*ppPrim)->pNext;   /* a new first node comes into place:
                                         * the former 2nd node now becomes the
                                         * first one.
                                         */
    } else {           /* vlad: delete some node, positioned after p */
        q = p->pNext;
        if(q) p->pNext = q->pNext; /* that's how we by-pass the node after p */
    }

    free(q);          /* effectively delete q.
                         * Note that it stores some address previously obtained
                         * through a call to malloc().
                         */
}

return;
}

void listTraversal(pNODE pPrim, unsigned int *pLen)
{
    *pLen = 0;

    pNODE tmp = pPrim;    /* vlad: start with the 1st node */
    while(tmp != NULL)
    {
        puts(tmp->Nume); puts(tmp->Adresa);
        fprintf(stdout, "Node: %u: year: %u, zi: %u\n",
                *pLen, tmp->anNastere, tmp->ziNastere);
        tmp = tmp->pNext;
        (*pLen)++;          /* Pay attention to the operators' precedence.
                               * Otherwise, you won't compute the list's length...
                               */
    } /* end-while */

    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"list_example.h"      /* vlad: this new header file stays within the
                               * same folder as the source-code files...
                               */

int main()
{
    unsigned int len;
    pNODE prim = NULL;      /* vlad: empty list, in the first place */

    /* 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");
    addNode(&prim, NULL);
    assert(prim != NULL);      /* vlad: double check that we inserted the 1st
                               * node.
                               * The list here should not be empty anymore (!)
                               */
    addNode(&prim, prim);
    addNode(&prim, prim->pNext);

    /* List traversal */
    puts("");
    fprintf(stdout, "Nodes' values...\n");
    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); )
        delNode(&prim, NULL); /* vlad: each time delete the 1st node.
                               * Eventually, the list is empty.
                               */
    /* A final traversal should report a 0-length value */
    listTraversal(&prim, &len); /* vlad: I only need the 1st node's 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;
}

```