

```

/* This is a personal header file. Call it double_list.h
 * Its name should be reflected into the macro definition (#define).
 * For instance, here I should say something like:
 *     #define __DOUBLE_LIST__
 */

#ifndef __DOUBLE_LIST__ /* this string SHOULD NOT previously exist */
#define __DOUBLE_LIST__


typedef struct Persoana
{
    char Nume[16];
    struct Persoana *pNext;
    struct Persoana *pPrec;
} PERS, *pPERS, **ppPERS;

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

/* Insertion and deletion functions */
/* If required to initialize each node's data (information fields)
 * then as last function argument I should have an entire structure
 */
void ins_node2(ppPERS, pPERS, char*);
void del_node2(ppPERS, pPERS);

/* vlad: compute the list's length inside each of these fcts */
void parcurg2_LR(pPERS, unsigned int*, ppPERS);
void parcurg2_RL(pPERS, unsigned int*, ppPERS);

/* check that the list is empty */
BOOL isEmpty(pPERS);

#endif

```

```

/* Another module in the project (double_list.c)
 * This one contains the implementation of the personal functions.
 */

#include "double_list.h"      /* should be placed within the same folder
                                * as the other modules belonging to this project
                                */
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <string.h>

/* vlad: the functions personal implementation */

void ins_node2(ppPERS ppPrim, pPERS p, char *Nume)
{
    pPERS q = (pPERS)malloc(sizeof(PERS));
    assert(q != NULL);

    strcpy(q->Nume, Nume);      /* vlad: update the information fields,
                                * once for any insert
                                */

    if(p == NULL) /* vlad: insert new node as the 1st node (mandatory: 'prim'
reference changes) */
    {
        q->pNext = *ppPrim;
        q->pPrec = NULL;
        if( *ppPrim ) (*ppPrim)->pPrec = q; /* typical for double-linked lists */
        *ppPrim = q;
    } else { /* insert in the list AFTER node p (as for simply linked lists) */
        q->pNext = p->pNext;
        q->pPrec = p;
        if(p->pNext)
        {
            p->pNext->pPrec = q; /* vlad: save the current link */
            p->pNext = q; /* vlad: update the new link; the old link vanishes */
        }
    }
    return;
}

void del_node2(ppPERS ppPrim, pPERS p)      /* vlad: here, p is the preceding node
                                                * and NOT the one about to be
                                                * deleted!
                                                */
{
    pPERS q /*, prec, urm*/;

    /* vlad: in the first place deal with the links... */
    if(p == NULL)
    {
        q = *ppPrim;      /* vlad: retain the address of the first node:
                                * this will be deleted
                                */
        if( (*ppPrim)->pNext )
        {
            (*ppPrim)->pNext->pPrec = NULL; /* the former second node won't have
                                                * any previous node anymore
                                                */
        }
        *ppPrim = (*ppPrim)->pNext; /* vlad: 'by-pass' the first node */
    }
}

```

```

} else {
    q = p->pNext;      /* the node about to be deleted */
    if(p->pNext) p->pNext->pPrec = p;
    p->pNext = q->pNext;

#ifndef 0
    /* This alternate approach uses some helpful notations.
     * It is perfectly valid
     */
    prec = q->pPrec; urm = q->pNext;      /* useful variables */
    prec->pNext = urm; /* by-pass node q: forward link */
    urm->pPrec = prec; /* by-pass node q: backward link */
#endif
}

/* ... and finally delete the node */
free(q);
return;
}

/* vlad: left-to-right traversal */
void parcurg2_LR(pPERS pStart, unsigned int *pLen, ppPERS ppUltim)
{
    assert(pStart != NULL);      /* vlad: just in case.
                                   * Shouldn't traverse an empty list (forbidden!) */
    /*/
    pPERS ref = pStart;
    *pLen = 0;      /* length is a sum-kind variable */

    do {
        puts(ref->Nume);    /* data processing */
        (*pLen)++;          /* list's length is computed */
        *ppUltim = ref;      /* vlad: the last one points to the very last node */
        ref = ref->pNext;   /* go 'forward' (advance in the list) */
    } while(ref != NULL); /* vlad: stop condition */

    return;
}

/* vlad: right-to-left traversal */
void parcurg2_RL(pPERS pStart, unsigned int *pLen, ppPERS ppPrim)
{
    assert(pStart != NULL);      /* vlad: just in case.
                                   * Shouldn't traverse an empty list (forbidden!) */
    /*/
    pPERS ref = pStart;         /* vlad: from outside, the pStart should now point
                                 * to the LAST node(!) */
    *pLen = 0;
    do {
        puts(ref->Nume);    /* data processing of each node (here it is minimal) */
        (*pLen)++;          /* compute the list's length */
        *ppPrim = ref;        /* vlad: at the last pass I retain 1st node's address */
        ref = ref->pPrec;   /* go 'backward' */
    } while(ref != NULL); /* vlad: the 1st node's prev link is NULL */

    return;
}

BOOL isEmpty(pPERS prim) { return (prim == NULL) ? True : False; }

```

```

/* The module file containing the main() function. Its name is main.c
 * The presence of this file gives this project the ability to run.
 * It contains calls to personal functions (the double-linked list's API).
 */

#include "double_list.h"          /* should be placed within the same folder
                                     * as the other modules belonging to this project
                                     */

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

int main()
{
    unsigned int dim = 0;
    pPERS prim = NULL;           /* vlad: right now the list is empty (!) */
    pPERS ultim = NULL;          /* vlad: here I will store the address
                                     * of the last node in the list.
                                     */
    pPERS prim_addr_computed = NULL;
                                /* I won't use the (same) variable 'prim'.
                                 * I keep 'prim' for the 1st node's address, and
                                 * 'prim_addr_computed' for the 1st node
                                 * address, computed after the list traversal.
                                */

/* vlad: create list. For instance 4 nodes.
 * The insertion takes place in several positions.
 */
fprintf(stdout,"Create the list...\n");
ins_node2(&prim, NULL, "N1");
fprintf(stdout, "Prim's address: %p\n", prim);
fprintf(stdout,"Inserted 1st node... OK\n");
ins_node2(&prim, prim->pNext, "N2");
fprintf(stdout, "Prim's address: %p\n", prim);
fprintf(stdout,"Inserted another node... OK\n");
ins_node2(&prim, prim->pNext->pNext, "N3");
fprintf(stdout, "Prim's address: %p\n", prim);
fprintf(stdout,"Inserted another node... OK\n");
ins_node2(&prim, NULL, "N4");
fprintf(stdout, "Prim's address: %p\n", prim);
fprintf(stdout,"Inserted another node as the 1st one... OK\n");

/* vlad: pass through the list */
fprintf(stdout,"Scan through the list...\n");
parcurg2_LR(prim, &dim, &ultim);      /* vlad: forward traversal */
fprintf(stdout,"Done passing through the list (L-R). Computed dim: %u\n",
        dim);
fprintf(stdout, "Last node's address (from traversal): %p\n", ultim);

parcurg2_RL(ultim, &dim, &prim_addr_computed); /* backward traversal */
fprintf(stdout,"Done passing through the list (R-L). Computed dim: %u\n",
        dim);
fprintf(stdout, "1st node's address (from traversal): %p\n",
        prim_addr_computed);

```

```
/* vlad: delete the list (list's clean-up) */
fputs("Now delete the list...\n", stdout);
do {          /* the loop here can be a for(), as well.
               * Please give the equivalence using for()
               */
    del_node2(&prim, NULL);
} while(!isEmpty(prim));
fputs("Done deleting the list.\n", stdout);

/* Double-check the correctness of the previous deletion operation */
fprintf(stdout, "Is the list empty now? %s\n", isEmpty(prim) ? "Yes":"No");
fprintf(stdout, "Double-check: the prim pointer has the value: %p\n",
        prim);      /* should be NULL */
fprintf(stdout, "All done!\n");

return 0;
}
```