```c
//Ej 1 lab04
typedef struct _person {
    int age;
    char name_initial;
} person_t;

int main(void) {

    int x = 1;
    person_t m = {90, 'M'};
    int a[] = {0, 1, 2, 3};

    //Point to x direction and change the value of
that direction
    aux_int = &x;
    *aux_int = 9;

    //Point to a[1] direction and change the value
of that direction
    aux_int = &a[1]; //aux_int = a + 1. Podrias
haber hecho *(a + 1) = 42
    *aux_int = 42;

    //Point to m direction and change the value of
the components of that direction
    aux_person = &m;
    aux_person->age=100;
    aux_person→name_initial='F';



//Ej 2 lab04
void absolute(int x, int *y) {
   *y = (x >= 0)? x : -x;

void swap(int *a, int *b) {
    int aux = *a;
    *a = *b;
    *b = aux;

//Ej3 lab04
void print_data(data_t *d) {
    printf("NOMBRE: %s\n"
           "EDAD  : %d años\n"
           "ALTURA: %d cm\n\n",
           d->name, d->age, d->height);
}
int main(void) {

    //Ask for memory
    data_t *messi = malloc(sizeof(*messi));
    if (!messi) {
       perror("malloc failed");
       return EXIT_FAILURE;
    }

    //Initilize values
    strcpy(messi->name, "Leo Messi");
    messi->age = 36;
    messi->height = 169;

    //Print data
    print_data(messi);

    //Size in bytes
    printf("name-size  : %lu bytes\n"
           "age-size   : %lu bytes\n"
           "height-size: %lu bytes\n"
           "data_t-size: %lu bytes\n",
           sizeof(messi->name),
           sizeof(messi->age),
           sizeof(messi->height),
           sizeof(data_t));

    //Memory address
    printf("\n\nname-memory address  : %p\n"
           "age-memory address   : %p\n"
           "height-memory address: %p\n"
           "data_t-memory address: %p\n",
           (void *) &messi->name,
           (void *) &messi->age,
           (void *) &messi->height,
           (void *) &messi);

    //Indexes
    printf("\n\nname-index  : %lu\n"
           "age-index   : %lu\n"
           "height-index: %lu\n"
           "data_t-index: %lu\n",
           (uintptr_t) &messi->name,
           (uintptr_t) &messi->age,
           (uintptr_t) &messi->height,
           (uintptr_t) &messi);

    free(messi);

    return EXIT_SUCCESS;
}



//Ej3b lab04
    int *array=NULL;
    array = malloc(sizeof(int) * size); //One per
element
    *length = size;



//Ej4a lab04
size_t string_length(const char *str) {
    size_t length = 0;

    while(str[length] != '\0')
        length++;

    return length;
}

char *string_filter(const char *str, char c) {
    if (!str)
        return NULL;

    size_t length = string_length(str);
    char *out = calloc(length + 1, sizeof(char));

    if (!out)
        return NULL;

    size_t i = 0, j = 0;
    while (str[i] != '\0') {
        if (str[i] != c) {
            out[j] = str[i];
            j++;
        }
        i++;
    }

    out[j] = '\0';
    return out;
}

bool string_is_symmetric(const char *str) {
    if  (!str)
        return false;

    bool res = true;
    size_t j = string_length(str);

    if (j < 2)
        return true;

    size_t i = 0;
    j = j -1;
    while (i < j) {
        if (str[i] != str[j]) {
            res = false;
        }
        i++; j--;
    }

    return res;
}
```

```
//Ej4c lab04
char *string_clone(const char *str, size_t length)
{
    if (!str)
        return NULL;

    //Ask for memory
    char *output = malloc(length + 1);
    if (!output) {
        fprintf(stderr, "malloc error");
        return NULL;
    }

    for (size_t i=0; i<length;i++) {
        output[i] = str[i];
    }
    output[length] = '\0';
    return output;
}
```

Si quiero usar punteros agrego esto al main:

```
    char *copy=NULL;
    size_t length = string_length(original);
    copy = string_clone(original, length);
```

Con strlen y strcpy(string.h):

```
char *string_clone(const char *str) {
    if (!str)
        return NULL;

    //char *clone=NULL;
    size_t length = strlen(str);
    char *output = malloc(length + 1);
    if (!output) {
        fprintf(stderr, "malloc error");
        return NULL;
    }
    strcpy(output, str);
    return output;
}
```

```
//Ej2 lab04-2 setup.ayed
List setup_example() {
    int i = 3; //Nodes quantity
    //Create my_list
    Node *my_list = malloc(sizeof(Node));
    if (!my_list) {
        fprintf(stderr, "malloc error");
        exit(EXIT_FAILURE);
    }

    //Create first node
    Node *a_node = my_list;
    i = i - 1;

    //Create intermediate nodes
    while (i > 0) {
        a_node->data = i * 10;
        a_node->next = malloc(sizeof(Node));
        if (!a_node->next) {
            fprintf(stderr, "malloc error");
            exit(EXIT_FAILURE);
        }

        a_node = a_node->next;
        i--;
    }

    //Last node
    a_node->data = 0;
    a_node->next = NULL;

    return my_list;
}
```

```
//Ej3 lab04-2 agrega un elemento al final
void append_example(List xs) {
    Node *a_node = xs;

    //Moving forward
    while (a_node->next != NULL) {
        a_node = a_node->next;
    }

    //New node to add
    a_node->next = malloc(sizeof(Node));
    if (!a_node->next) {
        fprintf(stderr, "malloc error");
        exit(EXIT_FAILURE);
    }

    //Put values as append.ayed
    a_node->next->data = 88;
    a_node->next->next = NULL;
}
```

```
//Ej4 lab04-2 Elimina el primer elemento de lista
List tail_example(List xs) {
    if (xs == NULL) {
        fprintf(stderr, "empty list");
        exit(EXIT_FAILURE);
    }

    //Two steps in one line
    Node *a_node = xs->next;

    //Cannot free a_node
    free(xs);
    return a_node;
}
```

```
//Ej1d lab05 TAD PAR. .h:
typedef struct s_pair_t * pair_t;

.c:
struct s_pair_t {
    int fst;
    int snd;
};

pair_t pair_new(int x, int y) {
    pair_t par = NULL;
    par = malloc(sizeof(struct s_pair_t));
    if (!par) {
        fprintf(stderr, "malloc error");
        exit(EXIT_FAILURE);
    }

    par->fst = x;
    par->snd = y;

    return par;
}

int pair_first(pair_t p) {
    if (!p) {
        fprintf(stderr, "malloc error");
        exit(EXIT_FAILURE);
    }
    return p->fst;
}
int pair_second(pair_t p) {
    if (!p) {
        fprintf(stderr, "malloc error");
        exit(EXIT_FAILURE);
    }
    return p->snd;
}
```

```
pair_t pair_swapped(pair_t p) {
    if (!p) {
        fprintf(stderr, "malloc error");
        exit(EXIT_FAILURE);
    }
    pair_t par = NULL;
    par = malloc(sizeof(struct s_pair_t));
    if (!par) {
        fprintf(stderr, "malloc error");
        exit(EXIT_FAILURE);
    }

    par->fst = p->snd;
    par->snd = p->fst;

    return par;
}

pair_t pair_destroy(pair_t p) {
    if (p != NULL) {
        free(p);
        p = NULL;
    }

    return p;
}
```

```
//Ej2 lab05-1 TAD CONTADOR .h:
typedef struct _counter * counter;

.c:
struct _counter {
    unsigned int count;
};

/* Constructors */
counter counter_init(void) {
    counter contador = NULL;
    contador = malloc(sizeof(counter));
    if (!contador) {
        exit(EXIT_FAILURE);
    }
    contador->count = 0;

    return contador;
}

void counter_inc(counter c) {
    c->count++;
}


/* Operations */
bool counter_is_init(counter c) {
    return c->count == 0 ? true : false;
}

void counter_dec(counter c) {
    assert(!counter_is_init(c));
    c->count--;
}

counter counter_copy(counter c) {
    counter copy = NULL;
    copy = counter_init();

    copy->count = c->count;

    return copy;
}

void counter_destroy(counter c) {
    if (c != NULL) {
        free(c);
        c = NULL;
    }
}
```

```
//Ej3 lab05 TAD LISTA listas enlazadas .h:
typedef int list_elem;

//This way is oppaque
typedef struct my_list *list;

.c:
struct my_list {
    list_elem value;
    struct my_list *next;
};

/* Constructors */
/* {- crea una lista vacia -} */
list empty() {
    list l = NULL;
    return l;
}

/* {- agrega el elemento al comienzo de la lista
-} */
list addl(list l, list_elem e) {
    list aux = malloc(sizeof(struct my_list));
    aux->value = e;
    aux->next = l;
    l = aux;
    return l;
}
```

```
/* Destroy */
/* {- libera memoria en caso que sea necesario -}
*/
list destroy(list l) {
    list aux;
    while (l != NULL) {
        aux = l;
        l = l->next;
        free(aux);
    }

    return l;
}



/* Operations */
/* {- devuelve True si l es vacia -} */
bool is_empty(list l) {
    return (l == NULL);
}

/* {- PRE: not is_empty() -} */
/* {- devuelve el primer elemento de la lista l -}
*/
list_elem head(list l) {
    assert(!is_empty(l));

    return l->value;
}

/* {- PRE: not is_empty() -} */
/* {- elimina el primer elemento de la lista l -}
*/
list tail(list l) {
    assert(!is_empty(l));
    list aux = l;
    l = l->next;
    free(aux);

    return l;
}

/* {- agrega el elemento e al final de la lista -}
*/
list addr(list l, list_elem e) {
    list aux = malloc(sizeof(struct my_list));
    aux->value = e;
    aux->next = NULL;
```

```
    if(is_empty(l)) {
        l = aux;
    } else {
        list aux2 = l;
        while (aux2->next != NULL) {
            aux2 = aux2->next;
        }
        aux2->next = aux;
    }

    return l;
}

/* {- devuelve la cantidad de elementos de la
lista l -} */
unsigned int length(list l) {
    unsigned int length = 0;
    list aux = l;

    while (aux != NULL) {
        aux = aux->next;
        length++;
    }

    return length;
}

/* {- agrega al final de l todos los elementos de
IO en el mismo orden} */
//No need to use malloc because is a list with
another list
list concat(list l, list IO) {
    if (is_empty(l)) {
        l = IO;
    } else {
        list aux = l;
        while (aux != NULL) {
            aux = aux->next;
        }
        aux->next = IO;
    }

    return l;
}

/* {- PRE: n < length(l) -} */
/* {- devuelve el n-esimo elemento de la lista -}
*/
list_elem index(list l, unsigned int e) {
    assert(e < length(l));
    list aux = l;
    unsigned int count = 0;

    while (count < e) {
        aux = aux->next;
        count++;
    }

    return aux->value;
}

/* {- deja en l solo los primeros n elementos
eliminando el resto -} */
list take(list l, int n) {
    if (n == 0) {
        destroy(l);
        return NULL;
    } else if (n < length(l)) {
        list aux = l;
        int count = 0;
        while (count < n-1) {
            aux = aux->next;
            count++;
        }
        list aux2 = aux->next;
        aux->next = NULL;
        destroy(aux2);
    }

    return l;
}
```

```
/* {- elimina los primeros n elementos de la lista
l -} */
list drop(list l, unsigned int n) {
    list aux = l;

    if (n > length(l)) {
        destroy(l);
        aux = NULL;
    } else {
        list aux2 = aux;
        while (n != 0) {
            free(aux);
            aux = aux2;
            aux2 = aux2->next;
            n--;
        }
    }

    return aux;
}

/* {- copia todos los elementos de la lista l1 en
la nueva lista l2 -} */
list copy_list(list l) {
    list aux = empty();
    list aux2 = l;

    while (aux2 != NULL) {
        aux = addr(aux, aux2->value);
        aux2 = aux2->next;
    }

    return aux;
}
```

```
//El ej4 es casi lo mismo, cambian los unsigned
int por int.

typedef int elem;
typedef struct _list * list;
```

```
//Ej5 lab05 arreglos1 .h:
typedef int elem;
typedef struct _list *list;

.c:
#define MAX_LENGTH 100

struct _list {
    elem elems[MAX_LENGTH];
    int size;
};

//typedef struct _list *list;


/* Constructors */
/* {- crea una lista vacia -} */
list empty() {
    list l = malloc(sizeof(struct _list));
    l->size = 0;
    return l;
}

/* {- agrega el elemento al comienzo de la lista
-} */
list addl(elem e, list l) {
    assert(l->size <= MAX_LENGTH-1);

    for (int i = l->size-1; i >= 0; i--) {
        l->elems[i+1] = l->elems[i];
    }

    l->size++;
    l->elems[0] = e;

    return l;
}


/* Operations */
/* {- devuelve True si l es vacia -} */
bool is_empty(list l) {
    return (l->size == 0);
}

/* {- PRE: not is_empty() -} */
/* {- devuelve el primer elemento de la lista l -}
*/
elem head(list l) {
    assert(!is_empty(l));

    return l->elems[0];
}

/* {- PRE: not is_empty() -} */
/* {- elimina el primer elemento de la lista l -}
*/
list tail(list l) {
    assert(!is_empty(l));
    for (int i = 0; i < l->size; i++) {
        //aux = l->elems[i+1];
        l->elems[i] = l->elems[i+1];
    }
    l->size--;
    return l;
}

/* {- agrega el elemento e al final de la lista -}
*/
list addr(list l, elem e) {
    assert(l->size <= MAX_LENGTH-1);

    l->size++;
    l->elems[l->size-1] = e;

    return l;
}

/* {- devuelve la cantidad de elementos de la
lista l -} */
int length(list l) {
    return l->size;
}

/* {- agrega al final de l todos los elementos de
IO en el mismo orden} */
//No need to use malloc because is a list with
another list
list concat(list l, list l0) {
    assert(l->size + l0->size <= MAX_LENGTH);

    if (is_empty(l)) {
        for (int i = 0; i < l0->size; i++) {
            l->size = l0->size;
            l->elems[i] = l0->elems[i];
        }
    } else if (l->size + l0->size <= MAX_LENGTH) {
        l->size = l->size + l0->size;
        for (int i = l->size - l0->size, j = 0; i
< l->size; i++, j++) {
            l->elems[i] = l0->elems[j];
        }
    }

    return l;
}

/* {- PRE: n < length(l) -} */
/* {- devuelve el n-esimo elemento de la lista -}
*/
elem index(list l, int n) {
    assert(n < length(l));

    return l->elems[n];
}

/* {- deja en l solo los primeros n elementos
eliminando el resto -} */
list take(list l, int n) {
    if (n > length(l))
        l->size = length(l);
    else
        l->size = n;

    return l;
}

/* {- elimina los primeros n elementos de la lista
l -} */
list drop(list l, int n) {
    if (n > l->size)
        l->size = 0;
    else {
        l->size = l->size - n;
    }

    if (n > 0) {
        for (int i = 0; i < l->size+n; i++) {
            l->elems[i] = l->elems[i+1];
        }
    }

    return l;
}

/* {- copia todos los elementos de la lista l1 en
la nueva lista l2 -} */
list copy_list(list l) {
    list l2 = malloc(sizeof(struct _list));
    l2->size = l->size;
    for (int i = 0; i < l->size; i++) {
        l2->elems[i] = l->elems[i];
    }
    return l2;
}
/* Destroy */
/* {- libera memoria en caso que sea necesario -}
*/
void destroy_list(list l) {
    if (l != NULL) {
        free(l);
    }
}
```

```
//Ej6 lab05 TAD LISTA arreglos circulares .h:
typedef int elem;
typedef struct _list *list;

.c:
#include <assert.h>
#include <stdbool.h>
#include <stdlib.h>

#include "list.h"
#define MAX_LENGTH 100

struct _list {
    elem a[MAX_LENGTH];
    int start;
    int size;
};

/* Constructors */
/* {- crea una lista vacia -} */
list empty() {
    list l = malloc(sizeof(struct _list));
    l->size = 0;
    l->start = 0;
    return l;
}

/* {- agrega el elemento al comienzo de la lista
-} */
list addl(elem e, list l) {
    if (l->size >= MAX_LENGTH)
        return l;

    l->start = (l->start - 1 + MAX_LENGTH) %
MAX_LENGTH;

    l->size++;
    l->a[l->start] = e;

    return l;
}



/* Operations */
/* {- devuelve True si l es vacia -} */
bool is_empty(list l) {
    return (l->size == 0);
}

/* {- PRE: not is_empty() -} */
/* {- devuelve el primer elemento de la lista l -}
*/
elem head(list l) {
    assert(!is_empty(l));

    return l->a[l->start];
}

/* {- PRE: not is_empty() -} */
/* {- elimina el primer elemento de la lista l -}
*/
list tail(list l) {
    assert(!is_empty(l));
    l->size--;
    l->start = (l->start + 1) % MAX_LENGTH;

    return l;
}

/* {- agrega el elemento e al final de la lista -}
*/
list addr(list l, elem e) {
    int index = (l->start + l->size) % MAX_LENGTH;
    l->size++;
    l->a[index] = e;

    return l;
}

/* {- devuelve la cantidad de elementos de la
lista l -} */
int length(list l) {

    return l->size;
}

/* {- agrega al final de l todos los elementos de
IO en el mismo orden} */
//No need to use malloc because is a list with
another list
list concat(list l, list l0) {
    assert(l->size + l0->size <= MAX_LENGTH);

    for (int i = 0; i < l0->size; i++) {
        int index_l0 = (l0-> start + i) %
MAX_LENGTH;    //position of l0 elems
        int index_l = (l->start + l->size) %
MAX_LENGTH; //End of l

        l->size++;
        l->a[index_l] = l0->a[index_l0];
    }

    return l;
}

/* {- PRE: n < length(l) -} */
/* {- devuelve el n-esimo elemento de la lista -}
*/
elem index(list l, int n) {
    assert(n < length(l));
    int index = (l->start + n) % MAX_LENGTH;
    return l->a[index];
}

/* {- deja en l solo los primeros n elementos
eliminando el resto -} */
list take(list l, int n) {
    if (n > length(l))
        l->size = length(l);
    else
        l->size = n;

    return l;
}

/* {- elimina los primeros n elementos de la lista
l -} */
list drop(list l, int n) {
    if (n >= l->size)
        l->size = 0;
    else
        l->size = l->size - n;

    l->start = (l->start + n) % MAX_LENGTH;

    return l;
}

/* {- copia todos los elementos de la lista l1 en
la nueva lista l2 -} */
list copy_list(list l) {
    list l2 = malloc(sizeof(struct _list));
    l2->size = l->size;
    l2->start = l->start;
    for (int i = 0; i < l->size; i++) {
        int index = (l->start + i) % MAX_LENGTH;
        l2->a[index] = l->a[index];
    }

    return l2;
}



/* Destroy */
/* {- libera memoria en caso que sea necesario -}
*/
void destroy_list(list l) {
    if (l != NULL) {
        free(l);
    }
}
```