Tema 4. LISTAS

Def. (lista): Colección de objetos donde todos menos uno tienen un objeto "siguiente" y todos menos uno tienen un objeto "anterior".

Funciones primitivas básicas:

```
Lista list_init()
void list_free(Lista l)
Boolean list_isEmpty(Lista l)           //No existe list_isFull!!!
Status list_insertIni(List l, Element e)    // Inserta al inicio
Element list_extractIni(Lista l)            // Extrae del inicio
Status list_insertEnd(List l, Element e)    // Inserta al final
Element list_extractEnd(Lista l)            // Extrae del final
```

(se pueden añadir otras: *list_insertPos(List l, Element e, int pos)*...)

LISTA ENLAZADA

Lo más conveniente es implementar la EdD de Lista como una **Lista Enlazada**:

- Una lista de Nodos. **Nodo**: (1) campo *info* (objeto a guardar),

(2) campo *next* (apunta al siguiente nodo de la lista; el último nodo apunta a NULL)

Para esta implementación se define la EdD Nodo en *list.c* y todas sus fiunciones asociadas:

---

list.h

```
typedef struct _List List;

List * list_init();
void list_free(List *pl);

Boolean list_isEmpty(List *pl);

Status list_insertIni(List *pl, Element *pe);
Element * list_extractIni(List *pl);

Status list_insertEnd(List *pl, Element *pe);
Element * list_extractEnd(List *pl);
```

---

list.c

Conviene utilizar los siguientes macros, para facilitar la implementación

```
#define first(plist) (plist)->first
#define next(pnode) (pnode)->next
#define info(pnode) (pnode)->info


typedef struct _Node {
    Element *info;
    struct _Node *next;
} Node;

struct _List {
    Node *first;
};


/*****funciones privadas******/

Node *node_create() {
    Node *pn = NULL;
    pn = (Node *) malloc(sizeof(Nodo));
    if (!pn) return NULL;
    pn->info = NULL; // info apuntará a un elemento
    pn->next = NULL;
    return pn;
}
```

```
void node_free(Node *pn) {
    if (!pn) return;

    element_free(pn->info); // Libera elemento de info
    free(pn);   //no liberamos *next, porque nos cargamos el siguiente
}

/******funciones públicas******/

List * list_init(){
    List *pl=NULL;

    if(!(pl=(List*)malloc(sizeof(List)))) return NULL;
    pl->first = NULL;
    return pl;
}

void list_free(List *pl){
    if(!pl) return ;

    while(list_isEmpty(pl)==FALSE)
        element_free(list_extractIni(pl));

    free(pl);
}

Boolean list_isEmpty(List *pl){
    if(!pl)return TRUE;  //caso de error
    if(first(pl)==NULL) return TRUE;
    return FALSE;
}

Status list_insertIni(List *pl, Element *pe){
    Node* pn=NULL;

    if(!pl||!pe)return ERROR;

    if(!(pn=node_create()))retrun ERROR;

    if(!(info(pn)=element_copy(pe))){
        node_free(pn);
        return ERROR;
    }
    next(pn) = first(pl);
    first(pl) = pn;
    return OK;
}

Element * list_extractIni(List *pl){
    Element *pe=NULL;
    Node *pn=NULL;

    if(!pl)return NULL;
    if(list_isEmpty(pl))return NULL;

    pn = first(pl);
    first(pl) = next(pn);

    pe = info(pn);
    info(pn) = NULL;

    node_free(pn);

    return pe;
}

Status list_insertEnd(List *pl, Element *pe){
    Node *pn = NULL, *qn = NULL;

    if(!pl||!pe) return ERROR;

    if(!(pn = node_create())) return ERROR;

    if(!(info(pn) = element_copy(pe))){
        node_freee(pn);
        rerturn ERROR;
    }

    if(list_isEmpty(pl)){
        first(pl) = pn;
        return OK;
    }

    for(qn = first(pl); next(qn) != NULL; qn = next(qn));

    next(qn) = pn;
```

```
    return OK;
}

Element * list_extractEnd(List *pl){
    Node *pn = NULL;
    Element *pe = NULL;

    if(!pl) return NULL;

    if(list_isEmpty(pl)) return NULL;

    if(next(first(pl)) == NULL) {
        pe = info(first(pl));
        info(first(pl)) = NULL;
        node_free(first(pl));
        first(pl) = NULL;
        return pe;
    }

    for(pn = first(pl); next(next(pn)) != NULL; pn = next(pn))
        ;
    pe = info(next(pn));
    info(next(pn)) = NULL;
    node_free(next(pn));
    next(pn) = NULL;

    return pe;
}


/*otra forma de implementar la función list_free() (EdD)*/

void list_free(List* pl){
    Node *pn = NULL;

    if(!pl) return;

    while (first(pl) != NULL){
        pn = first(pl);
        first(pl) = next(pl);
        node_free(pn);
    }
    free(pl);
}
```