Listing 1: Cdigo de bignum

```
#include "funciones.h"

void liberar_memoria_bignum (bignum_t* bignum) {

    if (bignum != NULL) {
        if ((*bignum).digits!=NULL)
            free((*bignum).digits);
        free(bignum);
    }
}
bignum_t* inicializar_bignum(void) {

    bignum_t* operando;/*o resultado*/

    if ((operando = (bignum_t*)malloc(sizeof(bignum_t))) == NULL){
        return NULL;
    }
    operando->longitud = 0;
    operando->sign = POSITIVO;
    (*operando).digits = NULL;
    return operando;
}
bignum_t* copiar_bignum(const bignum_t* bignum_in) {

    int i;
    bignum_t* resultado = NULL;

    if ((resultado = (bignum_t*)malloc(sizeof(bignum_t))) == NULL){
        return NULL;
    }
    (*resultado).longitud=(*bignum_in).longitud;
    (*resultado).sign=(*bignum_in).sign;

    (*resultado).digits=(unsigned short*)malloc(sizeof(unsigned
        short)*(*bignum_in).longitud);
    for (i=0;i < (*resultado).longitud;i++) {
        ((*resultado).digits)[i]=((*bignum_in).digits)[i];
    }
    return resultado;
}
/* Resta los modulos de dos bignum (|A|-|B|). El signo del resultado
     es calculado.
 * Devuelve NULL si hay un error de memoria.
 */
status_t agregar_ceros_a_la_izquierda(bignum_t* operando,int
    cantidad) {

    int i;
    /*cambiar la longitud y asignar memoria para nuevos digitos*/
    (*operando).longitud += cantidad;
    if (((*operando).digits = (unsigned short*)realloc((*operando).
        digits,sizeof(unsigned short)*(*operando).longitud)) == NULL
        ){
        return ERROR_ALLOCATING_MEMORY;
    }
    /*desplazar digitos a la derecha*/
    for (i = (*operando).longitud-cantidad-1; i >= 0; i--) {
        ((*operando).digits)[i+cantidad]=((*operando).digits)[i];
    }

    /*cargar con ceros los nuevos digitos*/
```

```
    for (i = 0; i<cantidad ;i++) {
        ((*operando).digits)[i]=0;
    }
    return OK;
}
```

Listing 2: Cdigo de bignuasdsadm

```
#include "funciones.h"

void liberar_memoria_bignum (bignum_t* bignum) {

    if (bignum != NULL) {
        if ((*bignum).digits!=NULL)
            free((*bignum).digits);
        free(bignum);
    }
}
bignum_t* inicializar_bignum(void) {

    bignum_t* operando;/*o resultado*/

    if ((operando = (bignum_t*)malloc(sizeof(bignum_t))) == NULL){
        return NULL;
    }
    operando->longitud = 0;
    operando->sign = POSITIVO;
    (*operando).digits = NULL;
    return operando;
}
bignum_t* copiar_bignum(const bignum_t* bignum_in) {

    int i;
    bignum_t* resultado = NULL;

    if ((resultado = (bignum_t*)malloc(sizeof(bignum_t))) == NULL){
        return NULL;
    }
    (*resultado).longitud=(*bignum_in).longitud;
    (*resultado).sign=(*bignum_in).sign;

    (*resultado).digits=(unsigned short*)malloc(sizeof(unsigned
        short)*(*bignum_in).longitud);
    for (i=0;i < (*resultado).longitud;i++) {
        ((*resultado).digits)[i]=((*bignum_in).digits)[i];
    }
    return resultado;
}
/* Resta los modulos de dos bignum (|A|-|B|). El signo del resultado
     es calculado.
 * Devuelve NULL si hay un error de memoria.
 */
status_t agregar_ceros_a_la_izquierda(bignum_t* operando,int
    cantidad) {

    int i;
    /*cambiar la longitud y asignar memoria para nuevos digitos*/
    (*operando).longitud += cantidad;
    if (((*operando).digits = (unsigned short*)realloc((*operando).
        digits,sizeof(unsigned short)*(*operando).longitud)) == NULL
        ){
```

```
        return ERROR_ALLOCATING_MEMORY;
    }
    /*desplazar digitos a la derecha*/
    for (i = (*operando).longitud-cantidad-1; i >= 0; i--) {
        ((*operando).digits)[i+cantidad]=((*operando).digits)[i];
    }

    /*cargar con ceros los nuevos digitos*/
    for (i = 0; i<cantidad ;i++) {
        ((*operando).digits)[i]=0;
    }
    return OK;
}
```
