Guía de Ejercicios 11 - Estructuras de Datos

```
Advertencia

La resolución conjunta o grupal de los ejercicios aquí presentes no está permitida, excepto en
la medida en que puedas pedir ayuda a tus compañeros de clase y a otras personas, y siempre que
esa ayuda no se reduzca a que otro haga el trabajo por vos.

El código fuente entregado por un estudiante debe ser escrito en su totalidad por dicha persona.
```

Condiciones de entrega:

¿Qué se entrega?	¿Qué no se entrega?
Archivos fuente/source (.c)	Archivos objeto (.o)
Archivos encabezado/header (.h)	Archivos ejecutables (programa, app, a.out, etc.)
Bibliotecas específicas (.a)	

Se deben entregar los ejercicios en un archivo zip (usar template como ayuda para el formato).

Importante: Recordá que se evaluarán las buenas prácticas de programación con respecto al pedido de memoria. Por lo tanto, ¡no te olvides de **liberar** la memoria pedida! (podés ayudarte del programa `valgrind` para verificarlo).

Tené en cuenta también que para **todos** los ejercicios, se pide también implementar (y entregar) una función main que demuestre el correcto funcionamiento del módulo.

Ejercicio 11.1

Desarrollá un módulo para el manejo de arreglos dinámicos (vectores). Utilizá los siguientes prototipos:

```c
typedef struct Dato
{
    char usuario[30];
    int edad;
} Dato;

typedef enum Status
{
    EXITO = 0,
    ERROR_FALLA_DE_MEMORIA,
    ERROR_POSICION_INVALIDA,
    ERROR_VECTOR_VACIO
} Status;

// @brief Agrega un elemento en el vector.
Status agregar(Dato** vector, int* largo, Dato dato);

// @brief Quita el elemento presente en la posición indicada.
Status eliminar(Dato** vector, int* largo, int posicion);

// @brief Devuelve el valor del elemento presente en la posición indicada.
Status obtener(Dato* vector, int largo, int posicion, Data* dato);

// @brief Imprime el contenido del vector.
Status imprimir(Dato* vector, int largo);

// @brief Destruye el vector.
void destruir(Dato** vector, int* largo);
```

Ejercicio 11.2

Desarrollá un módulo para el manejo de pilas. Dicho módulo deberá utilizar un arreglo dinámico (vector) para almacenar la información. Utilizá los siguientes prototipos:

```
typedef struct Dato
{
    char usuario[30];
    int edad;
} Dato;

typedef enum Status
{
    EXITO = 0,
    ERROR_FALLA_DE_MEMORIA,
    ERROR_PILA_VACIA
} Status;

// @brief Inserta un elemento en la pila.
Status push(Dato** pila, int* largo, Dato dato);

// @brief Quita un elemento de la pila.
Status pop(Dato** pila, int* largo, Dato* dato);

// @brief Devuelve el valor del próximo elemento a quitar de la pila (sin quitarlo).
Status espiar(Dato* pila, int largo, Dato* dato);

// @brief Imprime el contenido de la pila.
void imprimir(Dato* pila, int largo);

// @brief Destruye la pila.
void destruir(Dato** pila, int* largo);
```

Ejercicio 11.3

Desarrollá un módulo para el manejo de colas. Dicho módulo deberá utilizar un arreglo dinámico (vector) para almacenar la información. Utilizá los siguientes prototipos:

```
typedef struct Dato
{
    char usuario[30];
    int edad;
} Dato;

typedef enum Status
{
    EXITO = 0,
    ERROR_FALLA_DE_MEMORIA,
    ERROR_COLA_VACIA
} Status;

// @brief Inserta un elemento en la cola.
Status encolar(Dato** cola, int* largo, Dato dato);

// @brief Quita un elemento de la cola.
Status desencolar(Dato** cola, int* largo, Dato* dato);

// @brief Devuelve el valor del próximo elemento a quitar de la cola (sin quitarlo).
Status espiar(Dato* cola, int largo, Dato* dato);

// @brief Imprime el contenido de la cola.
void imprimir(Dato* cola, int largo);
```

```
// @brief Destruye la cola.
void destruir(Dato** cola, int* largo);
```

Ejercicio 11.4

Desarrollá un módulo para el manejo de arreglos dinámicos (vectores). Para mejorar la abstracción y el encapsulamiento del módulo, la información correspondiente al vector deberá estar contenida en una estructura creada para tal fin. Utilizá los siguientes prototipos:

```
typedef struct Dato
{
    char usuario[30];
    int edad;
} Dato;

typedef struct Vector
{
    Dato* datos;
    int largo;
} Vector;

typedef enum Status
{
    EXITO = 0,
    ERROR_FALLA_DE_MEMORIA,
    ERROR_POSICION_INVALIDA,
    ERROR_VECTOR_VACIO
} Status;

// @brief Crea un vector.
Vector* crear();

// @brief Agrega un elemento en el vector.
Status agregar(Vector* vector, Dato dato);

// @brief Quita el elemento presente en la posición indicada.
Status eliminar(Vector* vector, int posicion);

// @brief Devuelve el valor del elemento presente en la posición indicada.
Status obtener(Vector* vector, int posicion, Dato* dato);

// @brief Imprime el contenido del vector.
void imprimir(Vector* vector);

// @brief Destruye el vector.
void destruir(Vector* vector);
```

Ejercicio 11.5

Desarrollá un módulo para el manejo de pilas. Dicho módulo deberá utilizar internamente un arreglo dinámico (vector) para almacenar la información. Para mejorar la abstracción y el encapsulamiento del módulo, la información correspondiente a la pila deberá estar contenida en una estructura creada para tal fin. Utilizá los siguientes prototipos:

```
typedef struct Dato
{
    char usuario[30];
    int edad;
} Dato;

typedef struct Pila
{
    Dato* datos;
    int largo;
```

```
} Pila;

typedef enum Status
{
    EXITO = 0,
    ERROR_FALLA_DE_MEMORIA,
    ERROR_PILA_VACIA
} Status;

// @brief Crea una pila.
Pila* crear();

// @brief Inserta un elemento en la pila.
Status push(Pila* pila, Dato dato);

// @brief Quita un elemento de la pila.
Status pop(Pila* pila, Dato* dato);

// @brief Devuelve el valor del próximo elemento a quitar de la pila (sin quitarlo).
Status espiar(Pila* pila, Dato* dato);

// @brief Imprime el contenido de la pila.
void imprimir(Pila* pila);

// @brief Destruye la pila.
void destruir(Pila* pila);
```

Ejercicio 11.6

Desarrollá un módulo para el manejo de colas. Dicho módulo deberá utilizar internamente un arreglo dinámico (vector) para almacenar la información. Para mejorar la abstracción y el encapsulamiento del módulo, la información correspondiente a la cola deberá estar contenida en una estructura creada para tal fin. Utilizá los siguientes prototipos:

```
typedef struct Dato
{
    char usuario[30];
    int edad;
} Dato;

typedef struct Cola
{
    Dato* datos;
    int largo;
} Cola;

typedef enum Status
{
    EXITO = 0,
    ERROR_FALLA_DE_MEMORIA,
    ERROR_COLA_VACIA
} Status;

// @brief Crea una cola.
Cola* crear();

// @brief Inserta un elemento en la cola.
Status encolar(Cola* cola, Dato dato);

// @brief Quita un elemento de la cola.
Status desencolar(Cola* cola, Dato* dato);

// @brief Devuelve el valor del próximo elemento a quitar de la cola (sin quitarlo).
Status espiar(Cola* cola, Dato* dato);
```

```c
// @brief Imprime el contenido de la cola.
void imprimir(Cola* cola);

// @brief Destruye la cola.
void destruir(Cola* cola);
```

Ejercicio 11.7

Desarrollá un módulo para el manejo de pilas. Dicho módulo deberá utilizar internamente estructuras autorreferenciadas para almacenar la información. Utilizá los siguientes prototipos:

```c
typedef struct Dato
{
    char usuario[30];
    int edad;
} Dato;

typedef struct Nodo
{
    Dato dato;
    struct Nodo* siguiente;
} Nodo;

typedef enum Status
{
    EXITO = 0,
    ERROR_FALLA_DE_MEMORIA,
    ERROR_PILA_VACIA
} Status;

// @brief Inserta un elemento en la pila.
Status push(Nodo** pila, Dato dato);

// @brief Quita un elemento de la pila.
Status pop(Nodo** pila, Dato* dato);

// @brief Devuelve el valor del próximo elemento a quitar de la pila (sin quitarlo).
Status espiar(Nodo* pila, Dato* dato);

// @brief Imprime el contenido de la pila.
void imprimir(Nodo* pila);

// @brief Destruye la pila.
void destruir(Nodo** pila);
```

Ejercicio 11.8

Desarrollá un módulo para el manejo de colas. Dicho módulo deberá utilizar internamente estructuras autorreferenciadas para almacenar la información. Utilizá los siguientes prototipos:

```c
typedef struct Dato
{
    char usuario[30];
    int edad;
} Dato;

typedef struct Nodo
{
    Dato dato;
    struct Nodo* siguiente;
} Nodo;
```

```c
typedef enum Status
{
    EXITO = 0,
    ERROR_FALLA_DE_MEMORIA,
    ERROR_COLA_VACIA
} Status;

// @brief Inserta un elemento en la cola.
Status encolar(Nodo** cola, Dato dato);

// @brief Quita un elemento de la cola.
Status desencolar(Nodo** cola, Dato* dato);

// @brief Devuelve el valor del próximo elemento a quitar de la cola (sin quitarlo).
Status espiar(Nodo* cola, Dato* dato);

// @brief Imprime el contenido de la cola.
void imprimir(Nodo* cola);

// @brief Destruye la cola.
void destruir(Nodo** cola);
```

Referencias

Algunos ejercicios fueron obtenidos y adaptados de:

- Guía de Trabajos Prácticos 2011 - Informática I - Departamento de Electrónica - UTN FRBA