lista simple circular

Parcial	AutoEstudio

estructura base

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>

typedef struct {
    int dato;
    // Aquí puedes agregar otros campos si usas Estudiante
} Elemento;

typedef struct Nodo {
    Elemento info;
    struct Nodo* siguiente;
} Nodo;
```

✅ Inserción al inicio

```
bool insertarInicio(Nodo** final, Elemento nuevoDato) {
    Nodo* nuevo = (Nodo*)malloc(sizeof(Nodo));
    if (!nuevo) return false;

    nuevo->info = nuevoDato;
    if (*final == NULL) {
        nuevo->siguiente = nuevo;
        *final = nuevo;
    } else {
```

```
        nuevo->siguiente = (*final)->siguiente;
        (*final)->siguiente = nuevo;
    }
    return true;
}
```

✅ Inserción al final

```
bool insertarFinal(Nodo** final, Elemento nuevoDato) {
    Nodo* nuevo = (Nodo*)malloc(sizeof(Nodo));
    if (!nuevo) return false;

    nuevo->info = nuevoDato;
    if (*final == NULL) {
        nuevo->siguiente = nuevo;
        *final = nuevo;
    } else {
        nuevo->siguiente = (*final)->siguiente;
        (*final)->siguiente = nuevo;
        *final = nuevo;
    }
    return true;
}
```

✅ Inserción en posición específica

```
bool insertarEnPos(Nodo** final, Elemento nuevoDato, int pos) {
    if (pos < 0) return false;

    if (*final == NULL || pos == 0)
        return insertarInicio(final, nuevoDato);

    Nodo* actual = (*final)->siguiente;
    int i = 0;
```

```
    while (i < pos - 1 && actual != *final) {
        actual = actual->siguiente;
        i++;
    }

    Nodo* nuevo = (Nodo*)malloc(sizeof(Nodo));
    if (!nuevo) return false;

    nuevo->info = nuevoDato;
    nuevo->siguiente = actual->siguiente;
    actual->siguiente = nuevo;

    if (actual == *final)
        *final = nuevo;

    return true;
}
```

❌ Eliminación al inicio

```
bool eliminarInicio(Nodo** final) {
    if (*final == NULL) return false;

    Nodo* inicio = (*final)->siguiente;

    if (*final == inicio) {
        free(inicio);
        *final = NULL;
    } else {
        (*final)->siguiente = inicio->siguiente;
        free(inicio);
    }
    return true;
}
```

❌ Eliminación al final

```
bool eliminarFinal(Nodo** final) {
    if (*final == NULL) return false;

    Nodo* actual = (*final)->siguiente;

    if (actual == *final) {
        free(*final);
        *final = NULL;
    } else {
        while (actual->siguiente != *final) {
            actual = actual->siguiente;
        }
        actual->siguiente = (*final)->siguiente;
        free(*final);
        *final = actual;
    }
    return true;
}
```

❌ Eliminación en posición específica

```
bool eliminarEnPos(Nodo** final, int pos) {
    if (*final == NULL || pos < 0) return false;

    if (pos == 0)
        return eliminarInicio(final);

    Nodo* actual = (*final)->siguiente;
    int i = 0;

    while (i < pos - 1 && actual->siguiente != (*final)->siguiente) {
        actual = actual->siguiente;
```

```
        i++;
    }

    Nodo* temp = actual->siguiente;

    if (temp == (*final)->siguiente)
        return eliminarInicio(final);

    if (temp == *final)
        *final = actual;

    actual->siguiente = temp->siguiente;
    free(temp);
    temp = NULL;
    return true;
}
```

🔁 Recorrer lista circular

```
void recorrer(Nodo* final) {
    if (final == NULL) {
        printf("Lista vacía\n");
        return;
    }

    Nodo* actual = final->siguiente;
    do {
        printf("%d ", actual->info.dato); // Ajusta si usas estructura Estudiante
        actual = actual->siguiente;
    } while (actual != final->siguiente);
    printf("\n");
}
```

aplicado a Estudiante

```
int main() {
    Nodo* lista = NULL;
    int opcion, pos;
    Estudiante est;

    do {
        printf("\n--- MENU LISTA CIRCULAR DE ESTUDIANTES ---\n");
        printf("1. Insertar al inicio\n");
        printf("2. Insertar al final\n");
        printf("3. Insertar en posición\n");
        printf("4. Eliminar al inicio\n");
        printf("5. Eliminar al final\n");
        printf("6. Eliminar en posición\n");
        printf("7. Mostrar lista\n");
        printf("0. Salir\n");
        printf("Elige una opción: ");
        scanf("%d", &opcion);
        getchar(); // Limpia buffer de enter

        switch (opcion) {
            case 1:
            case 2:
            case 3:
                printf("Matricula: ");
                scanf("%d", &est.matricula);
                getchar(); // limpia enter

                printf("Nombre: ");
                fgets(est.nombre, sizeof(est.nombre), stdin);
                est.nombre[strcspn(est.nombre, "\n")] = 0; // quitar salto

                printf("¿Es regular? (1=Sí, 0=No): ");
                scanf("%d", (int*)&est.regular);
                getchar();
```

```
            if (opcion == 1)
                insertarInicio(&lista, est);
            else if (opcion == 2)
                insertarFinal(&lista, est);
            else {
                printf("Posición: ");
                scanf("%d", &pos);
                insertarEnPos(&lista, est, pos);
            }
            break;

        case 4:
            if (!eliminarInicio(&lista))
                printf("No se pudo eliminar. Lista vacía.\n");
            break;

        case 5:
            if (!eliminarFinal(&lista))
                printf("No se pudo eliminar. Lista vacía.\n");
            break;

        case 6:
            printf("Posición: ");
            scanf("%d", &pos);
            if (!eliminarEnPos(&lista, pos))
                printf("No se pudo eliminar. Posición inválida o lista vacía.\n");
            break;

        case 7:
            recorrerLista(lista);
            break;

        case 0:
            printf("Saliendo...\n");
            break;
```

```
            default:
                printf("Opción inválida. Intenta de nuevo.\n");
        }

    } while (opcion != 0);

    return 0;
}
```