Ejercicio 1: Estructura de Persona y Arreglo Estático

```
typedef struct {
    char nombre[30];
    char genero;
    int edad;
} persona;
```

// a. Crear un arreglo estático de 30 elementos de esta estructura y cargarlo mediante una función.

```
void cargarPersonas(persona arreglo[], int n) {
    for (int i = 0; i < n; i++) {
        printf("Ingrese nombre: ");
        scanf("%s", arreglo[i].nombre);
        printf("Ingrese genero (M/F): ");
        scanf(" %c", &arreglo[i].genero);
        printf("Ingrese edad: ");
        scanf("%d", &arreglo[i].edad);
    }
}
```

// b. Hacer una función que cuente la cantidad de un género determinado.

```
int contarGenero(persona arreglo[], int n, char generoBuscado) {
    int count = 0;
    for (int i = 0; i < n; i++) {
        if (arreglo[i].genero == generoBuscado) {
            count++;
        }
    }
    return count;
}
```

// c. **Hacer una función que copie los datos de un género determinado en un arreglo dinámico.**

```
persona* copiarGenero(persona arreglo[], int n, char generoBuscado, int* count) {
    *count = contarGenero(arreglo, n, generoBuscado);
    persona* nuevoArreglo = (persona*)malloc(*count * sizeof(persona));

    if (nuevoArreglo != NULL) {
        int j = 0;
        for (int i = 0; i < n; i++) {
            if (arreglo[i].genero == generoBuscado) {
                nuevoArreglo[j] = arreglo[i];
                j++;
            }
        }
    }
    return nuevoArreglo;}
```

Ejercicio 2: Ordenamiento por Selección (Edad)

```
// Algoritmo de ordenamiento por selección

void ordenarPorEdad(persona arreglo[], int n) {
    for (int i = 0; i < n - 1; i++) {
        int minIndex = i;
        for (int j = i + 1; j < n; j++) {
            if (arreglo[j].edad < arreglo[minIndex].edad) {
                minIndex = j;
            }
        }
        if (minIndex != i) {
```

```
            persona temp = arreglo[i];
            arreglo[i] = arreglo[minIndex];
            arreglo[minIndex] = temp;
        }
    }
}
```

Ejercicio 3:

```
typedef struct {
    int valores[100];
    int posTope;
} Pila;

void inicPila(Pila *p) {
    p->posTope = -1;
}

int pilavacia(Pila *p) {
    return p->posTope == -1;
}

void apilar(Pila *p, int valor) {
    if (p->posTope < 99) {
        p->posTope++;
        p->valores[p->posTope] = valor;
    } else {
        printf("La pila está llena, no se puede apilar.\n");
    }
}

int desapilar(Pila *p) {
    if (!pilavacia(p)) {
        int valor = p->valores[p->posTope];
        p->posTope--;
        return valor;
    } else {
        printf("La pila está vacía, no se puede desapilar.\n");
        return -1; // Valor sentinela para indicar un error
    }
}

int tope(Pila *p) {
    if (!pilavacia(p)) {
        return p->valores[p->posTope];
    } else {
        printf("La pila está vacía, no hay tope.\n");
        return -1; // Valor sentinela para indicar un error
    }
}
void mostrar(Pila *p) {
    if (!pilavacia(p)) {
        printf("Contenido de la pila:\n");
        for (int i = 0; i <= p->posTope; i++) {
            printf("%d ", p->valores[i]);
        }
        printf("\n");
    } else {
        printf("La pila está vacía.\n");
    }
}

void leer(Pila *p) {
    int valor;
    printf("Ingrese un valor: ");
    scanf("%d", &valor);
    apilar(p, valor);
}
```

```
int main() {
    Pila miPila;
    inicPila(&miPila);
    apilar(&miPila, 10);
    apilar(&miPila, 20);
    apilar(&miPila, 30);
    mostrar(&miPila);
    printf("Tope: %d\n", tope(&miPila));
    printf("Desapilado: %d\n", desapilar(&miPila));
    mostrar(&miPila);
    leer(&miPila);
    mostrar(&miPila);
    return 0;
}
```

Ejercicio 5: Función de Inserción en Arreglo Ordenado

```
// Función para insertar un elemento en un arreglo ordenado

void insertarOrdenado(int arreglo[], int n, int valor) {
    int i = n - 1;
    while (i >= 0 && arreglo[i] > valor) {
        arreglo[i + 1] = arreglo[i];
        i--;
    }
    arreglo[i + 1] = valor;
}
```

Ejercicio 6: Algoritmo de Ordenamiento por Inserción

```
// Algoritmo de ordenamiento por inserción

void ordenarPorInsercion(int arreglo[], int n) {
    for (int i = 1; i < n; i++) {
        int j = i - 1;
        int key = arreglo[i];
        while (j >= 0 && arreglo[j] > key) {
            arreglo[j + 1] = arreglo[j];
            j--;
        }
        arreglo[j + 1] = key;
    }}
```

Ejercicio 7: Función para Eliminar un Elemento de un Arreglo (Desplazamiento)

```
// Función para eliminar un elemento de un arreglo (desplazamiento)

int eliminarElemento(int arreglo[], int n, int valor) {
    int encontrado = 0;
    int pos = -1;

    for (int i = 0; i < n; i++) {
        if (arreglo[i] == valor) {
            encontrado = 1;
            pos = i;
            break;
        }
    }

    if (encontrado) {
        for (int i = pos; i < n - 1; i++) {
            arreglo[i] = arreglo[i + 1];
        }
    }

    return encontrado;
}
```