Punteros Simples:

1. **Intercambiar Valores:** Escribe un programa que intercambie los valores de dos variables utilizando punteros.

```
void intercambiar(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

int main() {
    int x = 5, y = 10;
    intercambiar(&x, &y);
    printf("x: %d, y: %d\n", x, y);
    return 0;
}
```

2. **Suma de Elementos:** Calcula la suma de los elementos de un arreglo usando punteros.

```
int sumaArreglo(int *arre, int dim) {
    int suma = 0;
    for (int i = 0; i < dim; i++) {
        suma += *arre; /// se suma el elemento apuntado en ese momento
        arre++; /// se "corre" el puntero al sig elemento del vector
    }
    return suma;
}

int main() {
    int numeros[] = {1, 2, 3, 4, 5};
    int dim = sizeof(numeros) / sizeof(numeros[0]);
    printf("Suma: %d\n", sumaArreglo(numeros, dim));
    return 0;
}
```

Tener en cuenta:

Cada elemento en un arreglo de enteros tiene su propia dirección de memoria. Cuando defines un puntero que apunta a un arreglo de enteros, el puntero apuntará al primer elemento del arreglo (índice 0). Sin embargo, a través de la aritmética de punteros, puedes acceder a los otros elementos del arreglo también.

Ejemplo:

```
int main() {
    int arreglo[5] = {10, 20, 30, 40, 50};
    int *puntero = arreglo;  // El puntero apunta al primer elemento del arreglo

    printf("Elemento 0: %d\n", *puntero);  // Imprime el valor del primer
elemento
    printf("Elemento 1: %d\n", *(puntero + 1));  // Accede al segundo elemento a
través de la aritmética de punteros

    return 0;
}
```

3. **Cadena en Mayúsculas:** Convierte una cadena de caracteres a mayúsculas utilizando punteros.

```
void convertirMayusculas(char *str) {
    while (*str) {
        *str = toupper(*str);
        str++;
    }
}

int main() {
    char cadena[] = "hola mundo";
    convertirMayusculas(cadena);
    printf("Cadena en mayusculas: %s\n", cadena);
    return 0;
}
```

4. **Eliminar Números Pares:** Elimina los números pares de un arreglo utilizando punteros.

```
void eliminarPares(int *arr, int *dim) {
    int *p = arr; // Puntero para recorrer el arreglo
    int newSize = 0; // Nuevo tamaño del arreglo después de eliminar los pares

    // Recorremos el arreglo original para contar los elementos impares y
      ajustar el tamaño
    for (int i = 0; i < *dim; i++) {
        if (*p % 2 != 0) {
            arr[newSize] = *p; // Sobrescribimos el arreglo original con
                                  elementos impares
            newSize++; // Incrementamos el nuevo tamaño
        }
        p++;
    }

    *dim = newSize; // Actualizamos el tamaño del arreglo
}

int main() {
    int numeros[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int dim = sizeof(numeros) / sizeof(numeros[0]);

    printf("Arreglo original:\n");
    for (int i = 0; i < dim; i++) {
        printf("%d ", numeros[i]);
    }
    printf("\n");

    eliminarPares(numeros, &dim);

    printf("Arreglo después de eliminar los pares:\n");
    for (int i = 0; i < dim; i++) {
        printf("%d ", numeros[i]);
    }
    printf("\n");

    return 0;
}
```

5. **Invertir Arreglo:** Invierte un arreglo utilizando punteros.

```
void invertirArreglo(int *arr, int dim) {
    int *inicio = arr;
    int *fin = arr + dim - 1;
    while (inicio < fin) {
        int temp = *inicio;
        *inicio = *fin;
        *fin = temp;
        inicio++;
        fin--;
    }
}

int main() {
    int numeros[] = {1, 2, 3, 4, 5};
    int dim = sizeof(numeros) / sizeof(numeros[0]);
    invertirArreglo(numeros, dim);
    for (int i = 0; i < dim; i++) {
        printf("%d ", numeros[i]);
    }
    printf("\n");
    return 0;
}
```

6. **Contar Vocales:** Cuenta el número de vocales en una cadena utilizando punteros.

```
#include <ctype.h>

int contarVocales(char *str) {
    int count = 0;
    while (*str) {
        char c = tolower(*str);
        if (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u') {
            count++;
        }
        str++;
    }
    return count;
}

int main() {
    char cadena[] = "Hola, esto es una cadena de prueba.";
    printf("Numero de vocales: %d\n", contarVocales(cadena));
    return 0;
}
```

7. **Copiar Cadena:** Copia una cadena en otra utilizando punteros.

```
void copiarCadena(char *dest, const char *src) {
    while (*src) {
        *dest = *src;
        dest++;
        src++;
    }
    *dest = '\0';
}

int main() {
    char origen[] = "Hola, esto es una cadena de prueba.";
    char destino[100];
    copiarCadena(destino, origen);
```

```
    printf("Destino: %s\n", destino);
    return 0;
}
```

8. **Encontrar Mínimo y Máximo:** Encuentra el valor mínimo y máximo en un arreglo utilizando punteros.

```
void encontrarMinimoMaximo(int *arre, int dim, int *min, int *max) {
    *min = *max = *arre;
    for (int i = 1; i < dim; i++) {
        if (*arre < *min) {
            *min = *arre;
        }
        if (*arre > *max) {
            *max = *arre;
        }
        arre++;
    }
}

int main() {
    int numeros[] = {34, 12, 56, 89, 7, 23};
    int dim = sizeof(numeros) / sizeof(numeros[0]);
    int min, max;
    encontrarMinimoMaximo(numeros, dim, &min, &max);
    printf("Minimo: %d, Maximo: %d\n", min, max);
    return 0;
}
```

9. **Concatenar Cadenas:** Concatena dos cadenas utilizando punteros.

```
void concatenarCadenas(char *dest, const char *src) {
    while (*dest) {
        dest++;
    }
    while (*src) {
        *dest = *src;
        dest++;
        src++;
    }
    *dest = '\0';
}

int main() {
    char cadena1[] = "Hola, ";
    char cadena2[] = "esto es una cadena de prueba.";
    char resultado[100];
    concatenarCadenas(resultado, cadena1);
    concatenarCadenas(resultado, cadena2);
    printf("Resultado: %s\n", resultado);
    return 0;
}
```

10. **Buscar Carácter:** Busca un carácter en una cadena utilizando punteros.

```
int buscarCaracter(const char *str, char c) {
    while (*str) {
        if (*str == c) {
            return 1;
        }
        str++;
```

```
    }
    return 0;
}

int main() {
    char cadena[] = "Hola, esto es una cadena de prueba.";
    char caracter = 'e';
    if (buscarCaracter(cadena, caracter)) {
        printf("El caracter '%c' esta presente en la cadena.\n", caracter);
    } else {
        printf("El caracter '%c' no esta presente en la cadena.\n", caracter);
    }
    return 0;
}
```

Funciones que Devuelven un Puntero:

1. **Crear Arreglo Dinámico:** Crea un arreglo dinámico de enteros y devuelve un puntero a él.

```
int *crearArregloDinamico(int dim) {
    int *arre = (int *)malloc(dim * sizeof(int));
    return arre;
}

int main() {
    int n = 5;
    int *ptr = crearArregloDinamico(n);
    if (ptr == NULL) {
        printf("No se pudo asignar memoria (crear arreglo).\n");
        return 1;
    }
    for (int i = 0; i < n; i++) {
        ptr[i] = i * 2;
        printf("%d ", ptr[i]);
    }
    free(ptr);
    printf("\n");
    return 0;
}
```

2. **Duplicar Cadena:** Duplica una cadena y devuelve un puntero a la nueva cadena.

```
#include <string.h>

char *duplicarCadena(const char *str) {
    char *duplicado = (char *)malloc((strlen(str) + 1) * sizeof(char));
    strcpy(duplicado, str);
    return duplicado;
}

int main() {
    char original[] = "Hola, esto es una cadena de prueba.";
    char *copia = duplicarCadena(original);
    printf("Copia: %s\n", copia);
    free(copia);
    return 0;
}
```

3 . **Clonar Arreglo Dinámico:** Clona un arreglo dinámico de enteros y devuelve un puntero al nuevo arreglo.

```
int *clonarArregloDinamico(int *arre, int dim) {
    int *clon = (int *)malloc(size * sizeof(int));
    for (int i = 0; i < dim; i++) {
        clon[i] = arre[i];
    }
    return clon;
}

int main() {
    int numeros[] = {2, 4, 6, 8, 10};
    int dim = sizeof(numeros) / sizeof(numeros[0]);
    int *copia = clonarArregloDinamico(numeros, dim);
    for (int i = 0; i < dim; i++) {
        printf("%d ", copia[i]);
    }
    printf("\n");
    free(copia);
    return 0;
}
```

4. **Crear Cadena Dinámica:** Crea una cadena dinámica y devuelve un puntero a ella.

```
#include <string.h>

char *crearCadenaDinamica(const char *str) {
    char *cadena = (char *)malloc((strlen(str) + 1) * sizeof(char));
    strcpy(cadena, str);
    return cadena;
}

int main() {
    char texto[] = "Esta es una cadena dinamica.";
    char *cadena = crearCadenaDinamica(texto);
    printf("Cadena: %s\n", cadena);
    free(cadena);
    return 0;
}
```

Punteros Dobles:

Un puntero en C es una variable que almacena la dirección de memoria de otro valor. Un puntero doble, por otro lado, almacena la dirección de memoria de otro puntero. Esto puede ser útil en situaciones en las que necesitas trabajar con punteros o arreglos de punteros.

Ejemplo:

```
int main() {
   int num = 42;
   int *ptr1 = &num; // Puntero simple que apunta a 'num'
```

```
    int **ptr2 = &ptr1; // Puntero doble que apunta a 'ptr1'


    printf("Valor de num: %d\n", num);
    printf("Valor apuntado por ptr1: %d\n", *ptr1);
    printf("Valor apuntado por ptr2 (usando puntero doble): %d\n", **ptr2);


    return 0;
}
```

En el ejemplo:

1. Declaramos una variable `num` de tipo entero y le asignamos el valor 42.
2. Declaramos un puntero simple `ptr1` que apunta a la dirección de memoria de `num`.
3. Declaramos un puntero doble `ptr2` que apunta a la dirección de memoria de `ptr1`.

Luego, si imprimimos, la salida será:

```
Valor de num: 42
Valor apuntado por ptr1: 42
Valor apuntado por ptr2 (usando puntero doble): 42
```

En este ejemplo, `ptr2` almacena la dirección de memoria de `ptr1`, que a su vez almacena la dirección de memoria de `num`. Al usar el puntero doble `ptr2`, podemos acceder al valor de `num` de manera indirecta.

Los punteros dobles son especialmente útiles cuando se trabaja con matrices de punteros, como matrices de cadenas de caracteres (arreglos de strings) o matrices de punteros a funciones.

1. **// Ejercicio 1: Función para intercambiar dos valores usando punteros dobles**

```
void swap(int **a, int **b) {
    int *temp = *a;
    *a = *b;
    *b = temp;
}
int main() {
    int num1 = 5, num2 = 10;
    int *ptr1 = &num1, *ptr2 = &num2;
    printf("Ejercicio 1: Intercambio de valores usando punteros dobles\n");
```

```
printf("Antes del intercambio: num1 = %d, num2 = %d\n", num1, num2);
swap(&ptr1, &ptr2);
printf("Después del intercambio: num1 = %d, num2 = %d\n", num1, num2);
```

2. **// Ejercicio 2: Uso de punteros dobles para acceder a un arreglo**

```
int arr[] = {1, 2, 3, 4, 5};
int *arrPtr = arr;
int i;
printf("\nEjercicio 2: Acceso a un arreglo usando punteros dobles\n");
for (i = 0; i < sizeof(arr) / sizeof(int); i++) {
    printf("Elemento %d: %d\n", i, *(*(&arrPtr) + i));
}
```

3. **// Ejercicio 3: Pasar un arreglo a una función usando punteros dobles**

```
void printArray(int **arrPtr, int size) {
    for (i = 0; i < size; i++) {
        printf("Elemento %d: %d\n", i, *(*arrPtr + i));
    }
}
printf("\nEjercicio 3: Pasar un arreglo a una función usando punteros dobles\n");
printArray(&arrPtr, sizeof(arr) / sizeof(int));
return 0;}
```