Índice de Contenidos

1 Introducción

1.1 Definición y conceptos básicos

En programación imperativa, un arreglo unidimensional es una estructura de datos que permite almacenar elementos del mismo tipo de forma contigua en memoria. También se le conoce como vector o array.

Puedes ver un arreglo unidimensional como una lista ordenada de elementos, donde cada elemento tiene una posición única dentro del arreglo. Estas posiciones comienzan desde 0 y van hasta el tamaño del arreglo menos 1. Por ejemplo, en un arreglo de tamaño 5, las posiciones serían 0, 1, 2, 3 y 4.

Una forma sencilla de visualizar un arreglo unidimensional es imaginar una fila de casillas, donde cada casilla representa un elemento del arreglo. Cada casilla tiene una dirección de memoria única y puede contener un valor del tipo de datos especificado para el arreglo.

En muchos lenguajes de programación, puedes acceder a los elementos de un arreglo utilizando su índice. Por ejemplo, en pseudocódigo:

```
arreglo = [5, 10, 15, 20, 25]
```

Aquí tenemos un arreglo de enteros con 5 elementos. Para acceder a un elemento específico del arreglo, simplemente indicamos su posición o índice. Por ejemplo, `arreglo[0]` nos daría el valor 5, `arreglo[2]` nos daría el valor 15, y así sucesivamente.

Los arreglos unidimensionales son muy útiles para almacenar colecciones de datos que necesitan ser accedidos de manera secuencial y eficiente. Pueden ser utilizados para representar listas, vectores, pilas, colas y muchos otros tipos de estructuras de datos en programación.

2 Declaración de Arreglos

En C, puedes declarar un arreglo unidimensional utilizando la siguiente sintaxis:

```
tipo_de_dato nombre_del_arreglo[tamaño];
```

Donde `tipo_de_dato` es el tipo de datos de los elementos del arreglo (por ejemplo, `int`, `float`, `char`, etc.), `nombre_del_arreglo` es el nombre que le das al arreglo, y `tamaño` es el número de elementos que puede contener el arreglo.

Por ejemplo, para declarar un arreglo de enteros con 5 elementos, lo harías de la siguiente manera:

```
int numeros[5];
```

Esto declara un arreglo llamado `numeros` que puede contener 5 elementos enteros.

También puedes inicializar el arreglo al momento de declararlo, proporcionando los valores entre llaves {} en la misma línea de declaración. Por ejemplo:

```
int numeros[5] = {1, 2, 3, 4, 5};
```

Esto inicializa el arreglo `numeros` con los valores 1, 2, 3, 4 y 5 en las primeras cinco posiciones.

3 Inicialización de Arreglos

En C, puedes inicializar un arreglo durante su declaración o después de haberlo declarado. Aquí te mostraré ambos casos:

1. Inicialización durante la declaración:

Durante la declaración del arreglo, puedes proporcionar los valores iniciales entre llaves {} en el mismo momento. La sintaxis es la siguiente:

```
tipo_de_dato nombre_del_arreglo[tamaño] = {valor1, valor2, ..., valorN};
```

Donde:

- `tipo_de_dato` es el tipo de dato de los elementos del arreglo.
- `nombre_del_arreglo` es el nombre que le das al arreglo.
- `tamaño` es el número de elementos que puede almacenar el arreglo.
- `valor1, valor2, ..., valorN` son los valores iniciales que deseas asignar al arreglo.

Por ejemplo, para declarar e inicializar un arreglo de enteros con 5 elementos:

```
int numeros[5] = {1, 2, 3, 4, 5};
```

2. Inicialización después de la declaración:
También puedes inicializar un arreglo después de haberlo declarado utilizando un bucle `for` o asignando valores individualmente a cada elemento del arreglo. Aquí hay un ejemplo utilizando un bucle `for`:

```
int numeros[5]; // Declaración del arreglo
int i;

// Inicialización después de la declaración
for (i = 0; i < 5; i++) {
    numeros[i] = i + 1;
}
```

En este ejemplo, estamos inicializando el arreglo `numeros` con valores del 1 al 5.
Recuerda que en ambos casos, el número de valores proporcionados durante la inicialización no debe exceder el tamaño del arreglo. De lo contrario, habrá un error de compilación.

4 Acceso a Elementos de un Arreglo

Para acceder a elementos individuales de un arreglo en C, utilizamos la notación de corchetes `[]` con el índice del elemento que queremos acceder. Aquí tienes una explicación detallada:
Supongamos que tenemos un arreglo de enteros llamado `numeros`:

```
int numeros[5] = {10, 20, 30, 40, 50};
```

Para acceder a un elemento específico del arreglo, simplemente indicamos su posición o índice entre los corchetes. Por ejemplo:

```
int primer_numero = numeros[0]; % Accediendo al primer elemento
int tercer_numero = numeros[2]; % Accediendo al tercer elemento
```

En este caso, `numeros[0]` nos dará el valor 10 (el primer elemento del arreglo), y `numeros[2]` nos dará el valor 30 (el tercer elemento del arreglo).
Recuerda que en C, los índices de los arreglos comienzan en 0. Por lo tanto, `numeros[0]` es el primer elemento, `numeros[1]` es el segundo elemento, y así sucesivamente.
También puedes usar variables para acceder a los elementos del arreglo:

```
int i = 3;
int cuarto_numero = numeros[i]; % Accediendo al cuarto elemento usando una variable
```

En este caso, `numeros[i]` nos dará el valor 40, ya que `i` es igual a 3 y estamos accediendo al cuarto elemento del arreglo.
Recuerda que es importante tener cuidado al acceder a los elementos de un arreglo para evitar desbordamientos de índices, lo que podría causar comportamientos inesperados o errores en tu programa. Es decir, si intentas acceder a un índice que está fuera del rango del arreglo, el comportamiento no está definido y podría causar fallos en el programa o acceder a datos no deseados.

4.1 Acceso a todos los elementos mediante un ciclo

En C, puedes usar un bucle, como un bucle `for` o un bucle `while`, para acceder a todos los elementos de un arreglo. Aquí te muestro cómo hacerlo con un bucle `for`:

```
#include <stdio.h>

int main() {
    int numeros[5] = {10, 20, 30, 40, 50};
    int i;

    // Usando un bucle for para acceder a todos los elementos del arreglo
    for (i = 0; i < 5; i++) {
        printf("Elemento en la posición %d: %d\n", i, numeros[i]);
    }

    return 0;
}
```

Este programa imprimirá cada elemento del arreglo `numeros` junto con su índice. El bucle `for` itera sobre cada posición del arreglo, comenzando desde 0 hasta el tamaño del arreglo menos 1, e imprime el elemento en esa posición en cada iteración.

También puedes utilizar un bucle `while` para lograr lo mismo:

```
#include <stdio.h>

int main() {
    int numeros[5] = {10, 20, 30, 40, 50};
    int i = 0;

    // Usando un bucle while para acceder a todos los elementos del arreglo
    while (i < 5) {
        printf("Elemento en la posición %d: %d\n", i, numeros[i]);
        i++;
    }

    return 0;
}
```

Este programa produce la misma salida que el anterior, pero utiliza un bucle `while` en lugar de un bucle `for`.

5 Operaciones con Arreglos

5.1 Busqueda de un elemento

En C, puedes usar un bucle `while` para buscar un elemento específico en un arreglo. De esta manera:

```
#include <stdio.h>

int main() {
    int numeros[5] = {10, 20, 30, 40, 50};
    int elemento_a_buscar = 30;
    int encontrado = 0; % Variable para indicar si se encontró el elemento
    int i = 0;

    % Usando un bucle while para buscar el elemento en el arreglo
    while (i < 5 && encontrado == 0) {
        if (numeros[i] == elemento_a_buscar) {
            printf("El elemento %d se encuentra en la posición %d.\n", elemento_a_buscar, i);
            encontrado = 1; % Cambiar el valor de encontrado a 1 para salir del bucle
        }
        i++;
    }

    % Verificar si el elemento no se encontró
    if (!encontrado) {
        printf("El elemento %d no se encuentra en el arreglo.\n", elemento_a_buscar);
    }

    return 0;
}
```

En este ejemplo, estamos buscando el elemento `30` en el arreglo `numeros`. Utilizamos un bucle `while` para iterar sobre cada elemento del arreglo. En cada iteración, verificamos si el elemento en la posición actual (`numeros[i]`) es igual al elemento que estamos buscando (`elemento_a_buscar`). Si encontramos el elemento, imprimimos su posición y cambiamos el valor de la variable `encontrado` a `1` para salir del bucle. Si después de terminar el bucle `while`, la variable `encontrado` sigue siendo `0`, significa que el elemento no se encontró en el arreglo, por lo que imprimimos un mensaje indicando que el elemento no se encontró.

Este es solo un ejemplo básico de cómo buscar un elemento en un arreglo usando un bucle `while`. Dependiendo de tus necesidades específicas, puedes modificar el código para que se adapte mejor a tu situación.

5.2 Ordenamiento de un arreglo

En este ejemplo, se presenta un programa en C que ordena un arreglo de números utilizando el algoritmo de ordenamiento burbuja, implementado únicamente con bucles `while`.

```
#include <stdio.h>

void bubbleSort(int arreglo[], int n) {
    int i = 0, temp;
    int ordenado = 0; % Variable para indicar si el arreglo está ordenado

    while (!ordenado) {
        ordenado = 1; % Suponemos que el arreglo está ordenado
        i = 0;
        while (i < n - 1) {
            if (arreglo[i] > arreglo[i + 1]) {
                % Intercambiar los elementos si están en el orden incorrecto
                temp = arreglo[i];
                arreglo[i] = arreglo[i + 1];
                arreglo[i + 1] = temp;
                ordenado = 0; % El arreglo aún no está ordenado
            }
            i++;
        }
    }
}

int main() {
    int numeros[] = {5, 2, 7, 1, 9};
    int n = sizeof(numeros) / sizeof(numeros[0]);
    int i = 0;

    % Imprimir el arreglo antes de ordenarlo
    printf("Arreglo antes de ordenarlo: ");
    while (i < n) {
        printf("%d ", numeros[i]);
        i++;
    }
    printf("\n");

    % Ordenar el arreglo utilizando bubbleSort
    bubbleSort(numeros, n);

    % Reiniciar el índice
    i = 0;

    % Imprimir el arreglo después de ordenarlo
    printf("Arreglo después de ordenarlo: ");
    while (i < n) {
        printf("%d ", numeros[i]);
        i++;
    }
    printf("\n");

    return 0;
}
```

Este código realiza el ordenamiento del arreglo utilizando un bucle `while` tanto para la ordenación como para imprimir los elementos antes y después del ordenamiento.

5.3 Modificación de un arreglo

En C, existen varias formas comunes de modificar un arreglo. A continuación, se presentan algunos ejemplos utilizando un bucle `while` en lugar de un bucle `for`:

1. **Asignación directa:**

```
int arreglo[5] = {1, 2, 3, 4, 5};
arreglo[2] = 10; // Modificar el tercer elemento del arreglo
```

   Esto cambia el valor del tercer elemento del arreglo `arreglo` de 3 a 10.

2. **Utilizando un bucle:**

```
int arreglo[5] = {1, 2, 3, 4, 5};
int i = 0;
while (i < 5) {
    arreglo[i] *= 2; % Multiplicar cada elemento por 2
    i++;
}
```

   Esto multiplica cada elemento del arreglo `arreglo` por 2.

3. **Copiando otro arreglo:**

```
int arreglo1[5] = {1, 2, 3, 4, 5};
int arreglo2[5] = {10, 20, 30, 40, 50};
int i = 0;
while (i < 5) {
    arreglo1[i] = arreglo2[i]; % Copiar elementos de arreglo2 a arreglo1
    i++;
}
```

   Esto copia los elementos del arreglo `arreglo2` al arreglo `arreglo1`.

4. **Utilizando funciones:**

   Puedes escribir funciones personalizadas para modificar arreglos según tus necesidades específicas. Las funciones pueden usar bucles `while` para realizar las operaciones necesarias dentro del arreglo.

6 Pedir y mostrar un arreglo

Para pedir al usuario que ingrese los datos de un arreglo paso a paso utilizando un bucle `while` en C, puedes seguir estos pasos:

1. Declara un arreglo con el tamaño deseado.
2. Utiliza un bucle `while` para solicitar al usuario que ingrese los elementos del arreglo uno por uno.
3. Dentro del bucle `while`, solicita al usuario que ingrese un elemento y almacena este valor en la posición actual del arreglo.
4. Incrementa el contador que indica la posición actual del arreglo.
5. Termina el bucle cuando se han ingresado todos los elementos.

Aquí tienes un ejemplo de cómo implementarlo en C:

```
#include <stdio.h>

int main() {
    int tamano;
    printf("Ingrese el tamaño del arreglo: ");
    scanf("%d", &tamano);

    int arreglo[tamano];
    int i = 0;

    printf("Ingrese los elementos del arreglo uno por uno:\n");
    while (i < tamano) {
        printf("Elemento %d: ", i + 1);
        scanf("%d", &arreglo[i]);
        i++;
    }

    // Imprimir el arreglo ingresado por el usuario
    printf("El arreglo ingresado es: ");
    i = 0;
    while (i < tamano) {
        printf("%d ", arreglo[i]);
        i++;
    }
    printf("\n");

    return 0;
}
```

En este código, primero solicitamos al usuario que ingrese el tamaño del arreglo. Luego, creamos un arreglo de ese tamaño. Después, utilizando un bucle `while`, solicitamos al usuario que ingrese cada elemento del arreglo uno por uno. Finalmente, imprimimos el arreglo ingresado por el usuario para verificar que los datos se almacenaron correctamente.

7 Solución de Ejercicios

7.1 Estructura básica

1. **Incluir bibliotecas**: Comienza incluyendo las bibliotecas necesarias para tu programa. Por lo general, necesitarás `#include <stdio.h>` para entrada y salida estándar, y otras bibliotecas según tus necesidades específicas.

2. **Declaraciones de funciones**: Después de las inclusiones de bibliotecas, puedes declarar las funciones que planeas utilizar en tu programa. Esto podría incluir funciones para inicializar arreglos, realizar operaciones en los arreglos, etc.

3. **Función principal (`main`)**: Luego, define la función `main()`. Esta es la función principal que será ejecutada cuando se inicie tu programa. Aquí es donde comenzará la ejecución del programa.

4. **Implementación de funciones**: Finalmente, después de la función `main()`, implementa las funciones que has declarado anteriormente. Estas funciones pueden ser para realizar operaciones específicas en los arreglos, como búsqueda, ordenación, etc.

7.1.1 Ejemplo:

```
#include <stdio.h>

// Declaraciones de funciones
void imprimirArreglo(int arreglo[], int longitud);

// Función principal (main)
int main() {
    int miArreglo[] = {1, 2, 3, 4, 5};
    int longitud = sizeof(miArreglo) / sizeof(miArreglo[0]);

    printf("Contenido del arreglo:\n");
    imprimirArreglo(miArreglo, longitud);

    return 0;
}

// Implementación de funciones
void imprimirArreglo(int arreglo[], int longitud) {
    for (int i = 0; i < longitud; i++) {
        printf("%d ", arreglo[i]);
    }
    printf("\n");
}
```

7.2 Variables o constantes

Los problemas suelen requerir que declares variables o constantes para almacenar datos específicos.

7.2.1 Constantes:

- Una constante es un valor que no cambia durante la ejecución del programa.
- En C, las constantes pueden ser definidas utilizando la directiva `#define` o mediante el uso de la palabra clave `const`.
- Cuando se define una constante con `#define`, el preprocesador de C realiza un reemplazo directo del nombre de la constante por su valor en todo el código fuente.
- Cuando se utiliza `const`, se reserva espacio en memoria para la constante, y su valor no puede ser modificado durante la ejecución del programa.

Ejemplo con `#define`:

```
#define PI 3.14159
```

Ejemplo con `const`:

```
const int MAX_VAL = 100;
```

7.2.2 Variables:

- Una variable es un espacio en memoria que se reserva para almacenar datos que pueden cambiar durante la ejecución del programa.
- En C, las variables se declaran especificando su tipo de dato y su nombre.

- El valor de una variable puede ser modificado durante la ejecución del programa.
- Las variables deben ser inicializadas antes de ser utilizadas, ya sea con un valor específico o mediante una operación de asignación.

Ejemplo:

```
int edad;
edad = 25; // Asignando un valor a la variable 'edad'
```

8 Ejemplos de Arreglos en C

8.1 Arreglos entrada-salida (Ejercicio 6 - Proyecto 4)

Escribir un programa que solicite el ingreso de un arreglo de enteros `int a[]` y luego imprime por pantalla. El programa debe utilizar dos nuevas funciones además de la función main:

- una que dado un tamaño máximo de arreglo y el arreglo, solicita los valores para el arreglo y los devuelve, guardándolos en el mismo arreglo `int a[]`; función con prototipo (también conocido como signatura o firma):

```
void pedir_arreglo(int n_max, int a[])
```

- otra que imprime cada uno de los valores del arreglo `int a[]`, de prototipo:

```
void imprimir_arreglo(int n_max, int a[])
```

Primero vamos a estructurar el programa de la siguiente forma:

1. Inclusión de librerías

```
#include <stdio.h>
```

2. Initialización de funciones

```
void pedir_arreglo(int n_max, int a[]);
void imprimir_arreglo(int n_max, int a[]);
```

3. Función main

```
int main() {
    int n_max = 5; // Tamano del arreglo
    int a[n_max]; // Declaracion del arreglo
    pedir_arreglo(n_max, a); // Llamada a la funcion para pedir el arreglo
    imprimir_arreglo(n_max, a); // Llamada a la funcion para imprimir el arreglo
    return 0; // Fin del programa
}
```

4. Implementación de funciones

- Función `pedir_arreglo`:

```
void pedir_arreglo(int n_max, int a[]) {
    int i = 0;
    while (i < n_max) {
        printf("Ingrese el valor %d del arreglo: ", i + 1);
        scanf("%d", &a[i]);
        i++;
    }
}
```

- Función `imprimir_arreglo`:

```
void imprimir_arreglo(int n_max, int a[]) {
    int i = 0;
    printf("El arreglo ingresado es: ");
    while (i < n_max) {
        printf("%d ", a[i]);
        i++;
    }
    printf("\n");
}
```

El código completo del programa es el siguiente:

```
#include <stdio.h>

void pedir_arreglo(int n_max, int a[]);
void imprimir_arreglo(int n_max, int a[]);

int main() {
    int n_max = 5; // Tamano del arreglo
    int a[n_max]; // Declaracion del arreglo
    pedir_arreglo(n_max, a); // Llamada a la funcion para pedir el arreglo
    imprimir_arreglo(n_max, a); // Llamada a la funcion para imprimir el arreglo
    return 0; // Fin del programa
}

void pedir_arreglo(int n_max, int a[]) {
    int i = 0; // Inicializacion de contador
    while (i < n_max) {
        printf("Ingrese el valor %d del arreglo: ", i + 1); // Solicitar el valor del arreglo
        scanf("%d", &a[i]); // Guardar el valor en el arreglo
        i++;
    }
}

void imprimir_arreglo(int n_max, int a[]) {
    int i = 0; // Inicializacion de contador
    printf("El arreglo ingresado es: "); // Imprimir mensaje
    while (i < n_max) {
        printf("%d ", a[i]); // Imprimir cada valor del arreglo
        i++; // Incrementar el contador
    }
    printf("\n"); // Imprimir salto de linea
}
```

8.2 Función Sumatoria (Ejercicio 7 - Proyecto 4)

Hacer un programa en un archivo con nombre `sumatoria.c` que contenga la función

```
int sumatoria(int tam, int a[])
```

que recibe un tamaño máximo de arreglo y un arreglo como argumento, y devuelve la suma de sus elementos (del arreglo). En la función main pedir los datos del arreglo al usuario asumiendo un tamaño constante previamente establecido (en tiempo de compilacilación).

Primero observamos que el tamaño del arreglo es constante, por lo que podemos definirlo como una constante en el programa. Luego, los datos del arreglo se piden dentro de la función main sin necesidad de usar funciones auxiliares para ello.

```
#include <stdio.h>

#define TAM 5

int sumatoria(int tam, int a[]);

int main() {
    int a[TAM]; // Declaracion del arreglo
    int i = 0; // Inicializacion de contador
    while (i < TAM) {
        printf("Ingrese el valor %d del arreglo: ", i + 1);
        scanf("%d", &a[i]);
        i++;
    }
    printf("La sumatoria del arreglo es: %d\n", sumatoria(TAM, a)); // Llamada a la funcion sumatoria
    return 0;
}

int sumatoria(int tam, int a[]) {
    int i = 0; // Inicializacion de contador
    int suma = 0; // Inicializacion de la variable suma
    while (i < tam) {
        suma += a[i]; // Sumar el valor actual al acumulado
        i++; // Incrementar el contador
    }
    return suma; // Devolver el valor de la suma
}
```

La función `sumatoria` en C calcula la suma de todos los elementos de un arreglo de enteros `a[]` de tamaño `tam`. Aquí está el desglose de la función:

- `int sumatoria(int tam, int a[])`: Esto define una función llamada `sumatoria` que toma dos argumentos: `tam`, que indica el tamaño del arreglo, y `a[]`, que es el arreglo de enteros.
- `int i = 0;`: Aquí se inicializa una variable `i` que actuará como contador para recorrer el arreglo.
- `int suma = 0;`: Se inicializa una variable `suma` que almacenará el resultado de la suma de todos los elementos del arreglo.
- `while (i < tam) { ... }`: Se utiliza un bucle `while` para recorrer el arreglo desde `a[0]` hasta `a[tam-1]`.
- `suma += a[i];`: En cada iteración del bucle, se suma el valor del elemento actual del arreglo `a[]` a la variable `suma`.

- `i++;`: Se incrementa el contador `i` para pasar al siguiente elemento del arreglo en la próxima iteración.
- `return suma;`: Una vez que se han sumado todos los elementos del arreglo, la función devuelve el valor total de la suma almacenado en la variable `suma`.

8.3 Función Múltiplos (Ejercicio 8 - Proyecto 4)

Hacer un programa en un archivo `multiplos.c` que contenga las siguientes funciones:

```
bool todos_pares(int tam, int a[])
bool existe_multiplo(int m, int tam, int a[])
```

La primera recibe un tamaño máximo de arreglo y un arreglo como argumento devolviendo `true` cuando todos los elementos del arreglo `a[]` son numeros pares y false en caso contrario. La segunda determina si hay en el arreglo `a[]` algún elemento que es múltiplo de m. En la función main se debe pedir al usuario los elementos del arreglo (asumiendo un tamaño constante) y luego permitirle elegir qué función ejecutar. En caso que se elija la función `existe_multiplo()` se le debe pedir al usuario un valor para m.

Primero observamos que el tamaño del arreglo es constante, por lo que podemos definirlo como una constante en el programa. Luego, los datos del arreglo se piden dentro de la función main sin necesidad de usar funciones auxiliares para ello. La estructura del programa será la siguiente:

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

#define TAM 5

bool todos_pares(int tam, int a[]);
bool existe_multiplo(int m, int tam, int a[]);

int main() {
    ...
    return 0;
}

bool todos_pares(int tam, int a[]) {
    ...
}

bool existe_multiplo(int m, int tam, int a[]) {
    ...
}
```

Por lo pronto sabemos que para crear la funcion `todos_pares`, vamos a tener que recorrer el arreglo con un ciclo, y verificar si cada elemento es par. La lógica que utilizaremos en estos casos es la siguiente, dentro del ciclo `while` va a haber un `if` que verifica si el elemento actual es par, si no lo es, la función devuelve `false`, si el ciclo termina, la función devuelve `true`.

```
bool todos_pares(int tam, int a[]) {
    int i = 0; // Inicializacion de contador
    while (i < tam) {
        if (a[i] % 2 != 0) { // Verificar si el elemento actual es par
            return false; // Devolver false si el elemento no es par
        }
        i++; // Incrementar el contador
    }
    return true; // Devolver true si todos los elementos son pares
}
```

Luego, para la función `existe_multiplo` vamos a tener que recorrer el arreglo con un ciclo, y verificar si existe al menos un elemento que sea múltiplo de `m`. La lógica es similar al anterior, dentro del ciclo `while` va a haber un `if` que verifica si algun elemento es múltiplo de `m`, si lo es, la función devuelve `true`, si el ciclo termina, la función devuelve `false` y esto significa que no existe ningún múltiplo de `m`, nunca entra en el condicional.

```
bool existe_multiplo(int m, int tam, int a[]) {
    int i = 0; // Inicializacion de contador
    while (i < tam) {
        if (a[i] % m == 0) { // Verificar si el elemento actual es multiplo de m
            return true; // Devolver true si el elemento es multiplo de m
        }
        i++; // Incrementar el contador
    }
    return false; // Devolver false si no hay ningun multiplo de m
}
```

Luego en la función main hay que pedir los datos del arreglo al usuario, y luego permitirle elegir qué función ejecutar. En caso que se elija la función `existe_multiplo()` se le debe pedir al usuario un valor para `m`. Para esto se puede utilizar un condicional para que el usuario pueda elegir entre las dos opciones:

```
int main() {
    int a[TAM]; // Declaracion del arreglo
    int i = 0; // Inicializacion de contador
    while (i < TAM) {
        printf("Ingrese el valor %d del arreglo: ", i + 1);
        scanf("%d", &a[i]);
        i++;
    }
    int opcion;
    printf("Elija una opcion:\n1. Verificar si todos los elementos son pares\n2. Verificar si
        existe un multiplo de m\n");
    scanf("%d", &opcion);
    if (opcion == 1) {
        if (todos_pares(TAM, a)) {
            printf("Todos los elementos del arreglo son pares.\n");
        } else {
            printf("No todos los elementos del arreglo son pares.\n");
        }
    } else if (opcion == 2) {
        int m;
        printf("Ingrese el valor de m: ");
        scanf("%d", &m);
        if (existe_multiplo(m, TAM, a)) {
            printf("Existe al menos un multiplo de %d en el arreglo.\n", m);
        } else {
            printf("No existe ningun multiplo de %d en el arreglo.\n", m);
        }
    }
    return 0;
}
```

El código completo del programa es el siguiente:

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

#define TAM 5

bool todos_pares(int tam, int a[]);
bool existe_multiplo(int m, int tam, int a[]);

int main() {
    int a[TAM]; // Declaracion del arreglo
    int i = 0; // Inicializacion de contador
    while (i < TAM) {
        printf("Ingrese el valor %d del arreglo: ", i + 1);
        scanf("%d", &a[i]);
        i++;
    }
    int opcion;
    printf("Elija una opcion:\n1. Verificar si todos los elementos son pares\n2. Verificar si
        existe un multiplo de m\n");
    scanf("%d", &opcion);
    if (opcion == 1) {
        if (todos_pares(TAM, a)) {
            printf("Todos los elementos del arreglo son pares.\n");
        } else {
            printf("No todos los elementos del arreglo son pares.\n");
        }
    } else if (opcion == 2) {
        int m;
        printf("Ingrese el valor de m: ");
        scanf("%d", &m);
        if (existe_multiplo(m, TAM, a)) {
            printf("Existe al menos un multiplo de %d en el arreglo.\n", m);
        } else {
            printf("No existe ningun multiplo de %d en el arreglo.\n", m);
        }
    }
    return 0;
}

bool todos_pares(int tam, int a[]) {
    int i = 0; // Inicializacion de contador
    while (i < tam) {
        if (a[i] % 2 != 0) { // Verificar si el elemento actual es par
            return false; // Devolver false si el elemento no es par
        }
        i++; // Incrementar el contador
    }
    return true; // Devolver true si todos los elementos son pares
}

bool existe_multiplo(int m, int tam, int a[]) {
    int i = 0; // Inicializacion de contador
    while (i < tam) {
        if (a[i] % m == 0) { // Verificar si el elemento actual es multiplo de m
```

```
            return true; // Devolver true si el elemento es multiplo de m
        }
        i++; // Incrementar el contador
    }
    return false; // Devolver false si no hay ningun multiplo de m
}
```

8.4 Procedimiento intercambio (Ejercicio 9 - Proyecto 4)

Hacer un programa en el archivo nuevo `intercambio_arreglos.c` que contenga la siguiente función:

```
void intercambiar(int tam, int a[], int i, int j)
```

que recibe un tamaño máximo de arreglo, un arreglo y dos posiciones como argumento, e intercambia los elementos del arreglo en dichas posiciones. En la función main pedirle al usuario que ingrese los elementos del arreglo y las posiciones, chequear que las posiciones estén en el rango correcto y luego imprimir en pantalla el arreglo modificado. La estructura del programa será la siguiente:

```
#include <stdio.h>

#define TAM 5

void intercambiar(int tam, int a[], int i, int j);

int main() {
    ...
    return 0;
}

void intercambiar(int tam, int a[], int i, int j) {
    ...
}
```

Para la función `intercambiar` se puede utilizar una variable auxiliar para guardar el valor de `a[i]` antes de modificarlo, luego se asigna el valor de `a[j]` a `a[i]` y finalmente se asigna el valor de la variable auxiliar a `a[j]`.

```
void intercambiar(int tam, int a[], int i, int j) {
    if (i >= 0 && i < tam && j >= 0 && j < tam) { // Verificar que las posiciones esten
    en el rango correcto
        int temp = a[i]; // Guardar el valor de a[i] en una variable auxiliar
        a[i] = a[j]; // Asignar el valor de a[j] a a[i]
        a[j] = temp; // Asignar el valor de la variable auxiliar a a[j]
    }
}
```

Luego en la función `main`, vamos a pedirle al usuario que ingrese el arreglo utilizando un ciclo y luego las posiciones `i` y `j` para intercambiar los elementos del arreglo. Finalmente, se imprime el arreglo modificado.

```
int main() {
    int a[TAM]; // Declaracion del arreglo
    int i = 0; // Inicializacion de contador
    while (i < TAM) {
        printf("Ingrese el valor %d del arreglo: ", i + 1);
        scanf("%d", &a[i]);
        i++;
    }
    int pos1, pos2;
    printf("Ingrese la primera posicion: ");
    scanf("%d", &pos1);
    printf("Ingrese la segunda posicion: ");
    scanf("%d", &pos2);
    intercambiar(TAM, a, pos1, pos2); // Llamada a la funcion intercambiar
    printf("El arreglo modificado es: ");
    i = 0; // Reiniciar el contador
    while (i < TAM) {
        printf("%d ", a[i]); // Imprimir cada valor del arreglo
        i++; // Incrementar el contador
    }
    printf("\n"); // Imprimir salto de linea
    return 0;
}
```

El código completo del programa es el siguiente:

```
#include <stdio.h>

#define TAM 5

void intercambiar(int tam, int a[], int i, int j);

int main() {
    int a[TAM]; // Declaracion del arreglo
    int i = 0; // Inicializacion de contador
    while (i < TAM) {
        printf("Ingrese el valor %d del arreglo: ", i + 1);
        scanf("%d", &a[i]);
        i++;
    }
    int pos1, pos2;
    printf("Ingrese la primera posicion: ");
    scanf("%d", &pos1);
    printf("Ingrese la segunda posicion: ");
    scanf("%d", &pos2);
    intercambiar(TAM, a, pos1, pos2); // Llamada a la funcion intercambiar
    printf("El arreglo modificado es: ");
    i = 0; // Reiniciar el contador
    while (i < TAM) {
        printf("%d ", a[i]); // Imprimir cada valor del arreglo
        i++; // Incrementar el contador
    }
    printf("\n"); // Imprimir salto de linea
    return 0;
}
```

```
void intercambiar(int tam, int a[], int i, int j) {
    if (i >= 0 && i < tam && j >= 0 && j < tam) { // Verificar que las posiciones esten
    en el rango correcto
        int temp = a[i]; // Guardar el valor de a[i] en una variable auxiliar
        a[i] = a[j]; // Asignar el valor de a[j] a a[i]
        a[j] = temp; // Asignar el valor de la variable auxiliar a a[j]
    }
}
```

9 Arreglos con tipos de datos compuestos

9.1 Declaración de la estructura (struct)

Primero, defines la estructura utilizando la palabra clave `struct`. Por ejemplo:

```
struct Persona {
    char nombre[50];
    int edad;
};
```

Esto define una estructura llamada `Persona` que contiene dos miembros: `nombre`, que es un arreglo de caracteres de tamaño 50, y `edad`, que es un entero.

9.2 Declaración del arreglo de estructuras

Después de definir la estructura, puedes declarar un arreglo de estructuras de ese tipo. Por ejemplo:

```
struct Persona personas[10];
```

Esto declara un arreglo llamado `personas` que puede contener hasta 10 estructuras del tipo `Persona`.

9.3 Acceso a los elementos del arreglo de estructuras

Puedes acceder a los elementos del arreglo de estructuras utilizando el operador de acceso a miembros (`.`). Por ejemplo:

```
int i = 0;
while (i < 10) {
    personas[i].nombre[0] = 'J';
    personas[i].nombre[1] = 'u';
    personas[i].nombre[2] = 'a';
    personas[i].nombre[3] = 'n';
    personas[i].nombre[4] = '\0'; // Asegurando que el nombre esté terminado con '\0'
    personas[i].edad = 30;
    i++;
}
```

Esto asigna el nombre "Juan" y la edad 30 a la primera estructura en el arreglo `personas`, sin usar la función `strcpy`.

Iteración sobre el arreglo de estructuras

Puedes iterar sobre el arreglo de estructuras de la misma manera que lo harías con un arreglo de cualquier otro tipo de datos. Por ejemplo:

```
int i = 0;
while (i < 10) {
    printf("Nombre: %s, Edad: %d\n", personas[i].nombre, personas[i].edad);
    i++;
}
```

Esto imprimirá el nombre y la edad de cada persona en el arreglo `personas` utilizando un bucle `while`.

9.4 Función strcpy - Asignación de Cadenas a Arreglos de Caracteres en C

En C, no puedes asignar directamente una cadena de caracteres a un arreglo de caracteres usando el operador de asignación `=`. Sin embargo, puedes copiar una cadena de caracteres en un arreglo de caracteres utilizando la función `strcpy()` de la biblioteca `string.h`.

9.4.1 Ejemplo:

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

struct Persona {
    char nombre[50];
    int edad;
};

int main() {
    struct Persona personas[10];

    int i = 0;
    while (i < 10) {
        strcpy(personas[i].nombre, "Juan");
        personas[i].edad = 30;
        i++;
    }

    // Imprimir los nombres y edades
    i = 0;
    while (i < 10) {
        printf("Nombre: %s, Edad: %d\n", personas[i].nombre, personas[i].edad);
        i++;
    }

    return 0;
}
```

En este ejemplo, `strcpy(personas[i].nombre, "Juan")` copia la cadena "Juan" en el arreglo de caracteres `nombre` en la estructura `Persona` en la posición `i`.

10 Ejemplos de Arreglos de Estructuras en C

10.1 Arreglo de asociaciones (Ejercicio 14 - Proyecto 4)

En `asoc.c` programar la función

```
bool asoc_existe(int tam, struct asoc a[], clave_t c)
```

Donde la estructura struct asoc y los tipos `clave_t`, `valor_t` se definen como:

```
typedef char clave_t;
typedef int valor_t;
struct asoc {
        clave_t clave
        valor_t valor
};
```

El llamado a asoc_existe(tam, a, c) debe indicar si la clave c se encuentra en el arreglo de asociaciones a[]. En la función main pedir al usuario los datos del arreglo (asumiendo un tamaño constante) y luego pedir una clave. Finalmente usar la función asoc_existe para verificar la existencia de la clave ingresada y mostrar por pantalla un mensaje indicando si la clave existe o no en el arreglo de asociaciones.

- Se asume un tamaño constante para el arreglo de asociaciones, por lo que se puede definir como una constante en el programa.
- Los datos del arreglo se piden dentro de la función main sin necesidad de usar funciones auxiliares para ello.
- Para verificar si la clave ingresada existe en el arreglo de asociaciones, se puede utilizar un ciclo para recorrer el arreglo y comparar cada clave con la clave ingresada.

La estructura del programa será la siguiente:

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

#define TAM 5

typedef char clave_t;
typedef int valor_t;
struct asoc {
    clave_t clave;
    valor_t valor;
};

bool asoc_existe(int tam, struct asoc a[], clave_t c);

int main() {
    ...
    return 0;
}

bool asoc_existe(int tam, struct asoc a[], clave_t c) {
    ...
}
```

Para la función `asoc_existe` se puede utilizar un ciclo para recorrer el arreglo de asociaciones y comparar cada clave con la clave ingresada. Si se encuentra una coincidencia, la función devuelve `true`, de lo contrario, devuelve `false`.

```
bool asoc_existe(int tam, struct asoc a[], clave_t c) {
    int i = 0; // Inicializacion de contador
    while (i < tam) {
        if (a[i].clave == c) { // Verificar si la clave actual es igual a la clave ingresada
            return true; // Devolver true si la clave existe
        }
        i++; // Incrementar el contador
    }
    return false; // Devolver false si la clave no existe
}
```

Luego en la función `main`, vamos a pedirle al usuario que ingrese el arreglo de asociaciones utilizando un ciclo y luego la clave para verificar si existe en el arreglo. Finalmente, se imprime un mensaje indicando si la clave existe o no.

```
int main() {
    struct asoc a[TAM]; // Declaracion del arreglo de asociaciones
    int i = 0; // Inicializacion de contador
    while (i < TAM) {
        printf("Ingrese la clave %d: ", i + 1);
        scanf(" %c", &a[i].clave);
        printf("Ingrese el valor %d: ", i + 1);
        scanf("%d", &a[i].valor);
        i++;
    }
    clave_t c;
    printf("Ingrese la clave a buscar: ");
    scanf(" %c", &c);
    if (asoc_existe(TAM, a, c)) {
        printf("La clave %c existe en el arreglo de asociaciones.\n", c);
    } else {
        printf("La clave %c no existe en el arreglo de asociaciones.\n", c);
    }
    return 0;
}
```

El código completo del programa es el siguiente:

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

#define TAM 5

typedef char clave_t;
typedef int valor_t;
struct asoc {
    clave_t clave;
    valor_t valor;
};

bool asoc_existe(int tam, struct asoc a[], clave_t c);

int main() {
    struct asoc a[TAM]; // Declaracion del arreglo de asociaciones
    int i = 0; // Inicializacion de contador
    while (i < TAM) {
        printf("Ingrese la clave %d: ", i + 1);
        scanf(" %c", &a[i].clave);
        printf("Ingrese el valor %d: ", i + 1);
        scanf("%d", &a[i].valor);
        i++;
    }
    clave_t c;
    printf("Ingrese la clave a buscar: ");
    scanf(" %c", &c);
    if (asoc_existe(TAM, a, c)) {
        printf("La clave %c existe en el arreglo de asociaciones.\n", c);
    } else {
        printf("La clave %c no existe en el arreglo de asociaciones.\n", c);
    }
    return 0;
}

bool asoc_existe(int tam, struct asoc a[], clave_t c) {
    int i = 0; // Inicializacion de contador
    while (i < tam) {
        if (a[i].clave == c) { // Verificar si la clave actual es igual a la clave ingresada
            return true; // Devolver true si la clave existe
        }
        i++; // Incrementar el contador
    }
    return false; // Devolver false si la clave no existe
}
```