Pedro Villar

Notas de Clase
Algoritmos y Estructuras de Datos 2
Laboratorio

Primer Cuatrimestre 2024

Índice de Contenido

1 Repaso de Arreglos

1.1 Arreglos multidimensionales

Los arreglos multidimensionales en C son estructuras de datos que permiten almacenar elementos en más de una dimensión. Por ejemplo, un arreglo unidimensional es similar a una lista, mientras que un arreglo bidimensional es similar a una tabla o una matriz.

1.1.1 Declaración

1. **Declaración directa:**

```
int matriz[3][3];
```

Esto declara una matriz de enteros de 3x3. Los elementos de la matriz no se inicializan y contendrán valores basura hasta que se les asigne un valor explícitamente.

2. **Inicialización explícita:**

```
int matriz[3][3] = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};
```

Aquí se declara e inicializa una matriz de 3x3 con valores específicos.

3. **Usando typedef:**

```
typedef int ArregloBidimensional[3][3];
ArregloBidimensional matriz;
```

Esto declara un tipo ‘ArregloBidimensional‘, que luego se usa para declarar una matriz de 3x3 llamada ‘matriz‘.

4. **Arreglo de punteros:**

```
int *matriz[3];
for (int i = 0; i < 3; i++) {
    matriz[i] = malloc(3 * sizeof(int));
}
```

En este caso, se declara un arreglo de punteros a enteros. Luego, en un bucle, se asigna memoria dinámica para cada fila de la matriz.

5. **Usando punteros dobles:**

```
int **matriz;
matriz = malloc(3 * sizeof(int *));
for (int i = 0; i < 3; i++) {
    matriz[i] = malloc(3 * sizeof(int));
}
```

Aquí, ‘matriz‘ es un puntero doble que apunta a un bloque de memoria que contiene punteros a enteros. Luego, en un bucle, se asigna memoria dinámica para cada fila de la matriz.

Es importante tener en cuenta que las diferentes formas de declarar arreglos multidimensionales tienen implicaciones diferentes, especialmente en lo que respecta a la asignación de memoria y la gestión de punteros. Selecciona el método que mejor se adapte a tus necesidades y ten en cuenta la gestión adecuada de la memoria para evitar fugas de memoria y comportamientos inesperados.

1.2 Acceso a elementos

Por ejemplo, un arreglo bidimensional se puede visualizar como una tabla con filas y columnas. Para acceder a un elemento específico en un arreglo multidimensional, necesitas especificar los índices correspondientes a cada dimensión del arreglo.

```
#include <stdio.h>

int main() {
    // Declaración e inicialización de un arreglo bidimensional 2x3
    int arreglo[2][3] = {
        {1, 2, 3},
        {4, 5, 6}
    };

    // Acceso a elementos individuales
    printf("Elemento en la fila 0, columna 0: %d\n", arreglo[0][0]);
    printf("Elemento en la fila 1, columna 2: %d\n", arreglo[1][2]);

    return 0;
}
```

En este ejemplo, `arreglo` es un arreglo bidimensional de tamaño 2x3. Para acceder a un elemento específico, utilizamos la sintaxis `arreglo[i][j]`, donde `i` representa el índice de la fila y `j` representa el índice de la columna. Recuerda que en C, los índices comienzan desde cero, por lo que el primer elemento de la matriz tiene índices `[0][0]`.

El mismo concepto se aplica a arreglos multidimensionales con más de dos dimensiones. Por ejemplo, para un arreglo tridimensional, necesitarías especificar tres índices: `arreglo[i][j][k]`, donde `i`, `j` y `k` representan las coordenadas en cada dimensión respectivamente.

1.3 Paso de arreglos multidimensionales a funciones

Cuando pasas un arreglo multidimensional a una función en C, debes tener en cuenta que la función no recibe un puntero a un arreglo multidimensional, sino un puntero a un arreglo unidimensional. Esto se debe a que los arreglos multidimensionales en C se almacenan en memoria de forma contigua, lo que significa que se pueden tratar como arreglos unidimensionales.

```
#include <stdio.h>

// Función que recibe un puntero a un arreglo multidimensional
void imprimirMatriz(int (*matriz)[3]) {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            printf("%d ", matriz[i][j]);
        }
        printf("\n");
    }
}

int main() {
    int matriz[3][3] = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };
```

```
    // Llamada a la función
    imprimirMatriz(matriz);

    return 0;
}
```

En este ejemplo, la función `imprimirMatriz` recibe un puntero a un arreglo multidimensional de 3x3. La sintaxis `int (*matriz)[3]` indica que la función recibe un puntero a un arreglo de 3 elementos (es decir, una fila de la matriz). Dentro de la función, podemos tratar el puntero como un arreglo unidimensional de 3 elementos, lo que nos permite acceder a los elementos de la matriz.

1.4 Juego del gato (Tic-Tac-Toe) en C

1.4.1 Inclusión de Bibliotecas:

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

Se incluyen las bibliotecas estándar de C, `<stdio.h>` para entrada/salida estándar y `<stdbool.h>` para usar el tipo de datos booleano y los valores `true` y `false`.

1.4.2 Declaración de Funciones:

```
bool has_free_cell(char board[3][3]);
bool get_winner(char board[3][3]);
```

Se declaran las funciones `has_free_cell` y `get_winner`, que son responsables de verificar si hay celdas libres en el tablero y de determinar si hay un ganador, respectivamente.

1.4.3 Función Principal (`main()`):

```
int main() {
    char board[3][3] = {
        {'-', '-', '-'},
        {'-', '-', '-'},
        {'-', '-', '-'}};
    char player = 'X'; // Jugador actual (inicia con 'X')

    while (has_free_cell(board) && !get_winner(board)) {
        // ...
    }

    // Verificación del ganador o empate
    // ...
    return 0;
}
```

Se declara e inicializa una matriz `board` de 3x3 que representa el tablero del juego, inicializando todas las celdas con el carácter `'-'`. Se declara la variable `player` para almacenar el símbolo del jugador actual, que comienza con `'X'`. Se inicia un bucle `while` que continuará mientras haya celdas libres en el tablero y no haya un ganador. Dentro del bucle, se imprime el tablero, se solicita al jugador que ingrese su jugada y se actualiza el tablero con la jugada del jugador actual. Luego se cambia el jugador. Después de salir del bucle, se verifica si hay un ganador o si hay un empate y se imprime el resultado correspondiente.

1.4.4 Función `has_free_cell()`:

```
bool has_free_cell(char board[3][3]) {
    // ...
}
```

Esta función recibe el tablero como argumento y verifica si hay alguna celda libre (`'-'`) en el tablero. Utiliza dos bucles `while` anidados para iterar sobre todas las celdas del tablero. Si encuentra al menos una celda libre, devuelve `true`; de lo contrario, devuelve `false`.

1.4.5 Función `get_winner()`:

```
bool get_winner(char board[3][3]) {
    // ...
}
```

Esta función recibe el tablero como argumento y verifica si hay un ganador. Comprueba todas las filas, columnas y diagonales para ver si tienen el mismo símbolo (`'X'` o `'O'`) y no son `'-'`. Si encuentra un ganador, devuelve `true`; de lo contrario, devuelve `false`.

1.4.6 Código completo

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

bool has_free_cell(char board[3][3]);
bool get_winner(char board[3][3]);

int main() {
    char board[3][3] = {
        {'-', '-', '-'},
        {'-', '-', '-'},
        {'-', '-', '-'}};
    // Cruz y círculo para jugar
    char player = 'X';

    while (has_free_cell(board) && !get_winner(board)) {
        // Imprimir el tablero
        printf("Tablero:\n");
        int i = 0;
        while (i < 3) {
            int j = 0;
            while (j < 3) {
                printf("%c ", board[i][j]);
                j++;
            }
            printf("\n");
            i++;
        }
        // Pedir la jugada
        int pos, x, y;
        printf("Ingrese la posición de la jugada: ");
        scanf("%d", &pos);
        switch (pos)
        {
        case 0:
```

```
            x = 0;
            y = 0;
            break;
        case 1:
            x = 0;
            y = 1;
            break;
        case 2:
            x = 0;
            y = 2;
            break;
        case 3:
            x = 1;
            y = 0;
            break;
        case 4:
            x = 1;
            y = 1;
            break;
        case 5:
            x = 1;
            y = 2;
            break;
        case 6:
            x = 2;
            y = 0;
            break;
        case 7:
            x = 2;
            y = 1;
            break;
        case 8:
            x = 2;
            y = 2;
            break;
        default:
            printf("Posición inválida\n");
        }
        if (board[x][y] != '-') {
            printf("Posición ocupada\n");
            continue;
        }
        board[x][y] = player;
        // Cambiar de jugador
        if (player == 'X') {
            player = 'O';
        } else {
            player = 'X';
        }
    }
    if (get_winner(board)) {
        if (player == 'X') {
            player = 'O';
        } else {
```

```
            player = 'X';
        }
        printf("¡Felicidades! Ganó el jugador %c\n", player);
    } else {
        printf("¡Empate!\n");
    }

    return 0;
}

bool has_free_cell(char board[3][3]) {
    int i = 0;
    int j = 0;
    while (i < 3) {
        while (j < 3) {
            if (board[i][j] == '-') {
                return true;
            }
            j++;
        }
        i++;
    }
    return false;
}

bool get_winner(char board[3][3]) {
    int i = 0;
    while (i < 3) {
        if (board[i][0] == board[i][1] && board[i][1] == board[i][2] && board[i][0] != '-') {
            return true;
        }
        if (board[0][i] == board[1][i] && board[1][i] == board[2][i] && board[0][i] != '-') {
            return true;
        }
        i++;
    }
    if (board[0][0] == board[1][1] && board[1][1] == board[2][2] && board[0][0] != '-') {
        return true;
    }
    if (board[0][2] == board[1][1] && board[1][1] == board[2][0] && board[0][2] != '-') {
        return true;
    }
    return false;
}
```

Este es un ejemplo de un juego del gato (Tic-Tac-Toe) en C. El programa utiliza una matriz de 3x3 para representar el tablero del juego y permite a dos jugadores alternar turnos para marcar las celdas del tablero. El programa verifica si hay un ganador o si hay un empate después de cada jugada. El juego continúa hasta que haya un ganador o hasta que no haya celdas libres en el tablero. El código utiliza funciones para verificar si hay celdas libres en el tablero y para determinar si hay un ganador. También incluye la lógica para imprimir el tablero y solicitar las jugadas de los jugadores.

1.5 Tic-Tac-Toe para tamaño n

Ahora vamos a ver una posible solución al ejercicio de crear un programa que emule un tictactoe para un tablero de tamaño n. La solución que se presenta a continuación es una de las muchas posibles, y se basa en la implementación de un tablero de tamaño n x n, y en la verificación de las condiciones de victoria para un tablero de tamaño variable.

1.5.1 Inclusión de Bibliotecas:

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

Se incluyen las bibliotecas estándar de C necesarias para el programa, una para el tipo de dato booleano (`<stdbool.h>`) y otra para entrada/salida estándar (`<stdio.h>`).

1.5.2 Declaración de Constantes:

```
#define SIZE 4
```

Se define una constante `SIZE` que representa el tamaño del tablero. En este caso, el tablero será de tamaño 4x4, pero se podrá cambiar y sigue funcionando para cualquier tamaño.

1.5.3 Función `has_free_cell()`:

```
bool has_free_cell(char board[SIZE][SIZE]) {
    int i = 0;
    while(i < SIZE) {
        int j = 0;
        while(j < SIZE) {
            if(board[i][j] == '-') {
                return true;
            }
            j++;
        }
        i++;
    }
    return false;
}
```

La función busca una celda libre en la matriz `board`, representada por el carácter ''-''. La lógica de funcionamiento es la siguiente:

1. Se inicializa un contador i en 0.
2. Se inicia un bucle `while` exterior que itera mientras i sea menor que `SIZE`. Esto permite recorrer todas las filas de la matriz.
3. Dentro de este bucle exterior, se inicializa un contador j en 0.
4. Se inicia un bucle `while` interior que itera mientras j sea menor que `SIZE`. Esto permite recorrer todas las columnas de la matriz para la fila actual.
5. Dentro de este bucle interior, se verifica si el elemento en la posición (i, j) de la matriz `board` es igual a ''-''. Si lo es, significa que hay una celda libre, por lo que la función devuelve `true`.
6. Si no se encuentra una celda libre en la fila actual, se incrementa j para pasar a la siguiente columna.
7. Una vez que se ha recorrido toda la fila actual, se incrementa i para pasar a la siguiente fila y se reinicia j a 0 para empezar a recorrer las columnas de esa fila.

8. Este proceso se repite hasta que se recorren todas las filas y columnas de la matriz.

9. Si no se encuentra ninguna celda libre en toda la matriz, la función devuelve `false` al final.

1.5.4 Función `get_winner()`:

```
bool get_winner(char board[SIZE][SIZE]) {
    int i = 0;
    while(i < SIZE) {
        int j = 0;
        char first = board[i][0];
        while(j < SIZE) {
            if(board[i][j] != first || first == '-') {
                break;
            }
            j++;
        }
        if(j == SIZE) {
            return true;
        }
        i++;
    }

    i = 0;
    while(i < SIZE) {
        int j = 0;
        char first = board[0][i];
        while(j < SIZE) {
            if(board[j][i] != first || first == '-') {
                break;
            }
            j++;
        }
        if(j == SIZE) {
            return true;
        }
        i++;
    }

    char first = board[0][0];
    i = 0;
    while(i < SIZE) {
        if(board[i][i] != first || first == '-') {
            break;
        }
        i++;
    }
    if(i == SIZE) {
        return true;
    }

    first = board[0][SIZE - 1];
    i = 0;
    while(i < SIZE) {
        if(board[i][SIZE - 1 - i] != first || first == '-') {
```

```
            break;
        }
        i++;
    }
    if(i == SIZE) {
        return true;
    }

    return false;
}
```

La función `get_winner`, busca determinar si hay un ganador en el juego representado por la matriz `board` de tamaño `SIZE` por `SIZE` (donde `SIZE` es una constante definida en otro lugar del código). La función busca ganadores en tres posibles direcciones: horizontal, vertical y diagonal.

1. La función comienza recorriendo todas las filas de la matriz `board`. En cada fila, verifica si todos los elementos son iguales al primer elemento de la fila y si ese elemento es diferente de '-' (que indica una celda vacía). Si esto es cierto, devuelve `true`, indicando que hay un ganador.

2. Luego, la función recorre todas las columnas de la matriz `board`. En cada columna, verifica si todos los elementos son iguales al primer elemento de la columna y si ese elemento es diferente de '-'. Si se cumple esta condición, devuelve `true`.

3. Después, la función verifica la diagonal principal de la matriz, es decir, los elementos donde el índice de fila es igual al índice de columna. Si todos los elementos en esta diagonal son iguales al primer elemento de la matriz y ese elemento es diferente de '-', devuelve `true`.

4. Finalmente, la función verifica la otra diagonal de la matriz (de arriba a la derecha hacia abajo a la izquierda). Si todos los elementos en esta diagonal son iguales al primer elemento de la matriz y ese elemento es diferente de '-', devuelve `true`.

Si en ninguno de estos casos se encuentra un ganador, la función devuelve `false`, indicando que no hay un ganador en el juego.

1.5.5 Función Principal (`main()`):

```
int main(){
char board[SIZE][SIZE];
int i = 0;
while(i < SIZE) {
    int j = 0;
    while(j < SIZE) {
        board[i][j] = '-';
        j++;
    }
    i++;
}
// Cruz y círculo para jugar
char player = 'X';

while(has_free_cell(board) && !get_winner(board)) {
    // Imprimir el tablero
    i = 0;
    while(i < SIZE) {
        int j = 0;
        while(j < SIZE) {
```

```
                printf("%c ", board[i][j]);
                j++;
            }
            printf("\n");
            i++;
        }

        // Pedir la jugada
        int position;
        printf("Jugador %c, ingrese la posición: ", player);
        scanf("%d", &position);

        // Verificar que se inserte una posición válida
        int x = position / SIZE;
        int y = position % SIZE;
        if(x < 0 || x >= SIZE || y < 0 || y >= SIZE || board[x][y] != '-') {
            printf("Posición inválida o ya ocupada, intente de nuevo.\n");
            continue;
        }

        // Realizar la jugada y cambiar de jugador
        board[x][y] = player;
        player = (player == 'X') ? 'O' : 'X';
    }

    // Corroborar si hay ganador
    if (get_winner(board)) {
        if (player == 'X') {
            player = 'O';
        } else {
            player = 'X';
        }
        printf("¡Felicidades! Ganó el jugador %c\n", player);
    } else {
        printf("¡Empate!\n");
    }
    return 0;
}
```

La función principal (`main()`) es el punto de entrada del programa. En este caso, el programa implementa un juego de tres en raya (Tic-Tac-Toe). A continuación se explica la lógica de la función:

1. Se declara una matriz bidimensional `board` de tamaño `SIZE` por `SIZE` para representar el tablero del juego.
2. Se inicializa el tablero, rellenándolo con el carácter ''-'', indicando que todas las celdas están vacías.
3. Se establece el jugador actual como ''X'', que será el primer jugador en realizar una jugada.
4. Se inicia un bucle que continuará mientras haya celdas libres en el tablero y no haya un ganador.
5. Dentro del bucle, se imprime el tablero actual.
6. Se solicita al jugador actual que ingrese la posición donde desea realizar su jugada.
7. Se verifica que la posición ingresada sea válida y esté libre en el tablero. Si no lo es, se muestra un mensaje de error y se solicita una nueva posición.

8. Si la posición es válida, se realiza la jugada colocando el símbolo del jugador actual en la posición correspondiente del tablero.

9. Se cambia el jugador actual para que el otro jugador pueda realizar su jugada en el siguiente ciclo del bucle.

10. Una vez que el bucle termina (debido a un ganador o un empate), se verifica si hay un ganador en el tablero. Si hay un ganador, se imprime un mensaje felicitando al jugador ganador. Si no hay ganador, se imprime un mensaje indicando que hubo un empate.

11. La función retorna 0 para indicar que el programa se ejecutó correctamente.

Esta función implementa la lógica principal del juego, gestionando las jugadas de los jugadores, la validación de las posiciones y la determinación del ganador o empate.

2 Arreglos, archivos y módulos

2.1 Abrir un Archivo (`fopen`)

La función `fopen()` se utiliza para abrir un archivo en C. Toma dos argumentos principales: el nombre del archivo y el modo en el que se desea abrirlo (lectura, escritura, etc.). Por ejemplo:

```
FILE *archivo;
archivo = fopen("archivo.txt", "r"); // Abre el archivo en modo lectura
if (archivo == NULL) {
    printf("No se pudo abrir el archivo\n");
    return 1;
}
```

2.1.1 Modos de Apertura de Archivo para `fopen()` en C

La función `fopen()` en C admite varios modos de apertura de archivo, que se especifican como el segundo parámetro. Aquí tienes una lista de los modos más comunes que puedes utilizar:

- `"r"`: Abre el archivo en modo lectura. El archivo debe existir, de lo contrario, `fopen()` devolverá `NULL`.
- `"w"`: Abre el archivo en modo escritura. Si el archivo no existe, se crea. Si el archivo existe, su contenido se sobrescribe.
- `"a"`: Abre el archivo en modo anexar (append). Si el archivo no existe, se crea. Si el archivo existe, los datos se agregan al final del archivo sin sobrescribir el contenido existente.
- `"r+"`: Abre el archivo en modo lectura/escritura. El archivo debe existir, de lo contrario, `fopen()` devolverá `NULL`. El puntero de archivo está al principio del archivo.
- `"w+"`: Abre el archivo en modo lectura/escritura. Si el archivo no existe, se crea. Si el archivo existe, su contenido se sobrescribe.
- `"a+"`: Abre el archivo en modo lectura/escritura. Si el archivo no existe, se crea. Si el archivo existe, los datos se agregan al final del archivo sin sobrescribir el contenido existente.

Además de estos modos básicos, también hay modos adicionales que pueden ser útiles en ciertas situaciones específicas.

- `"rb"`, `"wb"`, `"ab"`: Modos de apertura de archivo en modo binario (para archivos binarios en lugar de archivos de texto).
- `"rU"`, `"wU"`, `"aU"`: Modos de apertura de archivo en modo universal (universal newline mode), que permite manejar diferentes estilos de saltos de línea en archivos de texto.

2.2 Leer Datos desde un Archivo (fscanf)

Una vez que el archivo está abierto, puedes leer datos desde él utilizando `fscanf()` u otras funciones como `fgets()` o `fread()`. `fscanf()` lee datos formateados desde el archivo, similar a `scanf()` pero con el archivo como fuente en lugar de la entrada estándar. Por ejemplo:

```
int num;
fscanf(archivo, "%d", &num); // Lee un entero del archivo
```

2.3 Cerrar un Archivo (fclose)

Es importante cerrar un archivo después de haber terminado de trabajar con él para liberar los recursos del sistema operativo. Esto se hace con la función `fclose()`. Por ejemplo:

```
fclose(archivo); // Cierra el archivo
```

2.4 Punteros y Direcciones de Memoria

- En C, los punteros son variables que almacenan direcciones de memoria.
- Cuando abres un archivo en C, `fopen()` devuelve un puntero de tipo `FILE*`. Este puntero apunta a una estructura de datos interna que representa el archivo abierto.
- Cuando lees o escribes en un archivo utilizando funciones como `fscanf()` o `fprintf()`, proporcionas la dirección de memoria donde se deben almacenar o desde donde se deben leer los datos.

2.5 Ejemplo Completo

Este programa lee un archivo de texto llamado "datos.txt", que contiene números enteros separados por espacios, los suma y muestra el resultado:

```
#include <stdio.h>

int main() {
    FILE *archivo;
    int num, suma = 0;

    archivo = fopen("datos.txt", "r");
    if (archivo == NULL) {
        printf("No se pudo abrir el archivo\n");
        return 1;
    }

    while (fscanf(archivo, "%d", &num) != EOF) {
        suma += num;
    }

    printf("La suma de los numeros en el archivo es: %d\n", suma);

    fclose(archivo);

    return 0;
}
```

2.6 Función `array_from_file()`

A continuación, se muestra una función que lee un archivo de texto que contiene números enteros separados por espacios y los almacena en un arreglo dinámico. La función toma el nombre del archivo como argumento y devuelve un puntero al arreglo dinámico que contiene los números leídos.

```
unsigned int array_from_file(int array[],
            unsigned int max_size,
            const char *filepath) {
    //your code here!!!
        unsigned int array_size;
        FILE *f = fopen(filepath,"r");
        fscanf(f,"%u",&array_size);

        if(array_size > max_size){
                //Accederia a memoria no habilitada
                array_size = max_size;
        }

        for(unsigned int i = 0;i<array_size;i++){
                fscanf(f,"%d",&array[i]);
        }

        return array_size;
}
```

1. **Apertura del archivo**: El código comienza abriendo el archivo especificado por `filepath` en modo lectura (`"r"`) utilizando la función `fopen()`. Se utiliza un puntero de tipo `FILE` para manejar el archivo.

2. **Lectura del tamaño del array desde el archivo**: Utilizando `fscanf()`, se lee el primer valor del archivo, que representa el tamaño del array. Este valor se almacena en la variable `array_size`.

3. **Verificación del tamaño del array**: Se compara `array_size` con `max_size` para asegurarse de que el tamaño del array no exceda el límite especificado por el parámetro `max_size`. Si `array_size` es mayor que `max_size`, significa que el tamaño del array excede el límite y se ajusta `array_size` a `max_size` para evitar acceder a memoria no permitida.

4. **Lectura de elementos del array desde el archivo**: Se utiliza un bucle `for` para leer `array_size` elementos del archivo y almacenarlos en el array `array[]`. En cada iteración del bucle, se utiliza `fscanf()` para leer un entero desde el archivo y almacenarlo en la posición correspondiente del array.

5. **Cierre del archivo**: Una vez que se han leído todos los elementos del archivo y se han almacenado en el array, se cierra el archivo utilizando `fclose()` para liberar los recursos asociados al mismo.

6. **Retorno del tamaño del array leído**: Se devuelve `array_size`, que representa el número de elementos que se han leído y almacenado en el array `array[]`.

Por lo tanto, esta función se encarga de leer un array de enteros desde un archivo especificado por `filepath`, asegurándose de que el tamaño del array no exceda un límite dado por `max_size`, y luego devuelve el número de elementos leídos y almacenados en el array.

2.7 Función `array_dump()`

La función `array_dump()` se encarga de imprimir en la consola los elementos de un array de enteros `a[]` junto con su longitud `length`.

```
void array_dump(const int a[], unsigned int length) {
    printf("[");
    for(unsigned int i = 0;i<length;i++){
        if(i != 0){
            printf(", ");
        }
        printf("%d",a[i]);
    }
    printf("]\n");
}
```

A continuación, se explica paso a paso la lógica de la función:

1. **Impresión del encabezado del array**: La función comienza imprimiendo el carácter '[' utilizando `printf()` para indicar el comienzo del array en el formato de salida.
2. **Iteración a través del array**: Se utiliza un bucle `for` para iterar sobre cada elemento del array `a[]` desde el índice 0 hasta `length - 1`.
3. **Impresión de los elementos del array**: Dentro del bucle, se utiliza `printf()` para imprimir cada elemento del array. Antes de imprimir cada elemento, se verifica si es el primer elemento del array (`i != 0`). Si no es el primer elemento, se imprime una coma y un espacio para separar los elementos del array.
4. **Impresión del pie del array**: Una vez que se han impreso todos los elementos del array, se imprime el carácter ']' utilizando `printf()` para indicar el final del array en el formato de salida. Además, se añade un salto de línea (`\n`) para que la próxima salida en la consola esté en una nueva línea.

Por lo tanto, esta función se encarga de imprimir en la consola los elementos de un array de enteros `a[]` junto con su longitud `length` en un formato legible.

2.8 Implementación de las funciones en un programa principal

```
int main(int argc, char *argv[]) {
    char *filepath = NULL;

    // Verificar si se proporciono un argumento de linea de comandos
    filepath = parse_filepath(argc, argv);

    // Crear un array de enteros
    int array[MAX_SIZE];

    // Leer los elementos del archivo al array
    unsigned int length = array_from_file(array, MAX_SIZE, filepath);

    // Imprimir los elementos del array
    array_dump(array, length);

    return EXIT_SUCCESS;
}
```

El código principal (`main()`) utiliza las funciones `parse_filepath`, `array_from_file` y `array_dump` de la siguiente manera:

1. **Análisis de la ruta del archivo**: Se llama a la función `parse_filepath` para analizar los argumentos de la línea de comandos (`argc` y `argv[]`). Su objetivo es extraer la ruta del archivo del cual se leerá el array. El resultado se asigna a la variable `filepath`.

2. **Creación del array**: Se declara un array de enteros llamado `array` con un tamaño máximo definido por `MAX_SIZE`.

3. **Lectura de los elementos del archivo al array**: Se llama a la función `array_from_file` para leer los elementos del array desde el archivo especificado por `filepath`. Los elementos se almacenan en el array `array[]` y el número real de elementos leídos se devuelve como `length`.

4. **Impresión de los elementos del array**: Después de llenar el array con los datos del archivo, se llama a la función `array_dump` para imprimir los elementos del array junto con su longitud.

5. **Retorno del programa**: Finalmente, se devuelve `EXIT_SUCCESS` al sistema operativo para indicar que el programa se ejecutó correctamente.

2.9 Función `array_from_stdin()`