



**Universidad Nacional  
Autónoma de México**  
Facultad de Ingeniería



**Fundamentos de Programación (1122)**

Laboratorios de computación  
salas A y B

*Profesor: M.I. Marco Antonio Martínez Quintana*  
*Semestre 2021-1*

**Practica No. 11**

**Arreglos unidimensionales y multidimensionales**

Grupo: 1129

No. de Equipo de cómputo empleado: No aplica

No. de Lista o Brigada: No aplica

No. de Lista: 42

**Nombre: Adolfo Román Jiménez**

## **Objetivo:**

Reconocer la importancia y utilidad de los arreglos, en la elaboración de programas que resuelvan problemas que requieran agrupar datos del mismo tipo, así como trabajar con arreglos tanto unidimensionales como multidimensionales.

## **Introducción:**

Un arreglo (vector, array, matriz) es un conjunto de datos o una estructura de datos homogéneos que se encuentran ubicados en forma consecutiva en la memoria RAM (sirve para almacenar datos en forma temporal). También puede definirse como un grupo o una colección finita, homogénea y ordenada de elementos. Los arreglos pueden ser de los siguientes tipos:

- De una dimensión.
- De dos dimensiones.
- De tres o más dimensiones.

A cada elemento (dato) del arreglo se le asocia una posición particular, el cual se requiere indicar para acceder a un elemento en específico. Esto se logra a través del uso de índices.

Los arreglos pueden ser unidimensionales o multidimensionales. Los arreglos se utilizan para hacer más eficiente el código de un programa.

### **Arreglos unidimensionales**

Es un tipo de datos estructurado que está formado de una colección finita y ordenada de datos del mismo tipo. Es la estructura natural para modelar listas de elementos iguales. Están formados por un conjunto de elementos de un mismo tipo de datos que se almacenan bajo un mismo nombre, y se diferencian por la posición que tiene cada elemento dentro del arreglo de datos. Al declarar un arreglo, se debe inicializar sus elementos antes de utilizarlos. Para declarar un arreglo tiene que indicar su tipo, un nombre único y la cantidad de elementos que va a contener.

### **Arreglos multidimensionales**

Es un tipo de dato estructurado, que está compuesto por dimensiones. Para hacer referencia a cada componente del arreglo es necesario utilizar  $n$  índices, uno para cada dimensión. El término dimensión representa el número de índices utilizados para referirse a un elemento particular en el arreglo. Los arreglos de más de una dimensión se llaman arreglos multidimensionales.

### **Arreglos con múltiples subíndices**

Es la representación de tablas de valores, consistiendo de información arreglada en renglones y columnas. Para identificar un elemento particular de la tabla, deberemos de especificar dos subíndices; el primero identifica el renglón del elemento y el segundo

identifica la columna del elemento. A los arreglos que requieren dos subíndices para identificar un elemento en particular se conocen como arreglo de doble subíndice. Note que los arreglos de múltiples subíndices pueden tener más de dos subíndices. El estándar ANSI indica que un sistema ANSI C debe soportar por lo menos 12 subíndices de arreglo.

## Arreglos unidimensionales

Un arreglo unidimensional de  $n$  elementos en la memoria se almacena de la siguiente manera:



La primera localidad del arreglo corresponde al índice 0 y la última corresponde al índice  $n-1$ , donde  $n$  es el tamaño del arreglo.

La sintaxis para definir un arreglo en lenguaje C es la siguiente:

`tipoDeDato nombre[tamaño]`

Donde nombre se refiere al identificador del arreglo, tamaño es un número entero y define el número máximo de elementos que puede contener el arreglo. Un arreglo puede ser de los tipos de dato entero, real, carácter o estructura.

**NOTA:** Los tipos de datos estructuras no se abordarán en esta práctica.

## Código (arreglo unidimensional while)

```
practica11.c x mario.c +
1 #include <stdio.h>
2
3 /*
4 Este programa genera un arreglo unidimensional de 5 elementos y
5 los accede a cada elemento del arreglo a través de un ciclo while.
6 */
7
8 int main ()
9 {
10     #define TAMANO 5
11     int lista[TAMANO] = {10, 8, 5, 8, 7};
12
13     int indice = 0;
14
15     printf("\tLista\n");
16
17     while (indice < 5 )
18     {
19         printf("\nCalificación del alumno %d es %d", indice+1, lista[indice]);
20         indice += 1;    // análogo a indice = indice + 1;
21     }
22
23     printf("\n");
24
25     return 0;
26 }
27
```

```
~/ x +
~/ $ clear
~/ $ clang -o test practica11.c
~/ $ ./test
    Lista

Calificación del alumno 1 es 10
Calificación del alumno 2 es 8
Calificación del alumno 3 es 5
Calificación del alumno 4 es 8
Calificación del alumno 5 es 7
~/ $
```

## Código (arreglo unidimensional for)

```
practica11.c x mario.c +
1 #include <stdio.h>
2
3 /*
4 Este programa genera un arreglo unidimensional de 5 elementos
5 y accede a cada elemento del arreglo a través de un ciclo for.
6 */
7
8 int main ()
9 {
10     #define TAMANO 5
11     int lista[TAMANO] = {10, 8, 5, 8, 7};
12
13     printf("\tLista\n");
14
15     for (int indice = 0 ; indice < 5 ; indice++)
16     {
17         printf("\nCalificación del alumno %d es %d", indice+1, lista[indice]);
18     }
19
20     printf("\n");
21
22     return 0;
23 }
24
```

```
~/ $ clang -o test practica11.c
~/ $ ./test
    Lista

Calificación del alumno 1 es 10
Calificación del alumno 2 es 8
Calificación del alumno 3 es 5
Calificación del alumno 4 es 8
Calificación del alumno 5 es 7
~/ $
```

## Apuntadores

Un apuntador es una variable que contiene la dirección de una variable, es decir, hace referencia a la localidad de memoria de otra variable. Debido a que los apuntadores trabajan directamente con la memoria, a través de ellos se accede con rapidez a un dato.

La sintaxis para declarar un apuntador y para asignarle la dirección de memoria de otra variable es, respectivamente:

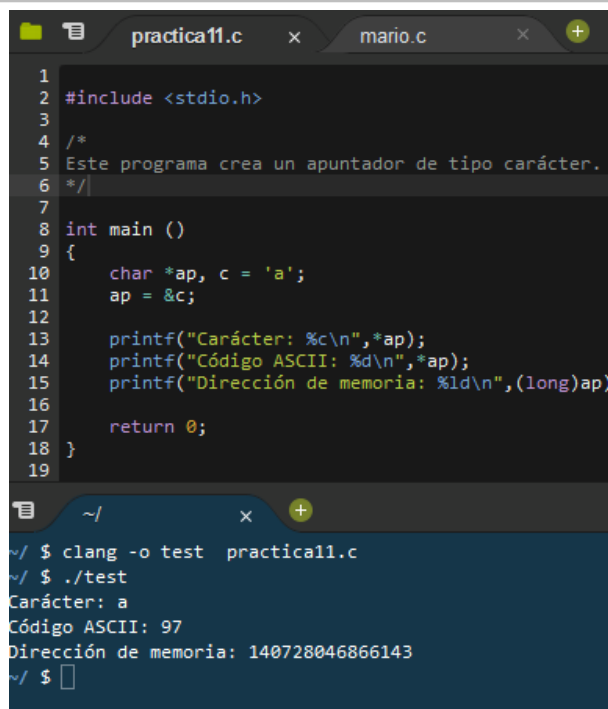
```
TipoDeDato *apuntador, variable;  
apuntador = &variable;
```

La declaración de una variable apuntador inicia con el carácter \*. Cuando a una variable le antecede un ampersand, lo que se hace es acceder a la dirección de memoria de la misma (es lo que pasa cuando se lee un dato con scanf).

Los apuntadores solo pueden apuntar a direcciones de memoria del mismo tipo de dato con el que fueron declarados; para acceder al contenido de dicha dirección, a la variable apuntador se le antepone \*.

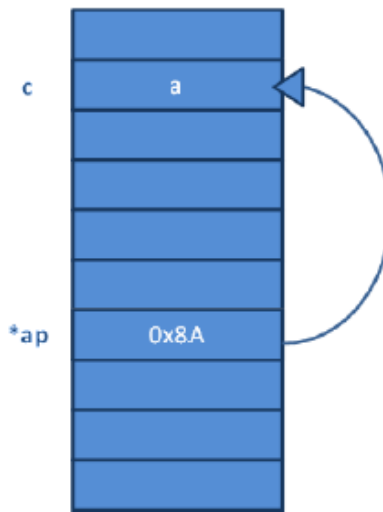
### Código (apuntadores)

```
#include <stdio.h>  
  
/*  
 Este programa crea un apuntador de tipo carácter.  
*/  
  
int main () {  
    char *ap, c = 'a';  
    ap = &c;  
  
    printf("Carácter: %c\n",*ap);  
    printf("Código ASCII: %d\n",*ap);  
    printf("Dirección de memoria: %d\n",ap);  
  
    return 0;  
}
```



The screenshot shows a code editor with two tabs: 'practica11.c' and 'mario.c'. The 'practica11.c' tab is active, displaying the C code from the previous block. Below the code editor is a terminal window. The terminal shows the command to compile the program using 'clang' and the command to run the resulting executable 'test'. The output of the program is displayed in the terminal, showing the character 'a', its ASCII code 97, and its memory address 140728046866143.

```
practica11.c x mario.c +  
1  
2 #include <stdio.h>  
3  
4 /*  
5 Este programa crea un apuntador de tipo carácter.  
6 */  
7  
8 int main ()  
9 {  
10     char *ap, c = 'a';  
11     ap = &c;  
12  
13     printf("Carácter: %c\n",*ap);  
14     printf("Código ASCII: %d\n",*ap);  
15     printf("Dirección de memoria: %ld\n", (long)ap);  
16  
17     return 0;  
18 }  
19  
~/ $ clang -o test practica11.c  
~/ $ ./test  
Carácter: a  
Código ASCII: 97  
Dirección de memoria: 140728046866143  
~/ $
```



Un apuntador almacena la dirección de memoria de la variable a la que apunta

### Código (apuntadores)

```
#include<stdio.h>

/*
    Este programa accede a las localidades de memoria de distintas variables a
    través de un apuntador.
*/

int main () {
    int a = 5, b = 10, c[10] = {5, 4, 3, 2, 1, 9, 8, 7, 6, 0};
    int *apEnt;
    apEnt = &a;

    printf("a = 5, b = 10, c[10] = {5, 4, 3, 2, 1, 9, 8, 7, 6, 0}\n");
    printf("apEnt = &a\n");

    b = *apEnt;
    printf("b = *apEnt \t-> b = %i\n", b);

    b = *apEnt +1;
    printf("b = *apEnt + 1 \t-> b = %i\n", b);

    *apEnt = 0;
    printf("*apEnt = 0 \t-> a = %i\n", a);

    apEnt = &c[0];
    printf("apEnt = &c[0] \t-> apEnt = %i\n", *apEnt);

    return 0;
}
```

```
practica11.c x mario.c +
1 #include<stdio.h>
2
3 /*
4 Este programa accede a las localidades de memoria de
5 distintas variables a través de un apuntador.
6 */
7
8 int main ()
9 {
10     int a = 5, b = 10, c[10] = {5, 4, 3, 2, 1, 9, 8, 7, 6, 0};
11     int *apEnt;
12     apEnt = &a;
13
14     printf("a = 5, b = 10, c[10] = {5, 4, 3, 2, 1, 9, 8, 7, 6, 0}\n");
15     printf("apEnt = &a\n");
16
17     b = *apEnt;
18     printf("b = *apEnt \t-> b = %i\n", b);
19
20     b = *apEnt + 1;
21     printf("b = *apEnt + 1 \t-> b = %i\n", b);
22
23     *apEnt = 0;
24     printf("*apEnt = 0 \t-> a = %i\n", a);
25
26     apEnt = &c[0];
27     printf("apEnt = &c[0] \t-> apEnt = %i\n", *apEnt);
28
29     return 0;
30 }
31
```

```
~/ $ clang -o test practica11.c
~/ $ ./test
a = 5, b = 10, c[10] = {5, 4, 3, 2, 1, 9, 8, 7, 6, 0}
apEnt = &a
b = *apEnt      -> b = 5
b = *apEnt + 1  -> b = 6
*apEnt = 0      -> a = 0
apEnt = &c[0]   -> apEnt = 5
~/ $
```

Cabe mencionar que el nombre de un arreglo es un apuntador fijo al primero de sus elementos; por lo que las siguientes instrucciones, para el código de arriba, son equivalentes:

```
apEnt = &c[0];
apEnt = c;
```



## Código (apuntadores)

```
#include <stdio.h>

/*
    Este programa trabaja con aritmética de apuntadores para acceder a los
    valores de un arreglo.
*/

int main () {
    int arr[] = {5, 4, 3, 2, 1};
    int *apArr;
    apArr = arr;

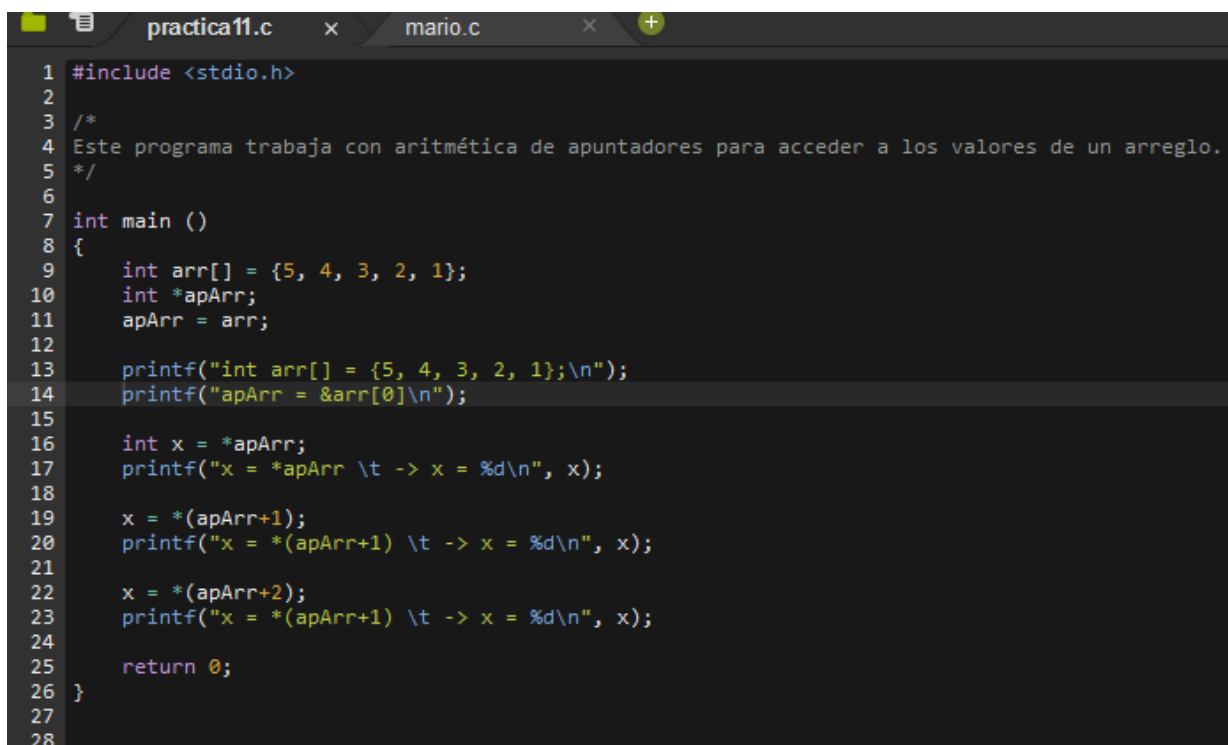
    printf("int arr[] = {5, 4, 3, 2, 1};\n");
    printf("apArr = &arr[0]\n");

    int x = *apArr;
    printf("x = *apArr \t -> x = %d\n", x);

    x = *(apArr+1);
    printf("x = *(apArr+1) \t -> x = %d\n", x);

    x = *(apArr+2);
    printf("x = *(apArr+1) \t -> x = %d\n", x);

    return 0;
}
```



```
1 #include <stdio.h>
2
3 /*
4  Este programa trabaja con aritmética de apuntadores para acceder a los valores de un arreglo.
5  */
6
7 int main ()
8 {
9     int arr[] = {5, 4, 3, 2, 1};
10    int *apArr;
11    apArr = arr;
12
13    printf("int arr[] = {5, 4, 3, 2, 1};\n");
14    printf("apArr = &arr[0]\n");
15
16    int x = *apArr;
17    printf("x = *apArr \t -> x = %d\n", x);
18
19    x = *(apArr+1);
20    printf("x = *(apArr+1) \t -> x = %d\n", x);
21
22    x = *(apArr+2);
23    printf("x = *(apArr+1) \t -> x = %d\n", x);
24
25    return 0;
26 }
27
28
```

## Código compilado y ejecutado

```
~/ $ clang -o test practica11.c
~/ $ ./test
int arr[] = {5, 4, 3, 2, 1};
apArr = &arr[0]
x = *apArr      -> x = 5
x = *(apArr+1)  -> x = 4
x = *(apArr+1)  -> x = 3
~/ $
```

## Código (apuntadores en ciclo for)

```
#include <stdio.h>

/*
 * Este programa genera un arreglo unidimensional de 5 elementos y
 * accede a cada elemento del arreglo a través de un apuntador
 * utilizando un ciclo for.
 */

int main (){
    #define TAMANO 5
    int lista[TAMANO] = {10, 8, 5, 8, 7};
    int *ap = lista;

    printf("\tLista\n");
    for (int indice = 0 ; indice < 5 ; indice++){
        printf("\nCalificación del alumno %d es %d", indice+1, *(ap+indice));
    }

    printf("\n");

    return 0;
}
```

```
practica11.c x mario.c +
1 #include <stdio.h>
2
3 /*
4  * Este programa genera un arreglo unidimensional de 5 elementos y accede a
5  * cada elemento del arreglo a través de un apuntador utilizando un ciclo for.
6  */
7
8 int main ()
9 {
10     #define TAMANO 5
11
12     int lista[TAMANO] = {10, 8, 5, 8, 7};
13     int *ap = lista;
14
15     printf("\tLista\n");
16
17     for (int indice = 0 ; indice < 5 ; indice++){
18     {
19         printf("\nCalificación del alumno %d es %d", indice+1, *(ap+indice));
20     }
21
22     printf("\n");
23     return 0;
24 }
25
```

```
~/ $ clang -o test practica11.c
~/ $ ./test
Lista

Calificación del alumno 1 es 10
Calificación del alumno 2 es 8
Calificación del alumno 3 es 5
Calificación del alumno 4 es 8
Calificación del alumno 5 es 7
~/ $
```

## Código (apuntadores en cadenas)

```
#include <stdio.h>

/*
Este programa muestra el manejo de cadenas en lenguaje C.
*/

int main(){
    char palabra[20];
    int i=0;

    printf("Ingrese una palabra: ");
    scanf("%s", palabra);
    printf("La palabra ingresada es: %s\n", palabra);

    for (i = 0 ; i < 20 ; i++){
        printf("%c\n", palabra[i]);
    }

    return 0;
}
```

```
1  #include <stdio.h>
2
3
4  /*
5  Este programa muestra el manejo de cadenas en lenguaje C.
6  */
7
8  int main()
9  {
10     char palabra[20]; int i=0;
11
12     printf("Ingrese una palabra: ");
13     scanf("%s", palabra);
14     printf("La palabra ingresada es: %s\n", palabra);
15
16     for (i = 0 ; i < 20 ; i++)
17     {
18         printf("%c\n", palabra[i]);
19     }
20
21     return 0;
22 }
23
```

```
~/ $ clang -o test practica11.c
~/ $ ./test
Ingrese una palabra: Oasis
La palabra ingresada es: Oasis
O
a
s
i
s

P

@

~/ $
```

## Arreglos multidimensionales

Lenguaje C permite crear arreglos de varias dimensiones con la siguiente sintaxis:

```
tipoDato nombre[ tamaño ][ tamaño ]...[tamaño];
```

Donde nombre se refiere al identificador del arreglo, tamaño es un número entero y define el número máximo de elementos que puede contener el arreglo por dimensión (el número de dimensiones está determinado por el número de corchetes). Los tipos de dato que puede tolerar un arreglo multidimensional son: entero, real, carácter o estructura.

De manera práctica se puede considerar que la primera dimensión corresponde a los renglones, la segunda a las columnas, la tercera al plano, y así sucesivamente. Sin embargo, en la memoria cada elemento del arreglo se guarda de forma contigua, por lo tanto, se puede recorrer un arreglo multidimensional con apuntadores.

## Código (arreglos multidimensionales)

```
#include<stdio.h>

/* Este programa genera un arreglo de dos dimensiones (arreglo
multidimensional) y accede a sus elementos a través de dos ciclos
for, uno anidado dentro de otro.
*/

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

    int i, j;

    printf("Imprimir Matriz\n");

    for (i=0 ; i<3 ; i++){
        for (j=0 ; j<3 ; j++){
            printf("%d, ",matriz[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

```
1 #include<stdio.h>
2
3 /* Este programa genera un arreglo de dos
4 dimensiones (arreglo multidimensional) y
5 accede a sus elementos a través de dos
6 ciclos for, uno anidado dentro de otro.
7 */
8
9 int main()
10 {
11     int matriz[3][3] = {{1,2,3},{4,5,6},{7,8,9}};
12
13     int i, j;
14     printf("Imprimir Matriz\n");
15
16     for (i=0 ; i<3 ; i++)
17     {
18         for (j=0 ; j<3 ; j++)
19         {
20             printf("%d, ",matriz[i][j]);
21         }
22         printf("\n");
23     }
24
25     return 0;
26 }
27
28
```

```
~/ $ clear
~/ $ clang -o test practica11.c
~/ $ ./test
Imprimir Matriz
1, 2, 3,
4, 5, 6,
7, 8, 9,
~/ $
```

## Código (arreglos multidimensionales con apuntadores)

```
#include<stdio.h>

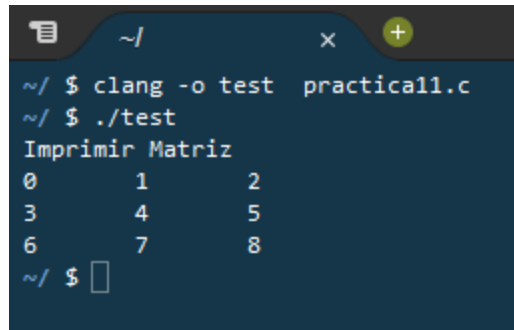
/* Este programa genera un arreglo de dos dimensiones (arreglo
multidimensional) y accede a sus elementos a través de un apuntador utilizando
un ciclo for.
*/

int main(){
    int matriz[3][3] = {{1,2,3},{4,5,6},{7,8,9}};
    int i, cont=0, *ap;
    ap = matriz;

    printf("Imprimir Matriz\n");
    for (i=0 ; i<9 ; i++){
        if (cont == 3){
            printf("\n");
            cont = 0;
        }
        printf("%d\t",*(ap+i));
        cont++;
    }
    printf("\n");

    return 0;
}
```

```
1 #include<stdio.h>
2
3 /* Este programa genera un arreglo de dos dimensiones
4 (arreglo multidimensional) y accede a sus elementos a
5 través de un apuntador utilizando un ciclo for.
6 */
7
8 int main()
9 {
10     int matriz[3][3] = {{1,2,3},{4,5,6},{7,8,9}};
11     int i, cont=0, *ap;
12
13     ap = matriz;
14
15     printf("Imprimir Matriz\n");
16     for (i=0 ; i<9 ; i++)
17     {
18         if (cont == 3)
19         {
20             printf("\n"); cont = 0;
21         }
22         printf("%d\t",*(ap+i)); cont++;
23     }
24     printf("\n");
25
26     return 0;
27 }
28
29
30
```



```
~/ $ clang -o test practica11.c
~/ $ ./test
Imprimir Matriz
0      1      2
3      4      5
6      7      8
~/ $
```

## Actividades:

- Elaborar un programa en lenguaje C que emplee arreglos de una dimensión.
- Resolver un problema que requiera el uso de un arreglo de dos dimensiones, a través de un programa en lenguaje C.
- Manipular arreglos a través de índices y apuntadores.

Scrabble.c

Readability.c

(Arreglo en una dimensión)

El siguiente programa es una implementación del índice Coleman-Liau. El cual está diseñado para mostrar que nivel escolar en los Estados Unidos es necesario para entender cierto texto sometido a prueba por medio del algoritmo escrito.

Esto no implica que una persona no pueda leer o entender cierto texto si es que no tiene la escolaridad que el índice indica, sino simplemente es lo que en general, de acuerdo con la complejidad del texto, este puede estar enfocado a ciertas audiencias con diferentes grados de escolaridad. De esta manera los textos para personas con educación universitaria que por lo regular manejan palabras más complejas tendrán un grado más alto de complejidad que los textos que contienen lenguaje más coloquial y que se puede manejar por ejemplo, en las novelas de Harry Potter.

El programa está escrito en inglés y el algoritmo está enfocado a los textos en inglés para la población estadounidense.

La principal fórmula que nos dará el grado estimado de escolaridad para comprender cierto texto es la siguiente:

$$\text{Grade} = 0.588 * L - 0.296 * S - 15.8$$

Donde “Grade” es el grado de escolaridad requerido, “L” es el contenido porcentual de letras por cada 100 palabras y “S” es el contenido porcentual de frases por cada 100 palabras de igual forma.

Este programa, al momento de ingresar textos completos los transforma en una “string” que es una cadena de caracteres y consecuentemente se convierten en un array.

```
1 #include <stdio.h>
2 #include <cs50.h>
3 #include <string.h>
4 #include <math.h>
5
6 int main(void)
7 {
8     string text = get_string("Text: "); //Prompts user for text to evaluate
9
10    float letters = 0, words = 1, sentences = 0; //Declares variables
11
```

Lo que nuestro programa hace es tomar ese array y contar casilla por casilla, cuantas palabras, oraciones y letras contiene. Esto se logra a través de identificar los signos de puntuación dentro del arreglo y algo de conocimiento de código ASCII.

Por ejemplo, si existe un espacio en blanco, podemos inferir que detrás de este espacio existe una palabra.

Para las oraciones, podemos contar los puntos, signos de interrogación y de exclamación que por lo regular son indicadores de que se termino una oración.

Y para las letras lo único que hacemos es evaluar que la casilla del arreglo que en ese momento se evalúa, su valor como entero o como carácter este dentro del rango de lo que corresponde a las mayúsculas y minúsculas dentro del ASCII.

```
12     for (int i = 0, n = strlen(text); i < n; i++)
13     {
14         if (text[i] == ' ') // Counts words
15         {
16             words++;
17         }
18         else if (text[i] == '!' || text[i] == '.' || text[i] == '?') //Counts sentences
19         {
20             sentences++;
21         }
22         else if ((text[i] >= 'A' && text[i] <= 'Z') || (text[i] >= 'a' && text[i] <= 'z')) //Counts letters
23         {
24             letters++;
25         }
26     }
27
```

Cuando tenemos estos valores, otorgamos el valor a L, a S y con esto, el programa resuelve la fórmula que lanza el resultado.



```

27
28     double L = (letters / words) * 100; //Letters percent over words
29     double S = (sentences / words) * 100; //Sentences percent over words
30
31     double grade = 0.0588 * L - 0.296 * S - 15.8; // Grade evaluation
32

```

Si este es menor a grado 1, se le indicara que el texto puede ser leído casi por cualquier persona. Si esta entre grado 1 y grado 16, se le indicara el nivel resultante y si es mayor a 16, simplemente se le mencionara que es mayor a 16.

```

33     if (grade < 1) //For grade less than 1
34     {
35         printf("Before Grade 1\n");
36     }
37     else if (grade > 16) //For grade bigger than 16
38     {
39         printf("Grade 16+\n");
40     }
41     else // For regular grade
42     {
43         printf("Grade %.01f\n", round(grade)); // Rounds grade
44     }
45 }

```

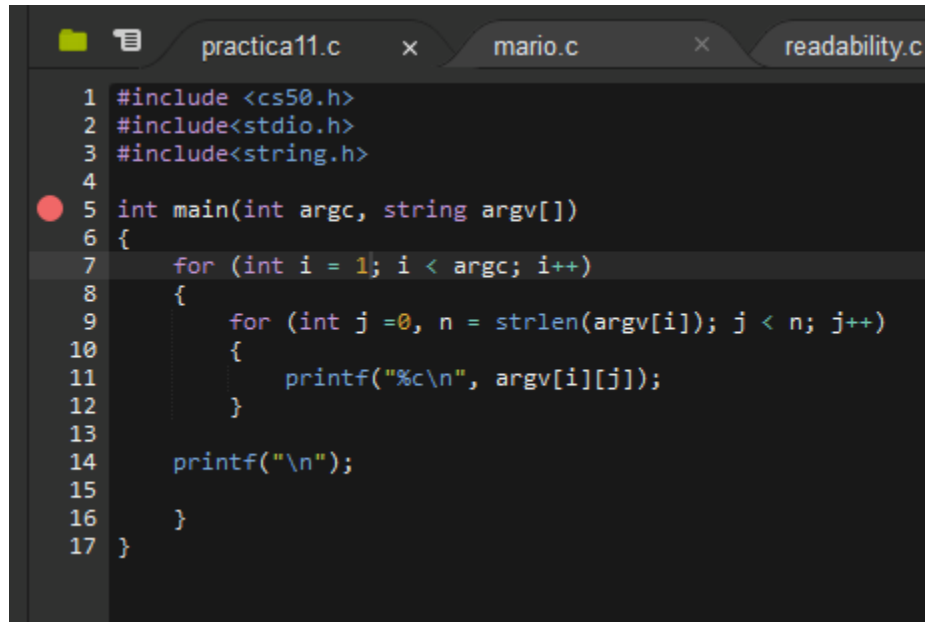
Funcionamiento:

```

~/ $ clear
~/ $ make practical1
clang -ggdb3 -O0 -std=c11 -Wall -Werror -Wextra -Wno-sign-compare -Wno-unused-parameter -Wno-unused-variable -Wshadow practical1.c -lcrypt -lcs50 -lm -o practical1
~/ $ ./practical1
Text: Harry Potter was a highly unusual boy in many ways. For one thing, he hated the summer holidays more than any other time of year. For another, he really wanted to do his homework, but was forced to do it in secret, in the dead of the night. And he also happened to be a wizard.
Grade 5
~/ $ ./practical1
Text: As the average number of letters and words per sentence increases, the Coleman-Liau index gives the text a higher reading level. If you were to take this paragraph, for instance, which has longer words and sentences than either of the prior two examples, the formula would give the text an eleventh grade reading level.
Grade 11
~/ $ ./practical1
Text: A large class of computational problems involve the determination of properties of graphs, digraphs, integers, arrays of integers, finite families of finite sets, boolean formulas and elements of other countable domains.
Grade 16+
~/ $ ./practical1
Text: I can see a little seal. It has a big ball. The ball in on it's nose. It likes to play with the ball.
Before Grade 1
~/ $

```

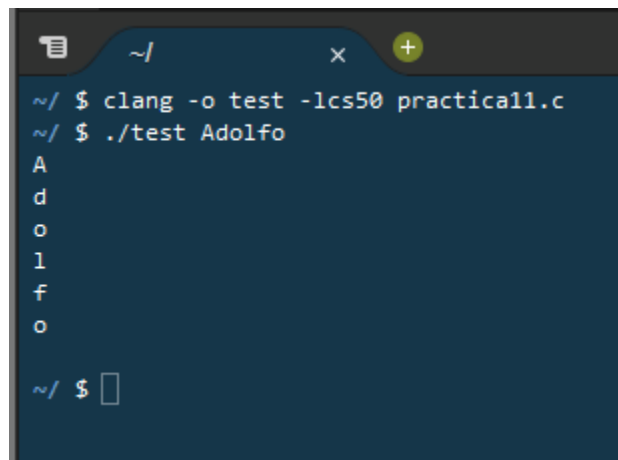
## Arreglo de 2 dimensiones



```
1 #include <cs50.h>
2 #include<stdio.h>
3 #include<string.h>
4
5 int main(int argc, string argv[])
6 {
7     for (int i = 1; i < argc; i++)
8     {
9         for (int j = 0, n = strlen(argv[i]); j < n; j++)
10         {
11             printf("%c\n", argv[i][j]);
12         }
13         printf("\n");
14     }
15 }
16
17 }
```

Con este programa declaramos argumentos dentro de la función main que tomaran esos valores al momento de ejecutarla adquiriendo lo que sea que tecleemos como arreglo y devolviéndolo en forma de columna.

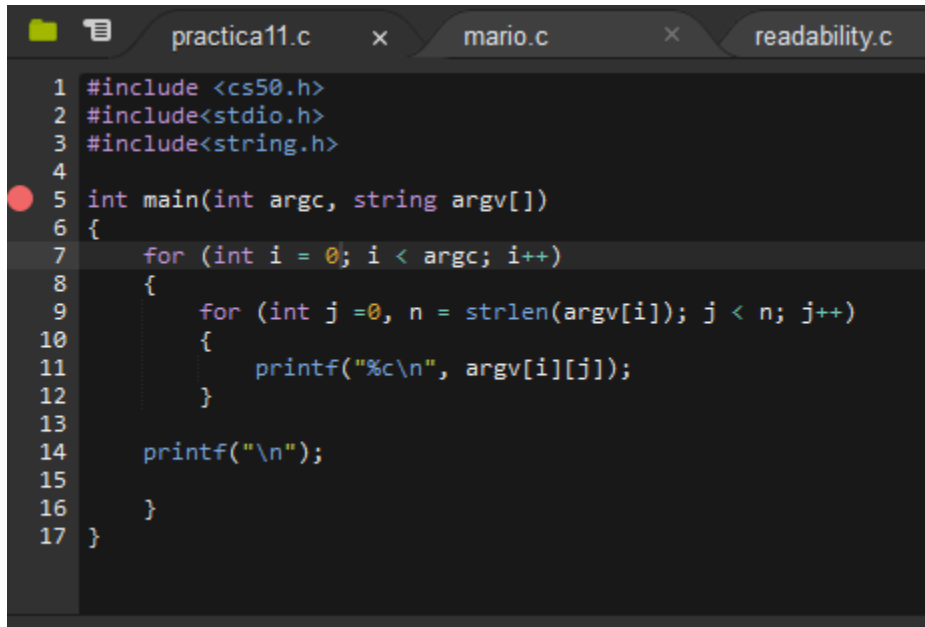
En este caso toma 2 argumentos pero solo imprimiremos el segundo a través de `argv[i][j]`



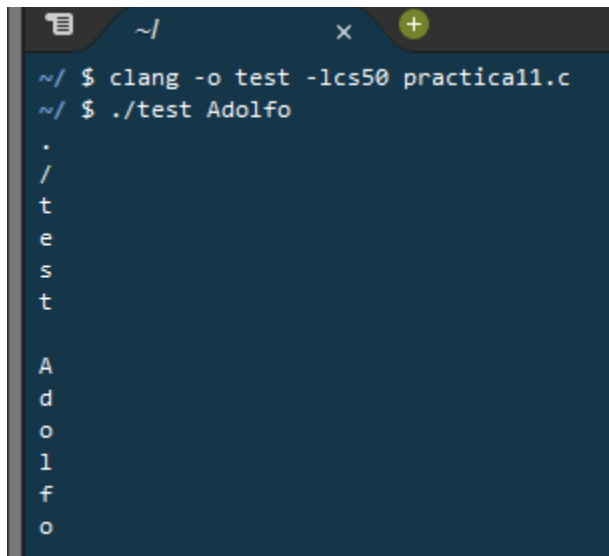
```
~/ $ clang -o test -lcs50 practica11.c
~/ $ ./test Adolfo
A
d
o
l
f
o

~/ $
```

Cuando modificamos el valor inicial del primer for-loop le decimos al programa que queremos imprimir los dos elementos que existen en el arreglo inicial, por lo que de igual forma se imprimirá el comando que corre el programa.



```
1 #include <cs50.h>
2 #include<stdio.h>
3 #include<string.h>
4
5 int main(int argc, string argv[])
6 {
7     for (int i = 0; i < argc; i++)
8     {
9         for (int j =0, n = strlen(argv[i]); j < n; j++)
10         {
11             printf("%c\n", argv[i][j]);
12         }
13     }
14     printf("\n");
15 }
16 }
17 }
```



```
~/ $ clang -o test -lcs50 practica11.c
~/ $ ./test Adolfo
.
/
t
e
s
t

A
d
o
l
f
o
```

## Conclusiones:

En estas semanas he mejorado bastante en mi entendimiento sobre como funcionan los arreglos y como funcionan sus dimensionalidades también, cosa que me sirvió muchísimo para poder organizar mucha información en mi proyecto final, que al principio no sabia realmente como la iba a ordenar pero me ha sido muy útil el saber como funcionan los arreglos.

Por otro lado, mantengo mi atención en los apuntadores ya que aun no entiendo del todo como funcionan, tengo una vaga idea aun, pero espero que durante este tiempo mi entendimiento de ellos se incremente.

## Referencias:

- Facultad de ingeniería - UNAM. Manual de prácticas del laboratorio de Fundamentos de programación: [http://odin.fi-b.unam.mx/salac/practicasFP/MADO-17\\_FP.pdf](http://odin.fi-b.unam.mx/salac/practicasFP/MADO-17_FP.pdf)

CS50 Open Course Ware, Week 2: Arrays, pset2: Readability problem: <https://cs50.harvard.edu/x/2021/psets/2/readability/>