SISTEMAS OPERATIVOS EN RED
TEMA 4

2º SMR
Profesora: Noelia Huguet Chacón
Centro de Estudios Tobalcaide
Curso 2.020 – 2.021

TABLA BIDIMENSIONAL

- Podemos declarar tablas de **dos o más dimensiones**.
- Por ejemplo, si queremos guardar datos de dos grupos de alumnos, cada uno de los cuales tiene 20 alumnos, tenemos dos opciones:
  - Podemos usar **int datosAlumnos[40]** y entonces debemos recordar que los 20 primeros datos corresponden realmente a un grupo de alumnos y los 20 siguientes a otro grupo.
  - O bien podemos emplear **int datosAlumnos[2][20**] y entonces sabemos que los datos de la forma **datosAlumnos[0][i]** son los del primer grupo, y los **datosAlumnos[1][i]** son los del segundo.
- En cualquier caso, si queremos indicar valores iniciales, lo haremos entre llaves, igual que si fuera una tabla de una única dimensión.

TABLA BIDIMENSIONAL

- Ejemplo:

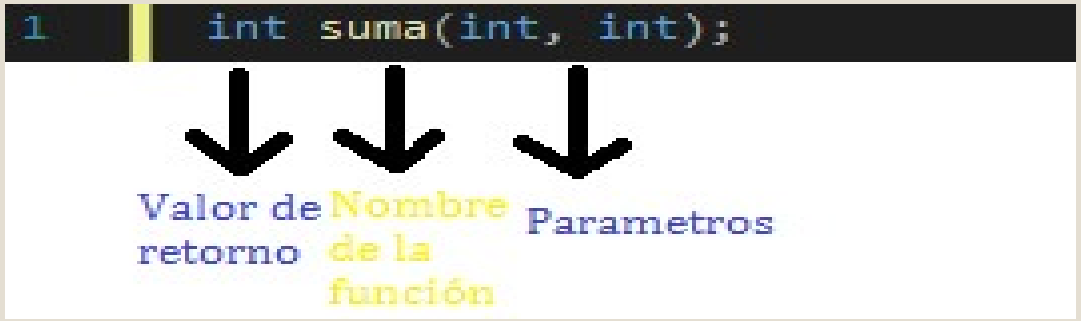
```
#include <iostream>
using namespace std;

int main()
{
    int notas[2][10] =
        {
            { 7, 2, 10, 6, 4, 6, 7, 8, 2, 1},
            {1, 4, 6, 7, 8, 2, 10, 8, 1, 5 }
        };

    cout << " La nota del tercer alumno del grupo 1 es: " << notas[0][2];
    cout << "\n La nota del quinto alumno del grupo 2 es: "<< notas[1][4];

    return 0;
}
```

DECLARACIÓN DE UNA FUNCIÓN

- Las **funciones** son código que se separa del programa y realizan una función específica.
- Una función es un bloque de código que realiza alguna operación.
- Una función puede definir opcionalmente parámetros de entrada que permiten a los llamadores pasar **argumentos** a la función.
- Una función también puede devolver un valor como salida.
- Las funciones son útiles para encapsular las operaciones comunes en un solo bloque reutilizable, idealmente con un nombre que describa claramente lo que hace la función.
- Con las funciones se consigue la **reutilización del código**.
- *Las funciones **se declaran siempre** antes de la función **main***

DECLARACIÓN DE UNA FUNCIÓN

- ***Partes de una función***
  - Una función se compone de tres partes principales
    - **Valor de retorno**: tipo que va a devolver la función. Una función puede no devolver nada, en este caso se indicará "**void**" como tipo de retorno.
    - **Nombre de la función**: nombre arbitrario que se le asigna a la función.
    - **Parámetros de la función**: tipo de los valores que se le deben proporcionar a la función. Una función puede no requerir ningún parámetro por lo que se dejará vacío (los paréntesis siempre se han de poner, aunque la lista de parámetros esté vacía).



DEFINICIÓN DE UNA FUNCIÓN

- Las funciones se definen después de la función main.
- Este es el "cuerpo" de la función, aquí es donde se escriben las instrucciones que va a realizar la función.
- Toda función debe tener al menos, un "**return (tipo)"** o "**return**" (si la función no devuelve nada) para indicar que la función ha terminado.
- Los parámetros han de tener un nombre para poder identificarlos (también se pueden nombrar en la declaración, pero es irrelevante para el compilador)
- Se puede invocar la función o *llamarla* desde cualquier número de lugares del programa.
- Los valores que se pasan a la función son los *argumentos*, cuyos tipos deben ser compatibles con los tipos de parámetro de la definición de función.

DEFINICIÓN DE UNA FUNCIÓN

```
int sum(int a, int b)
{
    return a + b;
}

int main()
{
    int i = sum(10, 32);
    int j = sum(i, 66);
    cout << "La suma de 10+32+66 es " << j;
}
```

```
void sum(int a, int b)
{
    int resultado;
    resultado=a+b;
    cout<<"La suma es "<< resultado;
    return;
}
int main()
{
    sum(10, 32);
    return 0;
}
```

FUNCIONES RECURSIVAS

- Una función es recursiva cuando entre sus líneas realiza una llamada o invocación a sí misma.
- Se pueden distinguir dos tipos de recursividad:
  - **Recursividad directa:** Es aquella en la que la función realiza una llamada a sí misma desde un punto concreto entre sus llaves de inicio y fin.
  - **Recursividad indirecta:** Es aquella en la que la función realiza una invocación a otra función que es quien llama a la primera.
- Todos los problemas recursivos presentan una estructura similar, contemplan:
  - Un **caso base** que permite la finalización del problema.
  - **Casos recursivos** que hacen que la función vuelva a ser ejecutada.

FUNCIONES RECURSIVAS

- **Ejemplo**:

Vamos a calcular el **factorial** de un número.

- El factorial de 5, por ejemplo es:

  5!=5*(5-1)*(5-2)*(5-3)*(5-4)

- En forma general el factorial de un número es:

  n! = n * (n-1)*(n-2)*(n-3)*…*1

- Crearemos una función para calcular el factorial de un número que la llamaremos:

  int factorial (int numero)

FUNCIONES RECURSIVAS

Factorial de un número → n! = n * (n-1)*(n-2)*(n-3)*…*1

¿Cuál podría ser el caso base?

- El caso base finaliza el programa, de forma que teniendo en cuenta esto, ¿cuándo finaliza el factorial? El factorial acaba cuando llega a 1, por lo que si la función tiene que calcular el factorial de 1, este sería 1.

  1! = 1

¿Cuál podría ser el caso recursivo?

- Se puede observar que el cálculo del factorial es multiplicar continuamente un número por su inmediatamente anterior, de forma que se puede decir que el factorial de un número podría expresarse como:

  n! = n * (n-1)! o lo que es lo mismo para nuestra función:

  factorial (n) = n * factorial (n-1);

FUNCIONES RECURSIVAS

Factorial de un número → n! = n * (n-1)*(n-2)*(n-3)*…*1

- **CASO BASE:**

  Si n =1, factorial (1) devuelve 1;

- **CASO RECURSIVO:**

  Si n>1, factorial (n) = n * factorial (n-1);

```
Función factorial (entero n) retorna entero{
    si n>1 entonces
        //Caso recursivo devuelve n*factorial (n-1);
    sino
        devuelve 1; //Caso base
    fin si;
}
```

```
int factorial (int n)
{
    if(n>1)
        return n* factorial(n-1);
    else
        return 1;
}
```

CADENA DE CARACTERES

- La biblioteca a utilizar será **<string.h>**
- Una **cadena en C++** es un conjunto de caracteres, o valores de tipo *char*, terminados con el carácter nulo, es decir el valor numérico 0.
- Internamente, en el ordenador, se almacenan en posiciones consecutivas de memoria.
- Este tipo de estructuras recibe un tratamiento muy especial, ya que es de gran utilidad y su uso es continuo.
- La manera de definir una cadena es la siguiente:

char <identificador> [<longitud máxima>]

- Cuando se declara una cadena hay que tener en cuenta que tendremos que reservar una posición para almacenar el carácter nulo terminador.

CADENA DE CARACTERES

- Si queremos almacenar la cadena "HOLA", tendremos que declarar la cadena como:

char Saludo[5];

- Las cuatro primeras posiciones se usan para almacenar los caracteres "HOLA" y la posición extra, para el carácter nulo.
- También nos será posible hacer referencia a cada uno de los caracteres individuales que componen la cadena, simplemente indicando la posición.
- Por ejemplo el tercer carácter de nuestra cadena de ejemplo será la 'L', podemos hacer referencia a él como Saludo[2].
- La asignación directa sólo está permitida cuando se hace junto con la declaración.

```
//La forma correcta de declaración

char Saludo[5]  = “HOLA”;
```

```
//La forma incorrecta de declaración

char Saludo[5] ;
Saludo = “HOLA”;
```

```
//Otra forma correcta de declaración

char Saludo[5];
Saludo[0]  =  ‘H’;
Saludo[1]  =  ‘O’;
Saludo[2]  =  ‘L’;
Saludo[3]  =  ‘A’;
Saludo[4]  = 0;
```

CADENA DE CARACTERES

- Para introducir una cadena de caracteres por el usuario

```
// con cin
// Solo coge lo escrito antes del primer espacio

        char nombre [15];

        cout <<"Introduce tu nombre";
        cin>>nombre;

        cout<<"Mi nombre es: "<< nombre<<endl;
```

```
Introduce tu nombre: Noelia Huguet Chacon
Mi nombre es Noelia
```

```
// con gets
// Solo coge todo lo escrito aunque supere el tamaño preestablecido.

        char nombre [15];

        cout <<"Introduce tu nombre";
        gets(nombre);

        cout<<"Mi nombre es: "<< nombre<<endl;
```

```
Introduce tu nombre: Noelia Huguet Chacon
Mi nombre es Noelia Huguet Chacon
```

CADENA DE CARACTERES

- Para introducir una cadena de caracteres por el usuario

```
// con cin
// Solo aunque se escriba más de lo establecido coge hasta el tamaño indicado
// o el final que se indica en la instrucción

      char nombre [15];

      cout <<"Introduce tu nombre";
      cin.getline(nombre,15,'\n');

      cout<<"Mi nombre es: "<< nombre<<endl;
```

```
Introduce tu nombre: Noelia Huguet Chacon
Mi nombre es Noelia Huguet C
```

cin.getline(donde se asigna la cadena, tamaño, final)

CADENA DE CARACTERES

La biblioteca a utilizar es <string.h>

- **strlen(cadena)** = Longitud de una cadena de caracteres
- **strcpy (cadena1, cadena2)** = Copiar el contenido de la cadena 2 a la cadena 1.
- **strcmp(cadena1, cadena2)** = comparar dos cadenas. Si son iguales devuelve un 0.

También sabe que cadena es mayor que la otra alfabéticamente (a es menor que b).

- **strcat(cadena1,cadena2)** = concatena dos cadena de caracteres.
- **strrev()** = invierte una cadena.
- **strupr()** = pasar una cadena a mayúsculas.
- **strlwr()** = pasar una cadena a minúsculas.

```
char cad1[] = "Esto en un";
char cad2[] = " ejemplo";
char cad3[30];

strcpy(cad3, cad1); //cad3 = "Esto es un"

strcat(cad3, cad2); // cad3 = "Esto es un ejemplo"
```

CADENA DE CARACTERES

La biblioteca a utilizar es <string.h>

- **aoti(cadena)** = transforma una cadena a números enteros.
- **atof cadena1)** = transforma una cadena a números reales.

```
char cad1[] = "1234"; // esto es una cadena que contiene números pero no se puede utilizar para operar.
int num1;

char cad2[] = " 98,76"; // esto es una cadena que contiene números pero no se puede utilizar para operar.
float num2;

num1 = atoi(cad1); //num1 ya se puede utilizar para hacer operaciones
num2 = atof(cad2); //num2 ya se puede utilizar para hacer operaciones
```