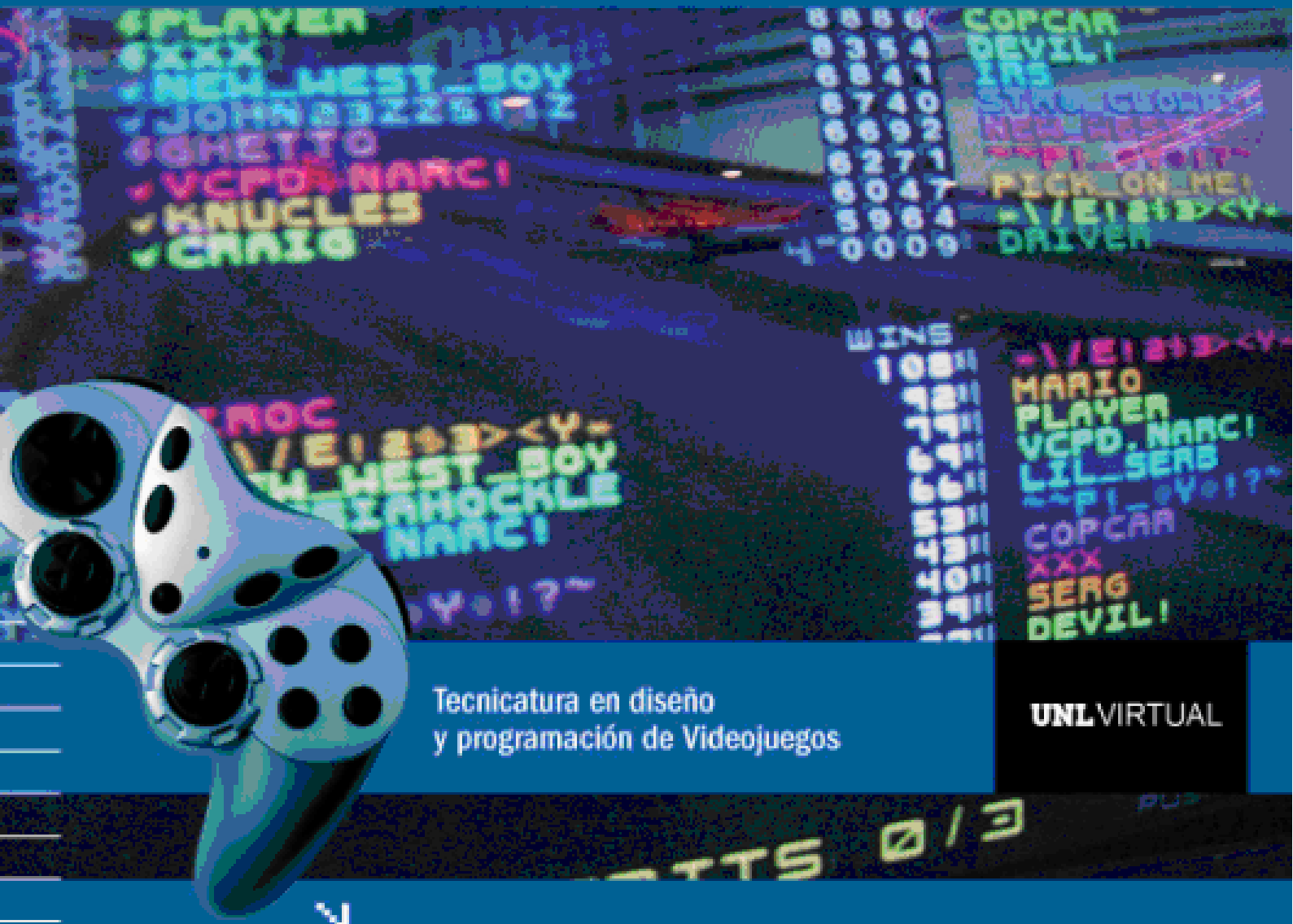
UNIVERSIDAD NACIONAL DEL LITORAL
FACULTAD DE INGENIERÍA Y CIENCIAS HÍDRICAS



Introducción a la programación

Unidad Temática Número 5

Arrays y Structs

Objetivo: introducir la estructura de datos arreglo para comprender las técnicas básicas para la solución de problemas como para almacenar, ordenar y buscar datos.

Temas: Arreglos. Declaración de arreglos. Como pasar arreglos a funciones. Arreglos con múltiples subíndices.

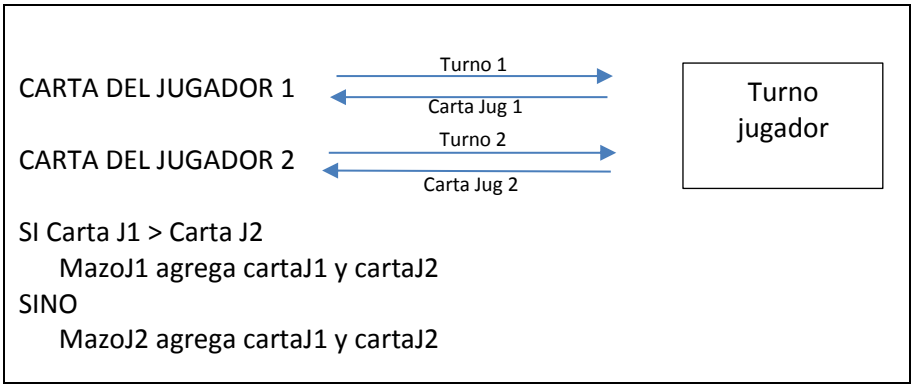
INTRODUCCIÓN

Imaginemos un juego de cartas muy sencillo, de dos jugadores. Cada jugador con su propio mazo, el primero toma una carta y la muestra. El segundo repite lo mismo con su mazo. Aquél que tenga la carta más alta, se lleva ambas y repiten hasta que se alguno se lleve todas las cartas.

Sin meternos en detalle de la implementación podemos imaginar que las acciones que realice el jugador uno y el jugador dos, van a ser muy similares sino idénticas: definir un mazo, tomar una carta al azar, verificar que esa carta no haya salido ya, quitar la carta del mazo. Una vez que tenemos las dos cartas, comparamos y agregamos esas dos cartas al mazo del jugador que ganó. En la Unidad Uno, cuando definimos a los Procedimientos como conjuntos de primitivas que nos servían para ahorrar escribir varias veces lo mismo. En este caso podríamos decir que este conjunto de primitivas que nombramos anteriormente podrían constituir un procedimiento, quizás llamado "Turno jugador" y lo llamaríamos dos veces, primero para el jugador uno y luego para el dos, recién una vez que tenemos la carta de ambos seguimos con el resto del algoritmo y volvemos a empezar.

El procedimiento es una subrutina, es decir, una porción del algoritmo principal que se encarga de resolver una tarea específica, y es un acercamiento al concepto que veremos en esta Unidad. Función. Veamos si podemos ampliar un poco más el concepto. Tenemos nuestro procedimiento llamado "Turno jugador", lo que se realice dentro de este procedimiento será general, tanto para el jugador Uno como para el jugador Dos o incluso la cantidad de jugadores que quieran jugar. El objetivo del procedimiento es darnos una carta "útil", es decir, una carta que haya sido verificada que esté dentro del mazo y que no se haya sacado antes, sea del jugador que sea, es decir, la carta siempre va a corresponder al jugador del cual corresponde el turno. De alguna forma deberíamos indicarle al procedimiento cuál es el jugador al cual pertenece el turno y "avisarle" para que pueda tomar las variables correctas, y devolvernos la carta correcta.

Si pudiéramos graficar el procedimiento como una caja que hace todo eso que dijimos antes y qué sólo nos interesa que nos dé una carta útil a partir de propinarle el turno del jugador, podríamos imaginar algo así:



Tenemos entonces el concepto de valor de retorno y valor de ingreso. Podemos decir, ahora sí, que "Turno jugador" es una función. Definimos función como un subalgoritmo encargado de realizar una tarea específica y general dentro del algoritmo, puede recibir un conjunto de parámetros y retornar un valor que se relaciona con su tarea y los valores de entrada.

En la vida cotidiana conocemos varias funciones, básicamente cualquier sistema que requiera una configuración, una la alarma, una radio, una calculadora, a la cual le ingresamos una serie de valores, una operación y nos da el valor de esa operación. En matemática conocemos las funciones trigonométricas. Las funciones trigonométricas son funciones que relacionan los ángulos con los lados del triángulo rectángulo formado en la circunferencia, cada relación recibe un nombre: seno, coseno y tangente. Cuando las usamos no nos interesa recordar cuál es la relación que se está operando, simplemente insertamos valor de ángulo y esperamos un valor de resultado. Todos esos conceptos corresponden al de función y son análogos en la programación.

Estamos en condiciones que definir, entonces, funciones en C++.

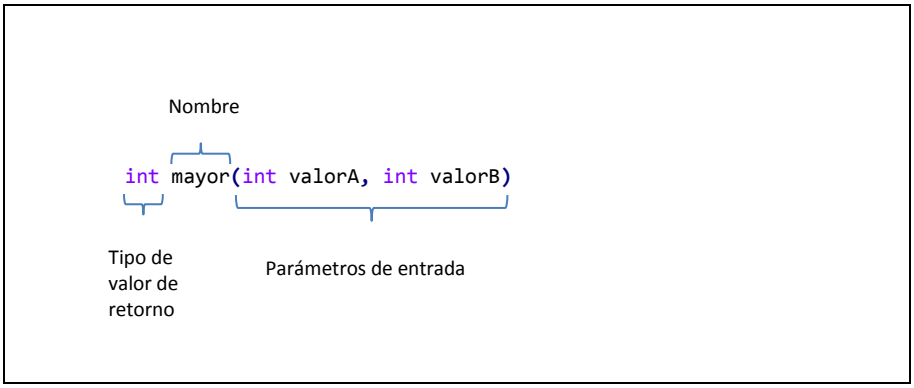
FUNCIONES

Las funciones poseen tres elementos esenciales, el nombre, los parámetros de entrada y el valor de retorno. Estos últimos dos pueden no estar presentes, dependiendo la función. Vamos a poner el ejemplo de una función que nos indique el mayor de dos valores.

```
int mayor(int valorA, int valorB){

    if (valorA > valorB){
        return valorA;
    }
    else{
        return valorB;
    }
}
```

En la primera línea encontramos la definición de la función:



Lo primero que encontramos es el tipo de valor de retorno de la función (resultado), en este caso la función devolverá un valor del tipo entero que casualmente coincide con alguno de los valores que ingresan. Lo segundo es el nombre, los nombres de las funciones tienen las mismas reglas que el de las variables. Luego, entre paréntesis, encontramos los parámetros de entrada, o sea, los valores que permiten configurar nuestra función, en este caso son vos valores enteros, los nombres que utilicemos en la definición de los parámetros deberán ser los mismos que utilizaremos dentro de la función. Luego de la definición abrimos las llaves y realizamos la implementación, aquí el ámbito dentro de la función es un ámbito nuevo, es decir, lo que declaremos dentro de la función existirá sólo en ese ámbito, además de los parámetros de ingreso.
La función deberá retornar como resultado un valor del mismo tipo del especificado en la definición, eso se realiza mediante la palabra reservada **return**. El return debe estar en el ámbito mayor de la función, en caso contrario, debe haber uno por cada ámbito de la siguiente jerarquía, etc. Es decir, no puede haber camino en la función que pueda finalizar la función sin tener un return (el compilador detecta esto como un error).
Ahora que ya tenemos la función la vamos a utilizar en nuestro algoritmo.
Las funciones siempre van fuera del ámbito del main (que por cierto, es una función).

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

int mayor(int valorA, int valorB){

    if (valorA > valorB){
        return valorA;
    }
    else{
        return valorB;
    }
}
```

```
int main (int argc, char *argv[]) {

    srand (time(NULL));

    int val1 = rand()%100;
    int val2 = rand()%100;

    cout<<"el valor 1 es "<<val1<<endl;
    cout<<"el valor 2 es "<<val2<<endl;

    int May = mayor(val1,val2);
    cout<<"el mayor entre ambos es "<<May<<endl;

    return 0;
}
```

Esto me muestra en consola lo siguiente:

```
el valor 1 es 41
el valor 2 es 62
el mayor entre ambos es 62
```

A continuación veremos en detalle cada uno de los conceptos que acabamos de nombrar.

RETORNO

El retorno de una función es el resultado que esta genera. El retorno, en la definición, se coloca a la izquierda del nombre, se coloca el tipo de dato de retorno, como la declaración de cualquier variable. Una función puede retornar cualquier tipo de dato: int, float, char, objetos, arreglos, etc. Pero siempre un único dato (no se pueden retornar dos valores). Si deseáramos retornar dos valores podríamos usar un arreglo o alguna función por referencia (ya veremos eso más adelante). Dentro de la declaración el retorno se especifica mediante la palabra reservada return.

```
// funcion de promedio de 3 valores

float promedio3(int valor1, int valor2, int valor3){

    float sum = valor1 + valor2 + valor3;
    sum = sum/3;
    return sum;  // retorno de la función (float)
}
```

En caso de no necesitar retornar nada, el tipo de retorno es la palabra reservada **void**. Las funciones que no retornan valor no son muy comunes, por lo general son aquellas relacionadas a variables del sistema. En caso de una función sin retorno, no hace falta (ni idebe) colocarse el **return**.

```
// funcion sin retorno
// muestra en pantalla el string pasado por parámetro

void SalidaEnPantalla(string salida){
    cout<<salida<<endl;

}
```

Luego, a la hora de llamar a la funciones desde nuestro programa, debemos respetar el tipo de dato de retorno. Las funciones pueden usarse como cualquier variable en las expresiones, para ello debemos guiarnos por el valor de retorno.

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

// funcion de promedio de 3 valores
float promedio3(int valor1, int valor2, int valor3){

    float sum = valor1 + valor2 + valor3;
    sum = sum/3;
    return sum;
}

void SalidaEnPantalla(string salida){
    cout<<salida<<endl;
}


int main (int argc, char *argv[]) {

    srand (time(NULL));

    int val1 = rand()%100;
    int val2 = rand()%100;
    int val3 = rand()%100;

    SalidaEnPantalla("El primer valor es");
    cout<<val1<<endl;
    SalidaEnPantalla("El segundo valor es");
    cout<<val2<<endl;
    SalidaEnPantalla("El tercer valor es");
    cout<<val3<<endl;
    SalidaEnPantalla("El valor de promedio de los datos es");
    cout<<promedio3(val1,val2,val3)<<endl;

    return 0;
}
```

Salida:

```
El primer valor es
52
El segundo valor es
25
El tercer valor es
35
El valor de promedio de los datos es
37.3333
```

NOMBRE

El nombre de la función sigue las mismas reglas que el de las variables. Es muy importante que el nombre de la función de un indicio de su acción, para que no sólo sea indicativo sino que sea coherente en el código. Si bien hay reglas generales y algunas específicas, dependiendo el grupo de trabajo, las más importantes son:

- No mezclar idiomas, o todo español o todo inglés
- No usar gerundio (sumando, quitando, ordenando)
- No hacer referencia al funcionamiento interno
- Para el caso de las banderas, siempre es útil iniciar con el estado que se consulta: esPar, esMayor, perteneceAlMazo

PARAMETROS

Los parámetros son aquellos valores que necesita la función como datos de entrada para operar. Se colocan entre paréntesis, separados por comas. Al momento de la definición (cuando se escribe la función) debe colocarse el tipo, a la hora de llamar a la función, sólo se colocan las variables sin su tipo. Debe respetarse el orden. Una función puede aceptar cualquier tipo de parámetros, int, float, string, arreglos, objetos, etc. La cantidad de parámetros puede ser desde cero (por ejemplo la función rand) a N.

```
// funcion que multiplica o divide segun bandera
float multiplicaOdivide(bool operacion, int valor1, int valor2){

    if (operacion){ // si true multiplica
        return (valor1*valor2);
    }else{  //sino divide
        return (valor1/valor2);
    }
}
```

Luego al llamarla desde el main debemos respectar el orden y el tipo de parámetros. La función **multiplicaOdivide** exige primero un booleano que define la operación y luego dos enteros donde el orden también importa. Puede que para la multiplicación sea indistinto alterar los factores, pero no lo es para la división.

```
    int val1 = rand()%100;
    int val2 = rand()%100;
    bool operacion = false;

    cout<< multiplicaOdivide(operacion,val1,val2);
```

PARAMETROS: ARREGLOS

Para pasar un arreglo unidimensional como parámetro de una función pasamos el arreglo sin especificar la cantidad de elementos.

Una función de promedio sería así:

```
float promedio(int valores[], int cant){

    float sum = 0;
    for (int i = 0; i< cant; i++){
        sum+=valores[i];
    }
    return (sum/cant);
}
```

Si en cambio tenemos más dimensiones en el arreglo, a partir de la segunda dimensión, debemos especificar la cantidad de elementos.

```
float promedio(int valores[][4], int cant){

    float sum = 0;
    for (int i = 0; i< cant; i++){
        for (int j = 0; j<4;j++){
            sum+=valores[i][j];
        }
    }
    return (sum/cant);
}
```

A la hora de llamar a estas funciones debemos hacerlo mediante las variables tal cual su nombre.

```
    cout<< promedio(miArreglo,10);
```

PARAMETROS: VALORES POR DEFECTO

En C++ existe la posibilidad de asignar valores por defecto a los parámetros, de esta manera, en caso de no pasarle valores a esos parámetros, se toman los valores por defecto. Para ello en la definición de la función debemos asignarle un valor en la misma definición por parámetros.

```
// funcion que permite
// sumar (1), restar (2), multiplicar (3) o dividir (4)
// dos valores, por defecto suma (operacion = 1)

float operar(int valor1, int valor2, int operacion = 1){

    switch(operacion){
        case 1: return valor1+valor2; break;
        case 2: return valor1-valor2; break;
        case 3: return valor1*valor2; break;
        case 4: return valor1/valor2; break;
    }
}

int main (int argc, char *argv[]) {


    srand(time(NULL));

    int v1 = rand()%10;
    int v2 = rand()%10;

    // Si no pasamos un tercer parámetro, toma el 1 por defecto

    cout<<"Suma: "<<operar(v1,v2)<<endl;
    return 0;
}
```

En caso de tener más de un parámetro que permita valores por defecto. Todos ellos deben ir al final, por ejemplo:

```
// 2do y 3er parámetro con valores por defecto

float operar(int valor1, int valor2 = 10, int operacion = 1)
```

No podemos intercalar valores por defecto de parámetros normales. Lo siguiente es incorrecto:

```
// 2do parámetro por defecto

float operar(int valor1, int valor2 = 10, int operacion)
```

No es posible asignar un parámetro por defecto entre dos parámetros normales ya que si llamamos a la función con dos parámetros es imposible para el compilador distinguir a qué parámetro corresponde cada valor.

PARAMETROS: PASAJE POR REFERENCIA

Hasta ahora, todos los ejemplos de parámetros que vimos son pasados por valor, esto quiere decir que cuando nosotros ponemos una variable como parámetro de una función, esta no se modificará al concluirse la función. Supongamos la función alCuadrado, que eleva un valor al cuadrado:

```
// funcion que eleva al cuadrado el parametro
int alCuadrado(int valor){
    valor = valor*valor;
    return valor;
}
```

Cuando la llamemos con algún valor, el valor de la variable (salvo que le reasignemos un valor) no será modificado.

```
    int v = 10;
    cout << "v vale: "<<v<<endl;
    int v2 = alCuadrado(v);
    cout << "después de la funcion: "<<endl;
    cout << "v2 vale: "<<v2<<endl;
    cout << "v vale: "<<v<<endl;
```

En el ejemplo usamos **v** como variable para pasar de parámetro a la función **alCuadrado**. Dentro de la función, el parámetro modifica su valor (a **valor** se le asigna su cuadrado), aunque esto no impacta en la variable que se usó para llamar a la función. Si vemos por consola el valor de **v** antes y después de llamar a la función, éste será el mismo.

```
v vale: 10
despues de la funcion:
v2 vale: 100
v vale: 10
```

Esta forma de usar los parámetros se llama pasaje "por valor", que hace le compilador es crear una copia de la variable para ser usada dentro de la función, de esta manera no afecta el valor original de la variable.

Hay también otra forma de pasar parámetros que se llama pasaje "por referencia". En el pasaje por referencia el valor que tome el parámetro dentro de la función impactará en la variable que se utilizó para llamar a la función. Para pasar un parámetro y que sea tomado por referencia, debemos anteponerle el operador ampersand al nombre del parámetro, el resto queda igual.

De esta manera, para la misma función:

```
// funcion que eleva al cuadrado el parametro
int alCuadrado(int &valor){
    valor = valor*valor;
    return valor;
}
```

A la hora de llamar a la función realizamos el mismo procedimiento:

```
    int v = 10;
    cout << "v vale: "<<v<<endl;
    int v2 = alCuadrado(v);
    cout << "después de la funcion: "<<endl;
    cout << "v2 vale: "<<v2<<endl;
    cout << "v vale: "<<v<<endl;
```

Pero esta vez veremos que el valor de **v** sí se ve afectado por lo que se realiza dentro de la función.

```
v vale: 10
despues de la funcion:
v2 vale: 100
v vale: 100
```

Podamos optar por pasar por referencia cualquier valor que queramos, no importa si es uno o todos o si alternamos entre parámetros por valor y por referencia.

```
int sumados(int &valor1,int valor2,int &valor3,int valor4){
    valor1 = valor1+1;
    valor2 = valor2+2;
    valor3 = valor3+3;
    valor4 = valor4+4;

    return (valor1+valor2+valor3+valor4);
}
```

Aquí tenemos al primer y al tercer parámetro pasado por referencia, mientras que el segundo y el cuarto se pasan por valor. A cada parámetro se le suma 1, 2, 3 y 4 respectivamente. A la hora de llamar a la función esperaríamos que las variables usadas en el primer y tercer parámetro, modifiquen su valor fuera del ámbito de la función.

```
int main (int argc, char *argv[]) {

    int v1 = 10;
    int v2 = 20;
    int v3 = 30;
    int v4 = 40;

    cout << "v1 vale: "<<v1<<endl;
    cout << "v2 vale: "<<v2<<endl;
    cout << "v3 vale: "<<v3<<endl;
    cout << "v4 vale: "<<v4<<endl;
    int total = sumados(v1,v2,v3,v4);
    cout << "despues de la funcion: "<<endl;
    cout << "v1 vale: "<<v1<<endl;
    cout << "v2 vale: "<<v2<<endl;
    cout << "v3 vale: "<<v3<<endl;
    cout << "v4 vale: "<<v4<<endl;
    cout << "total: "<<total<<endl;


    return 0;
}
```

Este código nos muestra la siguiente salida por consola:

```
v1 vale: 10
v2 vale: 20
v3 vale: 30
v4 vale: 40
despues de la funcion:
v1 vale: 11
v2 vale: 20
v3 vale: 33
v4 vale: 40
total: 110
```

Esta forma de utilizar los parámetros nos puede dar otras herramientas para trabajar con los retornos, aunque hay que tener cuidado de no modificar variables en nuestro código que no queremos modificar.

SOBRECARGA DE FUNCIONES

C++ nos permite escribir varias funciones con el mismo nombre siempre y cuando las podamos identificar de alguna manera. Esta manera es difiriendo en la cantidad y/o tipos de parámetros utilizados.
Imaginemos tres funciones que calculan el promedio y las tres se llaman de la misma manera. La primera recibe por parámetro 4 elementos enteros, la segunda recibe 3 elementos enteros, la tercera también recibe 3 elementos, pero esta vez son flotantes. Estas diferencias le permiten al compilador poder identificarlas y saber a cuál se hace referencia.

```
// 4 parametros enteros
float promedio(int valor1,int valor2,int valor3,int valor4){

    return ((valor1+valor2+valor3+valor4)/4);
}

// 3 parametros enteros
float promedio(int valor1,int valor2,int valor3){

    return ((valor1+valor2+valor3)/3);
}

// 3 parametros flotantes
float promedio(float valor1,float valor2,float valor3){

    return ((valor1+valor2+valor3)/3);
}
```

A la hora de llamarla podemos hacerlo como ya sabemos.

```
    int v1 = 10;
    int v2 = 20;
    int v3 = 30;
    int v4 = 40;
    int v5 = 10;
    int v6 = 20;
    int v7 = 30;

    cout << "promedio 1: "<< promedio(v1,v2,v3,v4) <<endl;
    cout << "promedio 2: "<< promedio(v1,v2,v3) <<endl;
    cout << "promedio 3: "<< promedio(v5,v6,v7) <<endl;
```

La salida en consola:

```
promedio 1: 25
promedio 2: 20
promedio 3: 20
```

La posibilidad que nos ofrece la sobrecarga es la de darle más trasparencia al usuario (el desarrollador que va a usar las funciones) para no tener que recordar cual función es la que debe utilizar dependiendo las variables con las que está trabajando.

CABECERAS Y DECLARACIONES

Desde que comenzamos esta unidad en ningún momento mencionamos dónde se colocan las funciones, sólo dijimos que van fuera del main. Y en los ejemplos venimos colocándolas en la parte superior. La realidad es que por convención y facilidad de lectura, se colocan por debajo del código principal, al final de todo. El motivo es que cuando uno lee o trabaja en un código espera hacerlo desde el hilo principal (el main en nuestro caso) y de ahí ver cuáles son las funciones que se usan, en caso de necesitar ver cómo trabaja o qué retorna cierta función en particular, se puede ir a ver su definición, pero en general con el nombre de la función debería bastar para darnos una idea. ¡Es por eso que es muy importante poner buenos nombres! Sin embargo hay un detalle que nos impide colocar una función al principio así nomás, y es que el compilador al leer el código (de arriba hacia abajo) leerá una función de la cual no recibió ningún tipo de información y, lejos de saber que tiene su definición más abajo en el código, cortará su compilación y dará un mensaje que no puede compilar porque la función no está definida. Esto se soluciona de una manera muy sencilla, colocando la cabecera de la función en la parte superior y su implementación en la inferior.

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

// Cabecera de la funcion Promedio. solo van sus componentes
float promedio(int valor1,int valor2,int valor3,int valor4);

int main (int argc, char *argv[]) {

    int v1 = 10;
    int v2 = 20;
    int v3 = 30;
    int v4 = 40;

    cout << "promedio "<< promedio(v1,v2,v3,v4) <<endl;

    return 0;
}

// Implementacion de la funcion promedio
// Ahora si ya colocamos la funcion entera con su implementacion
float promedio(int valor1,int valor2,int valor3,int valor4){

    return ((valor1+valor2+valor3+valor4)/4);
}
```

En la parte superior, por encima de main ponemos todas las cabeceras de las funciones, que no es más que sus componentes principales: retorno, nombre y parámetros, finalizando la línea con un punto y coma (;).

```
float promedio(int valor1,int valor2,int valor3,int valor4);
```

Luego, a continuación del main o nuestro hilo principal, colocamos todas las declaraciones de las funciones que definimos arriba.