Implementación de Métodos en C++

Definición

Un *método* o también llamado *función*, es una rutina de código que se encarga de realizar una tarea. Tal rutina se encuentra definida o escrita dentro de un *bloque* identificado con un nombre.

Este *bloque* esta delimitado por una llave que abre ({) y una llave que cierra (})

Declaración de un método (1)

Sintaxis:

***public*: método accesible por cualquier otra clase**
***private*: método solo accesible por su propia clase**
***protected*: método accesible por cualquier otra clase que se encuentre en el mismo paquete**

void* = si el método no retorna nada, o tipos primitivos de datos (*bool, int, double, char*), u objetos de una clase (*string, Estudiante, Vendedor*). También *arreglos.

```
Modificador_acceso Tipo_dato_retorna nombreMetodo (arg1, arg2, arg3, ...)
{
   rutina de código
   return variable_o_valor;
}
```

Opcional

Si el método retorna algo, se debe escribir la palabra *return* seguido de una variable o valor.

Identificador o nombre del método.

Generalmente todo identificador se escribe en minúsculas, y si son mas de dos palabras, la inicial de cada palabra en mayúscula. (Notación CamelCase)

Opcional

Lista de argumentos.

Declaración de un método (2)

Ejemplo #1:

Escribir un método en C++ que lea la edad de una persona y determine si es mayor o menor de edad.

Tipo de retorno. ***void*** el método no retorna nada

Identificador del método. Cumple con el estándar Camel Case

```
void leerEdad() {
    int edad;
    cout<<"Ingrese la edad:";
    cin>>edad;

    if (edad > 17)
        cout<<"Es mayor de edad"<<endl;
    else
        cout<<"Es menor de edad"<<endl;
}
```

Como el método es ***void***, no aparece le palabra ***return***

Ver **Ejemplo1.cpp**

Métodos que reciben valores (1)

Es muy común que se necesite implementar métodos que reciban una serie de valores, que sirven como *alimento* para la función, y con estos valores realizar cualquier tipo de operación.

Cuando un *método* o *función* recibe valores, se dice que el método tiene *lista de argumentos*.

Métodos que reciben valores (2)

Ejemplo #2:

Escribir un método en C++ que reciba la edad de una persona y determine si es mayor o menor de edad.

```
void leerEdad(){
    int edad;
    cout<<"Ingrese la edad:";
    cin>>edad;
    esMayorEdad(edad);
}
```

El método ***leerEdad()*** llama al método ***esMayorEdad(int)***, y le envía la variable **int** edad

El método recibe como argumento un valor de tipo **int**

```
void esMayorEdad(int edad) {
    if (edad > 17)
        cout<<"Es mayor de edad"<<endl;
    else
        cout<<"Es menor de edad"<<endl;
}
```

Ver **Ejemplo2.cpp**

Métodos que reciben valores (3)

Ejemplo #3:

Escribir un método en C++ que reciba como argumento, tres valores de tipo *double* que representan las tres notas obtenidas por un estudiante. El método debe calcular la nota definitiva.

```
void leerNotas(){
    double parcial1, parcial2, parcialFinal;
    cout<<"Ingrese parcial 1: ";
    cin>>parcial1;
    cout<<"Ingrese parcial 2: ";
    cin>>parcial2;
    cout<<"Ingrese parcial final: ";
    cin>>parcialFinal;

    notaDefinitiva(parcial1, parcial2, parcialFinal);
}

void notaDefinitiva(double p1, double p2, double pf) {
    double def = (p1 * 0.3) + (p2 * 0.3) + (pf * 0.4);
    cout<<"Definitiva: "<<def<<endl;
}
```

Ver **Ejemplo3.cpp**

Métodos que retornan valores (1)

En los ejemplos anteriores, los métodos reciben argumentos y los procesan, pero estos no retornan ningún tipo de valor, es decir son ***void***.

Cuando se requiere que un método retorne un valor, se debe especificar en la declaración del método el tipo de valor de retorno y añadir la palabra ***return*** dentro del método. Cabe anotar que un método solo puede retornar un valor.

En C++ existen muchos métodos con estas características. Por ejemplo:

```
double num = rand();
```

Calcula un número aleatorio entre 0.0 y 0.9, y lo retorna

Métodos que retornan valores (2)

Ejemplo #4:

Escribir un método en C++ que retorne la sumatoria de los elementos de un arreglo de tipo int.

```
long calcularSuma(){
    long suma = 0;
    for(int i = 0; i < tam; i++){
        suma+=arreglo[i];
    }
    return suma;
}

void mostrarSuma(){
    long sum = calcularSuma();
    cout<<"La sumatoria de los elementos es: "<<sum<<endl;
}
```

Ver **Ejemplo4.cpp**

Métodos que retornan valores (3)

Ejemplo #5:

Escribir un método en C++ que obtenga la fecha actual del sistema, y la retorne como una cadena con el formato DD/MM/AAAA.

```
string getFechaActual(){
    string fecha = "";
    time_t t = time(0); // get time now
    tm* now = localtime(&t);
    int dia = now->tm_mday;
    int mes = now->tm_mon + 1;
    int year = now->tm_year + 1900;

    if(dia < 10) fecha+="0" + to_string(dia) + "/";
    else fecha+=to_string(dia) + "/";

    if(mes < 10) fecha+="0" + to_string(mes) + "/";
    else fecha+=to_string(mes) + "/";

    fecha+=to_string(year);

    return fecha;
}
```

Ver **Ejemplo5.cpp**

Métodos que retornan valores (4)

Ejemplo #6:

Escribir un método en C++ que calcule tres valores reales aleatorios, obtenga el mayor y lo retorne.

```
double getMayor(){
    srand(time(0));

    double num1 = rand();
    double num2 = rand();
    double num3 = rand();

    cout<<num1<<" "<<num2<<" "<<num3<<endl;

    double mayor = max(num1, max(num2, num3));
    return mayor;
}
```

Ver **Ejemplo6.cpp**

Métodos que retornan valores (5)

Ejemplo #7:

Escribir un método en C++ que obtenga y retorne el menor y el mayor valor almacenado en un arreglo de tipo long.

Solución: como un método solo puede retornar ***un solo valor***, una solución seria retornar un arreglo con dos posiciones. En la posición cero se almacenara el menor valor, y en la posición 1 el mayor.

```
void mostrarMenorMayor(){
    long *datos = getMenorMayor();
    cout<<"El menor elemento es: "<<datos[0]<<endl;
    cout<<"El mayor elemento es: "<<datos[1]<<endl;
}
```

Ver **Ejemplo7.cpp**

Métodos que reciben y retornan valores (1)

Un método que recibe y retorna valores, es similar a una función matemática. Por ejemplo *f*(x). En este caso el método se llama *f*, el cual *recibe* un argumento *x*, procesa *x* y retorna un valor.

Ejemplos de funciones en C++ que reciben y retornan:

```
double potencia = pow(2, 3);
```

Valor de retorno Argumentos

```
double raíz = sqrt(2);

string cadena = to_string(123);
```

Métodos que reciben y retornan valores (2)

Ejemplo #8:

Escribir un método en C++ que reciba como argumentos tres valores de tipo int y retorne el menor valor.

```
int getMenor(int n1, int n2, int n3){
    int menor = min(n1, min(n2, n3));
    return menor;
}
```

Ver
Ejemplo8.cpp

Métodos que reciben y retornan valores (3)

Ejemplo #9:

Escribir un método en C++ que reciba como argumentos un arreglo A[ ] de tipo int y un valor X a buscar dentro del arreglo. Retornar cuantas veces aparece X en A[ ]

```
long frecuencia(int *A, int x){
    long cont = 0;
    for(int i = 0; i < tam; i++){
        if(A[i] == x) cont++;
    }
    return cont;
}
```

Ver
Ejemplo9.cpp

Métodos que reciben y retornan valores (4)

Ejemplo #10:

Escribir un método en C++ que reciba como argumentos un arreglo A[ ], y un arreglo B[ ], ambos de tipo int y retorne *true* si A y B son iguales, o *false* en caso contrario.

Ayuda: dos arreglos son iguales, si posición a posición en ambos arreglos existen los mismos datos.

Por ejemplo:

- Sea A[] = {1,2,3} y B[] = {1,2,3}. En este caso el método retorna *true*.

- Sea A[] = {1,2,3} y B[] = {1,3,2}. En este caso el método retorna *false*.

Ver
Ejemplo10.cpp

Métodos que reciben y retornan valores (5)

Ejemplo #11:

Escribir dos métodos en C++.

El primero recibe como argumentos una matriz A[ ][ ] de tipo int y el indicador de fila cuyos elementos se desea sumar. El método debe retornar la suma.

El segundo método también recibe como argumentos una matriz A[ ][ ] de tipo int y el indicador de columna cuyos elementos se desea sumar. El método debe retornar la suma.

Ver
Ejemplo11.cpp

Métodos que reciben y retornan valores (6)

Ejemplo #12:

Escribir un método en C++ que reciba como argumentos una matriz A[ ][ ], y una matriz B[ ][ ], ambas de tipo int y retorne un arreglo M[ ] con los valores de A y B.

Ver
Ejemplo12.cpp

Paso de valores a funciones

En C y C++ básicamente existen dos maneras de enviar los valores a las funciones implementadas por el programador. La primera es denominada ***paso por valor*** y la segunda es ***paso por referencia***.

Paso por valor significa que el lenguaje (es decir, el compilador) crea una copia de la variable(s) que se pasa(n) como argumento, y ejecuta la función con esa copia. Al retornar, esas variables asignadas se borran de la memoria, y la variable original no sufre cambios en su valor.

Paso de valores a funciones
Paso por valor

Ejemplo #13:

Escribir un método en C++ que reciba como argumentos tres valores de tipo long, los multiplique por 10, y retorne la suma de los valores.

```
long multiplicacion(long numero1, long numero2, long numero3){
    long resul = 0;

    numero1 = numero1 * 10;
    numero2 = numero2 * 10;
    numero3 = numero3 * 10;

    cout<<"Valores en el metodo multiplicacion"<<endl;
    imprimir(numero1, numero2, numero3);

    resul = numero1 + numero2 + numero3;
    return resul;
}
```

Ver **Ejemplo13.cpp**

Paso de valores a funciones
Paso por referencia (1)

Paso por referencia significa que las funciones operan siempre con una referencia a la variable, y todas las operaciones que se realizan dentro de la función se están realizando en la variable pasada como argumento.

Para lograr el paso por referencia, en la declaración del método se antepone un ***&*** (ampersand) a la variable que se desea recibir por referencia.

Paso de valores a funciones
Paso por referencia (2)

Ejemplo #14:

Escribir un método en C++ que reciba como argumentos tres valores de tipo long por referencia, los multiplique por 10, y retorne la suma de los valores.

```
//Definición de los prototipos de funciones
void proceso();
void imprimir(long, long, long);
long multiplicacion(long &, long &, long &);
//-------------------------------------------------------------------------
long multiplicacion(long &num1, long &num2, long &num3){
     long resul = 0;

     num1 = num1 * 10;
     num2 = num2 * 10;
     num3 = num3 * 10;

     cout<<"Valores en el metodo multiplicacion"<<endl;
     imprimir(num1, num2, num3);

     resul = num1 + num2 + num3;
     return resul;
}
```

Ver **Ejemplo14.cpp**

Paso de valores a funciones
Paso por referencia (3)

Se debe tener en cuenta que al trabajar con un puntero, cuando se implementa un arreglo dinámico, este es tratado como una variable por referencia. Es decir, cualquier cambio que se realice sobre él en cualquier función, afecta sus valores originales.

En este caso no es necesario usar el ***&*** al momento de llamar un método y pasarle el arreglo.

Ejemplo #15:

Escribir un método en C++ que reciba como argumentos un arreglo A[ ] y un valor X, ambos de tipo long y retorne un arreglo M[ ] con los valores de A * X.

(Escalar por un vector)

Ver
Ejemplo15.cpp

Recursividad (1)

Una **función recursiva** es una función que se llama a si misma, ya sea en forma directa o indirecta (a través de otra función).

Por lo general, el paso recursivo incluye la palabra clave ***return***, ya que su resultado se combina con la parte del problema que la función supo como resolver, para formar un resultado que se pasara de vuelta a la función original que hizo la llamada.

Recursividad (2)

Ejemplo #16:

Escribir un método recursivo en C++ que reciba como argumento un valor de tipo long, calcule su factorial y lo retorne.

```
unsigned long factorial(long numero){
    if ( numero <= 1 ) // evalúa el caso base
        return 1; // casos base: 0! = 1 y 1! = 1
    else // paso recursivo
        return numero * factorial(numero - 1);
}
```

Ver
Ejemplo16.cpp

Recursividad (3)

Ejemplo #17:

Escribir un método recursivo en C++ que reciba como argumento un valor ***n*** de tipo long, y calcule el ***n***-ésimo número de Fibonacci y lo retorne.

```
unsigned long fibonacci(long numero){
    if ((numero == 0) || (numero == 1))//casos base
        return numero;
    else //paso recursivo
        return fibonacci(numero - 1) + fibonacci(numero - 2);
}
```

Ver
Ejemplo17.cpp

Sobrecarga de métodos (1)

C++ permite definir varias funciones con el mismo nombre, siempre y cuando éstas tengan distintas firmas.

A esta capacidad se le conoce como sobrecarga de funciones. Cuando se hace una llamada a una función sobrecargada, el compilador selecciona la función apropiada al examinar el número, tipos y orden de los argumentos en la llamada.

Sobrecarga de métodos (2)

Ejemplo #18:

Escribir un método en C++ sobrecargado 4 veces. 2 de los métodos reciben como argumentos valores de tipo long. Los otros 2 reciben como argumentos valores de tipo double. Todos deben retornar el mayor valor.

La primer versión del método recibe 3 argumentos long.

La segunda versión del método recibe 4 argumentos long.

La tercer versión del método recibe 3 argumentos double.

La cuarta versión del método recibe 4 argumentos double.

```
//prototipos de las funciones
long mayor(long, long, long);
long mayor(long, long, long, long);
double mayor(double, double, double);
double mayor(double, double, double, double);
```

Ver **Ejemplo18.cpp**