

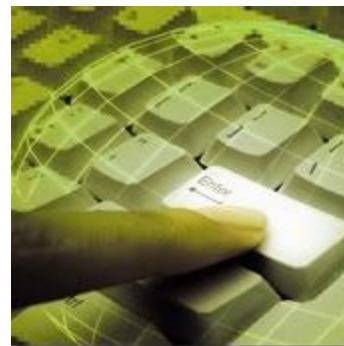


- Unidad 3

INSTRUCCIONES DE REPETICIÓN

(Capítulo 5 bibliografía)

Estructuras de ciclos – Ciclos FOR



» Ventre, Luis O.



Intro

- Como hemos visto la construcción de ciclos en C++ puede realizarse de varias formas. Una de estas es utilizando una **instrucción FOR**, cuya traducción puede interpretarse como “**para**” o “**para cada**”.
- En muchas situaciones, en particular aquellas que se necesita de una **cuenta fija**, o sea se conoce en un comienzo la cantidad de iteraciones a realizar es mas fácil el **formato de la instrucción FOR**.
 - **La sintaxis de la instrucción FOR tiene la siguiente forma:**

```
for (lista de inicialización; expresión; lista de alteración)  
    instrucción;
```



Ciclos FOR

- Aunque esta instrucción **parezca compleja**, es bien simple si analizamos sus **componentes por separado**.
- La sintaxis de esta instrucción muestra entre paréntesis 3 elementos **OPCIONALES**, **separados por “;”**; aunque los elementos pueden faltar, los “;” deben estar SIEMPRE.

```
for (lista de inicialización; expresión; lista de alteración)  
    instrucción;
```

- **lista de inicialización:** en su forma mas común, consiste de una sola instrucción usada para **establecer el valor inicial de un contador**.



OJO

- Al finalizar la instrucción FOR, NO se debe colocar ;

```
for (lista de inicialización; expresión; lista de alteración)  
    instrucción;
```

- Al finalizar la instrucción WHILE, NO se debe colocar ;

```
while (expresión)  
    instrucción;
```



Ciclos FOR ANIDADOS

- En muchas situaciones es necesario y conveniente utilizar un ciclo contenido dentro de otro ciclo. Esto es posible y los ciclos resultantes se conocen como **ciclos anidados**.
- El primer ciclo se llama **CICLO EXTERIOR**, el segundo es llamado **CICLO INTERIOR**. Generalmente todas las instrucciones del CICLO INTERIOR están contenidas en el cuerpo del ciclo EXTERIOR.
- Debido a esto, **en CADA ITERACION DEL CICLO EXTERIOR se PRODUCE todas las iteraciones DEL CICLO INTERIOR**.
- Se usan DIFERENTES VARIABLES PARA EL CONTROL DE LAS ITERACIONES DE CADA CICLO.



Ciclos FOR ANIDADOS

Cual será la salida?

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

int main()
{
    const int MAXI = 5;
    const int MAXJ = 4;
    int i, j;

    for(i = 1; i <= MAXI; i++)
    {
        cout << "\ni es ahora " << i << endl; //

        for(j = 1; j <= MAXJ; j++) // inicio ciclo interior
            cout << "  j = " << j; // fin ciclo interior
        // fin ciclo exterior -----+
    }

    cout << endl;

    system("PAUSE");
    return 0;
}
```

```
i es ahora 1
    j = 1  j = 2  j = 3  j = 4
i es ahora 2
    j = 1  j = 2  j = 3  j = 4
i es ahora 3
    j = 1  j = 2  j = 3  j = 4
i es ahora 4
    j = 1  j = 2  j = 3  j = 4
i es ahora 5
    j = 1  j = 2  j = 3  j = 4
Presione una tecla para continuar . . .
```



Ciclos FOR ANIDADOS

Para observar la utilidad de un ciclo anidado, veremos un ejemplo mas concreto, suponga que se necesita un programa en c++ para calcular la **calificación final para cada estudiante** en una clase de **20** alumnos. Cada alumno ha presentado **cuatro** exámenes durante el semestre. La calificación final es el **promedio** de las 4 calificaciones obtenidas.

El pseudocódigo seria:

Hago un ciclo que recorra los 20 alumnos

Establezco total de calificaciones del alumno en 0

Hago un ciclo para las 4 calificaciones de cada alumno

introducir calificación

acumular calificación al total

Fin del ciclo de calificaciones

Calculo promedio e imprimo nota final del alumno

Fin del ciclo de alumnos.



Ciclos FOR ANIDADOS

```
int main()
{
    const int NUMEROCLIFICACIONES = 4;
    const int NUMEROESTUDIANTES = 20;
    int i,j;
    double calificacion, total, promedio;

    for (i = 1; i <= NUMEROESTUDIANTES; i++) // inicio ciclo exterior
    {
        total = 0; // limpia el total para este estudiante
        for (j = 1; j <= NUMEROCLIFICACIONES; j++) // inicio ciclo interior
        {
            cout << "Ingrese la calificacion del examen para este estudiante: ";
            cin >> calificacion;
            total = total + calificacion; // agregar la calificacion al total
        } // fin del ciclo interior
        promedio = total / NUMEROCLIFICACIONES; // calcula el promedio
        cout << "\nEl promedio para el estudiante " << i
            << " es " << promedio << "\n\n";
    } // fin del ciclo exterior

    system("PAUSE");
}
```

Es importante notar el lugar donde se coloca total=0 y donde se calcula el promedio.



Ciclos FOR ANIDADOS

La salida generada será:

```
Ingrese la calificación del examen para este estudiante: 45
Ingrese la calificación del examen para este estudiante: 68
Ingrese la calificación del examen para este estudiante: 58
Ingrese la calificación del examen para este estudiante: 98
```

El promedio para el estudiante 1 es 67.25

```
Ingrese la calificación del examen para este estudiante: 45
Ingrese la calificación del examen para este estudiante: 35
Ingrese la calificación del examen para este estudiante: 46
Ingrese la calificación del examen para este estudiante: 98
```

El promedio para el estudiante 2 es 56

```
Ingrese la calificación del examen para este estudiante: 25
Ingrese la calificación del examen para este estudiante: 78
Ingrese la calificación del examen para este estudiante: 98
Ingrese la calificación del examen para este estudiante: 65
```

El promedio para el estudiante 3 es 66.5

```
Ingrese la calificación del examen para este estudiante: -
```

```
examen para este estudiante: 58
examen para este estudiante: 46
examen para este estudiante: 46
examen para este estudiante: 76
ante 18 es 56.5
```

```
examen para este estudiante: 98
examen para este estudiante: 48
examen para este estudiante: 54
examen para este estudiante: 65
```

El promedio para el estudiante 19 es 66.25

```
Ingrese la calificación del examen para este estudiante: 54
Ingrese la calificación del examen para este estudiante: 57
Ingrese la calificación del examen para este estudiante: 98
Ingrese la calificación del examen para este estudiante: 64
```

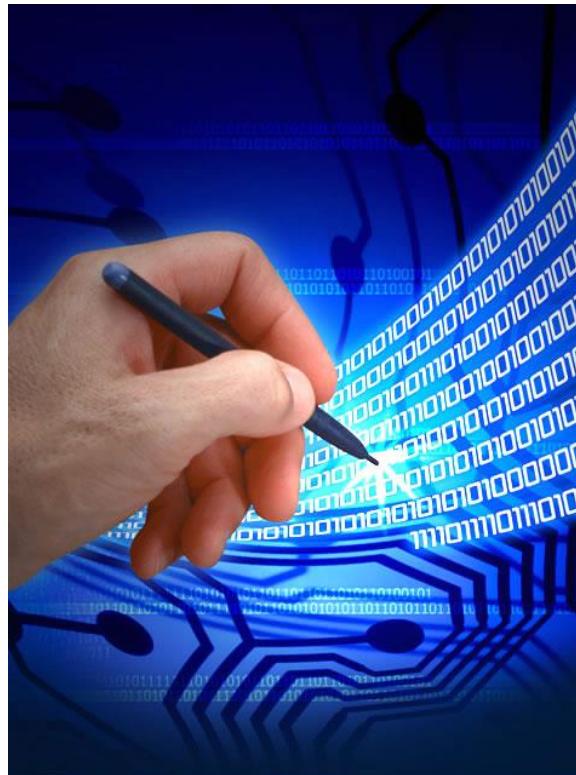
El promedio para el estudiante 20 es 68.25

Presione una tecla para continuar . . . -

- Puede implementar este programa con bucles while anidados?...



- Unidad 4



FUNCIONES

(Capítulo 6 bibliografía)

» Ventre, Luis O.



Intro

- El principal objetivo a cumplir, es entender la magnitud del problema a solucionar y poder **fraccionarlo o dividirlo** en problemas menores.
- **DIVIDE Y CONQUISTARAS.**
- Con esta idea en mente es posible establecer una analogía en el desarrollo de software con la producción de hardware e incluso con el proceso de fabricación de un automóvil.
- Cada modulo es desarrollado de manera individual y se encuentra libre de defectos antes de la integración, lo que permite optimizar el **tiempo de desarrollo, y desvincular el producto final de las partes internas.**
- Ahora si quisiera un automóvil con mas potencia que tendría que hacer?



Intro

- Ahora si, todos estos módulos forman el vehículo, pero cuando el usuario necesita acelerar **llama al motor**, este acepta entradas como combustible, aire y electricidad y produce potencia, que se la envía a la transmisión y así sucesivamente.
- Observe que estos módulos “conocen el mundo” a través de sus **entradas y salidas**. Y que el conductor no necesita saber nada de cómo funciona internamente la transmisión, ni el motor...
- De la misma manera, con este enfoque modular, que no son mas que funciones, los ingenieros diseñan y crean programas confiables.
- Hemos visto que todo programa en C++ debe tener una única función main, además de esta puede tener cualquier cantidad de funciones adicionales. Veremos como **escribir, transmitirle valores procesar datos y devolver resultados a continuación...**



Declaración de Funciones y Parámetros

- **UNA FUNCION SE INVOCA o Llama** dando su **NOMBRE** y transmitiéndole la lista de **parámetros** o valores a procesar, como argumentos, en los paréntesis que siguen al nombre de función.

nombre-de-la-función (datos transmitidos a la función)

Identifica la función

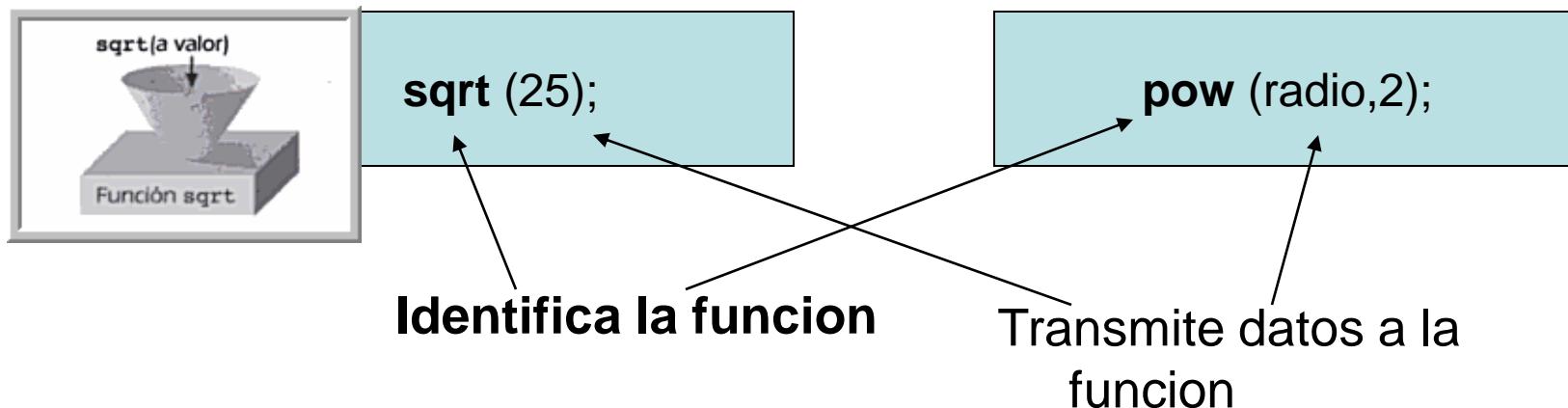
Transmite datos a la función

- MAXIMO CUIDADO EN EL **ENVIO DE PARAMETROS CUANDO SE INVOCARÁ LA FUNCION Y EN LOS VALORES DEVUELTOS.**



Declaración de Funciones y Parámetros

- Un ejemplo utilizado con anterioridad, fue cuando necesitábamos calcular la raíz cuadrada, o la potencia de dos números, estas son **FUNCIONES** que trae incorporada la **biblioteca CMATH**.



- La función INVOCADA, debe ser capaz de ACEPTAR los datos que le son transmitidos.



Declaración de Funciones y Parámetros

- ANTES QUE UNA FUNCION PUEDA SER LLAMADA DEBE SER DECLARADA, LA DECLARACION INICIAL DE LA MISMA SE CONOCE COMO **PROTOTIPO DE LA FUNCION**
- En el prototipo de una función se puede reconocer, su **NOMBRE**, sus datos válidos como **ARGUMENTOS Y EL ORDEN**, y su valor **DEVUELTO**.
- Por ejemplo el siguiente prototipo:

```
void encontrarMax(int, int);
```

Declara una función llamada encontrarMax, la cual recibe dos valores enteros como parámetro y no devuelve ningún valor(void).



Declaración de Funciones y Parámetros

- La forma general de escritura de un prototipo de función es:

tipo-de-datos-a-devolver nombre-de-función (lista de tipos de datos argumento)

- Ejemplos de prototipos de funciones:

```
int fmax(int, int);
```

```
double intercambio(int, char, char, double);
```

- Los prototipos de funciones permiten la verificación de errores en los tipos de datos por el compilador. Y asegura la conversión de todos los argumentos enviados al tipo de datos declarado.



Declaración de Funciones y Parámetros

- Llamada/ejecución de una función:

El llamado a una función es simple, basta con utilizar su nombre, y entre paréntesis enviar los argumentos necesarios.

- **Por ejemplo:**

```
encontrarMax(primerum, segundonum);
```

- Si uno de los argumentos es una variables, la función llamada recibe una copia del valor almacenado en la variable.
- En el ejemplo anterior, encontrarMax recibe una copia del valor de las variables primerum y segundonum pero no tiene conocimiento del nombre de estas variables. La función solo recibe los valores y debe almacenarlos, este mecanismo es por seguridad.



Declaración de Funciones y Parámetros

- Definición a una función:

Una función se define cuando se escribe. Cada función es definida solo una vez y puede ser utilizada por cualquier otra función que la declare correctamente.

Al igual que la función main, toda función en C++ consta de dos partes: **el encabezado y el cuerpo de la función.**

Línea de encabezado de función

{

declaraciones de constante y
variable; cualquier
otra instrucción de C++

}

El encabezado es siempre la primer línea, contiene **tipo de valor devuelto, su nombre, y los nombres tipos y orden de datos de argumento**



Declaración de Funciones y Parámetros

El encabezado de la función encontrarMax sería:

```
void encontrarMax(int x, int y)
```

Sin ; ojo!

Debe estar el tipo de dato y separarse con “,”

Los nombres de argumento en el encabezado se conocen como **parámetros formales**, el **parámetro x** se usara para almacenar el primer valor transmitido y el **parámetro y** para el segundo.

El primer paso en la invocación, es buscar los valores de primernum y segundonum, y enviárselos a la función, en este momento se almacenan en los parámetros nombrados “x” e “y”.



Declaración de Funciones y Parámetros

Una vez listo el encabezado puede escribirse el cuerpo, este debe comenzar con una llave de apertura “{” al igual que la función main, y debe finalizar con una llave de cierre “}”

```
void encontrarMax(int x, int y)
{
    int numMax; //inicio de cuerpo de la funcion
    /declaracion de variable

    if(x >= y)
        numMax=x;
    else
        numMax=y;

    cout<<“\n El maximo de los dos numeros es “
    <<numMax<<endl;
}
```

//fin de cuerpo y de la funcion



Declaración de Funciones y Parámetros

Colocación de instrucciones:

directivas del preprocesador

prototipos de función

int main()

{

 constantes simbólicas

 declaraciones de variables

 otras instrucciones ejecutables

 return valor

}



Declaración de Funciones y Parámetros

Generalmente, primero se enlista MAIN, ya que es la función principal que dará una idea al lector del programa que hace el mismo antes de ver los detalles de cada función.

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

void encontrarMax(int, int); // decla

int main()
{
    int primernum, segundonum;

    cout << "\nIngrese un numero: ";
    cin >> primernum;
    cout << "Bien! Por favor ingrese un ";
    cin >> segundonum;

    encontrarMax(primernum, segundonum);

    system("PAUSE");
    return 0;
}
```

```
Void encontrarMax(int x, int y) // inicio del cuerpo de la función
{
    int maxnum; // declaracion de variable

    if (x >= y) // encuentra el numero máximo
        maxnum = x;
    else
        maxnum = y;

    cout << "\nEl maximo de los dos numeros es: "
        << maxnum << endl;

    return;
} // fin de la funcion y del cuerpo de la misma
```



Declaración de Funciones y Parámetros

Cabos de FUNCION:

Enfoque común de programación, terminar la función main y luego las demás funciones. Problema para pruebas intermedias, compilación.

Se puede hacer una función que actúe como si fuera la correcta y que acepte los datos e imprima un mensaje en pantalla.

Esta función FALSA se llama CABO.

Funciones con listas de parámetros vacías:

En su prototipo pueden llevar void o nada en el lugar de los argumentos.

Argumentos por omisión:

Estos se enlistan en el prototipo de la función, y son transmitidos a la función en forma automática cuando los argumentos correspondientes son omitidos.



Declaración de Funciones y Parámetros

Ejemplo de argumentos por omisión:

Si el prototipo de nuestra función fuera:

```
void ejemplo (int, int = 5, double = 6.78);
```

Este prototipo proporciona valores por omisión para los dos últimos argumentos. Por lo tanto las siguientes llamadas a función son validas:

```
ejemplo ( 7, 2, 9.3) // no utiliza valores por omisión
```

```
ejemplo ( 7, 2)      // igual que llamado ejemplo( 7, 2, 6.78)
```

```
ejemplo (7)          // igual que llamado ejemplo (7, 5, 6.78)
```



Sobrecarga de funciones

Reutilización de nombres de FUNCION:

Único requisito, que el compilador pueda determinar que función utilizar basándose en los tipos de datos de los argumentos enviados a la función y no los tipos de datos del valor devuelto.

Cada función debe escribirse como una entidad separada.

El código puede tener leves cambios, aunque una buena práctica de programación es que funciones sobrecargadas realicen en esencia las mismas operaciones.

Cuando las funciones son idénticas y solo cambia el tipo de dato, es mejor implementar “plantillas de función” (ver pagina 314 libro)



Sobrecarga de funciones

Ejemplo:

```
void vabs(int x)
{if(x<0)
 x= -x;
 cout<<" el valor absoluto del numero entero es"<<x<<endl;
}

void vabs(float x)
{if(x<0)
 x= -x;
 cout<<" el valor absoluto del numero de punto flotante es"<<x<<endl;
}

void vabs(double x)
{if(x<0)
 x= -x;
 cout<<" el valor absoluto del numero de doble precision es"<<x<<endl;
}
```



Devolver un solo valor

Al utilizar el método de transmitir datos a una función presentado, la función llamada solo recibe **copias de los valores** contenidos en los **argumentos**. Esto se conoce como “**llamada por valor**” o “**pasaje de datos por valor**”.

De esta forma, la función invocada puede procesar los datos y devolver **un y solo un valor legitimo** a la función que la invoco.

Para que esto no produzca errores indeseados debe manejarse con cautela y de manera correcta:

Desde el punto de vista de la función llamada esta debe indicar:

- * El tipo de dato del valor devuelto (**encabezado 1º línea de la fx**)
- * El valor real que se devuelve



Devolver un solo valor

Para que la función retorne el valor solo es necesario colocar la instrucción:

return valor;

Debe cuidarse para evitar errores indeseados que el tipo de dato devuelto por la función declarado en el encabezado y el tipo utilizado en la instrucción de return coincidan!.

Desde el punto de vista del receptor, la función que llama debe:

- * Ser alertada del tipo de valor a esperar (**prototipo de función**)
- * Usar de manera apropiada el valor.

La variable utilizada para almacenar el valor devuelto debe ser del mismo tipo de dato.

```
max = encontrarMax(num1, num2);
```

```
cout << encontrarMax(num1, num2);
```



Devolver un solo valor

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

int encontrarMax(int, int); // prototipo de la función

int main()
{
    int primernum, segundonum, max;

    cout << "\nIngrese un numero: ";
    cin >> primernum;
    cout << "Bien! Por favor ingrese un segundo numero: ";
    cin >> segundonum;

    // la función es llamada aquí
    max = encontrarMax(primernum, segundonum);

    cout << "\nEl maximo de los dos numeros es: " << max << endl;

    system("PAUSE");
    return 0;
}
```

Alerta a main
y demás fx
sobre el valor
devuelto

Asignación
a la variable max
del valor devuelto
por la función



Devolver un solo valor

```
int encontrarMax(int x, int y)
{
    int maxnum; // inicio del cuerpo de la función
    // declaración de variable

    if (x >= y) // encuentra el numero máximo
        maxnum = x;
    else
        maxnum = y;

    return maxnum; // instrucción return
}
```

Encabezado declara que valor será devuelto

Ver en el libro ejemplo similar programa 6.6