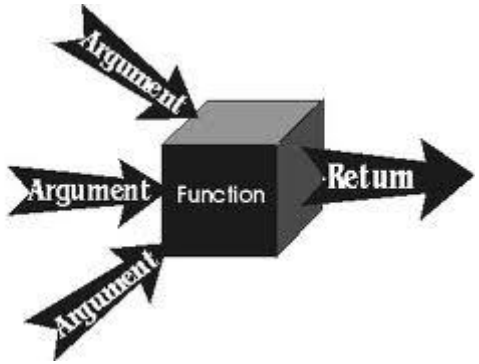
Universidad de Morón
Facultad de Informática, Cs. de la Comunicación y T. Especiales

Asignatura:

(701) Programación II

CLASE 9

Funciones
Parámetros por valor



Prof. Lic. Sonia Zugna de Jausoro

Modularidad

- ✓ El principio para resolver problemas es el de *"dividir para reinar"*,
- ✓ programas compuestos de bloques de sentencias que se conocen como **módulos**
- ✓ Reciben el nombre de: ***subprogramas***, ***subrutinas***, ***procedimientos***, ***funciones***.
- ✓ Se debe definir ese "subprograma", que luego será **llamado** o **invocado**.
- ✓ La **definición** del subprograma: sentencias que definirán lo que hará ese módulo, es única.
- ✓ La **Invocación** o **Llamada** será cuando se necesite ejecutar esas acciones definidas (en el módulo), puede ser única o varias.
- ✓ Reutilización del código.

Módulos en lenguaje C → FUNCIONES.

- ✓ Un programa C, se crea combinando el uso de:
  1) **funciones definidas y escritas por el programador**, y
  2) **funciones predefinidas** en bibliotecas.

- ✓ La ejecución de un programa en C, se hace siempre de forma secuencial, comenzando por la **función main**.

- ✓ Cada vez que se invoca una función, **la ejecución "salta":**
  de la función que se está ejecutando en ese momento
  a la función invocada.

  En cuanto la invocada termina, **la ejecución retoma** en el punto en que se produjo la llamada.

Definición

*La forma general de la **definición de una Función** es:*

```
tipo-de-retorno   identificador   ( lista-opcional-de-parámetros-formales )
{
declaraciones
sentencias
}
```

- ✓ El **tipo-de-retorno** es el tipo del valor que retorna la función. El tipo **void** es usado para indicar que la función no retorna ningún valor.
- ✓ **identificador** es el nombre que le damos a la función y que usaremos en la invocación a la misma
- ✓ La **lista-opcional-de-parámetros-formales** es una lista separada por comas con los tipos y los nombres de sus parámetros formales, la cual puede estar vacía (sin parámetros).
- ✓ Luego, entre llaves, **el cuerpo de la función** (declaraciones y sentencias)

- ✓ **return**: para devolver un resultado *al punto de llamada*.

  Al final de main, hay una sentencia return 0; puesto que main es una función que devuelve un int.

Invocación o Llamada

- ✓ La **invocación** de una función se hace empleando el **nombre de la función** junto con los **parámetros actuales**, si los tuviera. En caso de no tener parámetros, simplemente se colocará el nombre seguido de ().

- ✓ Una función puede ser **invocada** en el programa:

  1) **dentro de una expresión** del mismo tipo retornado por la función.

  2) **en una sentencia**, cuando el tipo de retorno de una función es void (no retorna ningún valor).

Función Separador.

// Función SEPARADOR que no recibe nada y no devuelve nada:

```
#include<iostream.h>
#include<conio.h>

// Definición de la Función Separador
void separador()
{
   cout<<endl<<"****************************************"<<endl;
}

int main(){
   separador();
   cout<<"   ***  FACULTAD DE INFORMATICA  ***";
   separador();
   cout<<"      ***  PROGRAMACION II  ***";
   separador();
   cout<<"   ***  Tema del dia: FUNCIONES  ***";
   separador();
```

// llamada a la Función Separador, constituye *por sí misma una sentencia*.

```
   getch();
   return(0);
}
```

Pasaje de Parámetros

- ✓ Las funciones en C, se comunican entre sí por medio de **datos**.
  Datos que son pasados como parámetros en la invocación y
  los datos que son devueltos como resultado de esas llamadas.

- ✓ Un módulo, generalmente, realizará alguna tarea empleando datos que le pasan como parámetros.

- ✓ en la **definición de la Función**, se define **cuántos y de qué tipo** son los datos que *espera* recibir cuando ella sea invocada y **con qué nombres** se van a referenciar a esos en el cuerpo de la función, éstos son los **parámetros formales**.

- ✓ Al "llamar" a una Función, habrá que pasarle los **valores** con los que ésta trabajará: uno por cada parámetro formal declarado. Estos valores se conocen como **parámetros actuales**.

Función con 2 Parámetros y que devuelve un resultado

```
[tipo_que_retorna]   nombre_función  ([tipo parametro1][, tipo parametro2][, ....])
{
   sentencia/s
  [return valor;]
}
```

```
// Función para sumar dos números
int  suma (int a, int b)
{
        int c;
        c = a + b;
        return c;
}
```

```
//Programa que Calcula el perímetro de un rectángulo
// Usa Función DOBLE

#include<iostream.h>
#include<conio.h>

float doble ( float valor )
{
    float  resu;
    resu = valor*2;
    return (resu);
}

int main(){
   float largo;
   float ancho;
   float perimetro;

   cout<<"Ingrese el largo: ";
   cin>>largo;
   cout<<endl;
   cout<<"Ingrese el ancho: ";
   cin>>ancho;

   perimetro = doble(largo) + doble(ancho);

   cout<<endl <<"El perimetro es: "<<perimetro;

   getch();
   return(0);
}
```

```
// Calcular el perímetro de un rectángulo
// la Función Perimetro llama a la Función Doble

#include<iostream.h>
#include<conio.h>

float doble(float valor)
{
    float resu;
    resu = valor*2;
    return(resu);
}

float perimetro(float largo, float ancho)
{
    float contorno;
    contorno = doble(largo) + doble(ancho);
    return(contorno);
}

int main(){
   float largo;
   float ancho;

   cout<<"Ingrese el largo: ";
   cin>>largo;
   cout<<endl<<"Ingrese el ancho: ";
   cin>>ancho;

   cout<<endl<<"El perimetro es: "<< perimetro(largo,ancho);

   getch();
   return(0);
}
```

Alcance o Ámbito de una variable

- ✓ **Es donde fue declarada.**
- ✓ **Es donde puede ser usada (o referenciada).**

- ➢ **Una variable declarada en una función, es LOCAL a la función, *fuera de ella no puede ser usada*.**

- ➢ Todo identificador declarado por fuera de cualquier función tiene alcance “de archivo”. Es accesible (o conocido) en “el archivo” desde el punto en donde ha sido declarado hasta el final del mismo.

  Estos identificadores suelen ser llamados **globales**.

  Las variables globales, las definiciones de funciones y los prototipos de funciones tienen **alcance global**.

Prototipo de una Función

Si:

1) Se desea usar una función antes de definirla ó
2) La función se encuentra en la biblioteca de funciones,

la manera de ***informar al compilador:***

- el tipo de resultado,
- cantidad de parámetros y
- tipos de parámetros,

es haciendo una declaración **prototipo** de la función.

*Nota: no hace falta informarle los **nombres** de los parámetros.*

El Prototipo de la Función, incluye la definición de la cabecera de la función, omitiendo el cuerpo de ésta y terminando en ";".

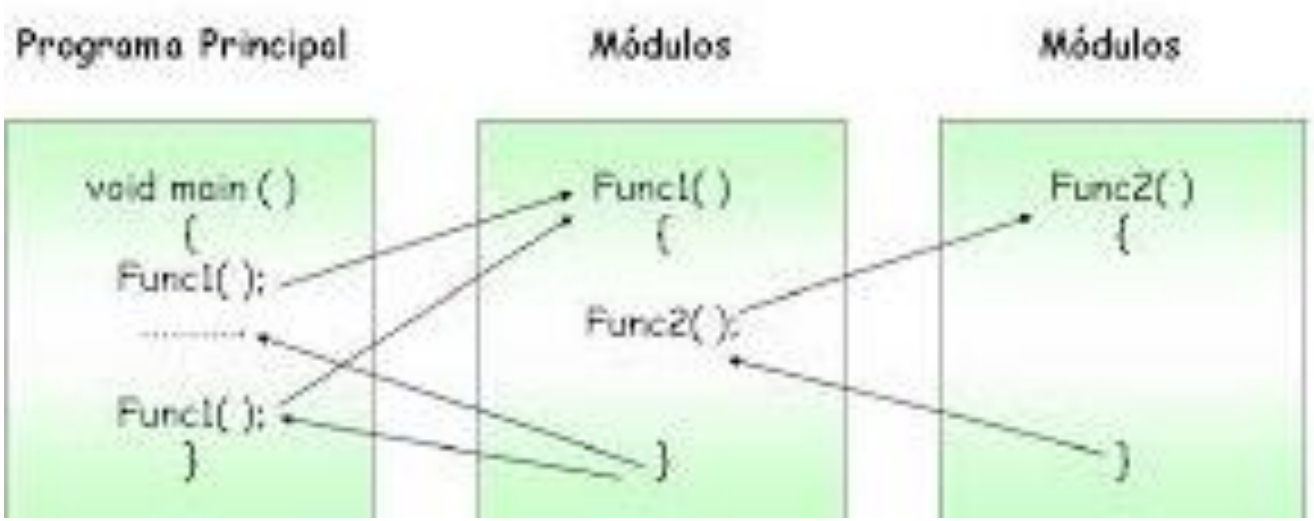
```
// Programa que lee un año y muestra si es o no bisiesto
//  Un año es bisiesto cuando es (divisible por 4 y no por 100) o (es divisible por 400).

#include <iostream.h>
#include <conio.h>

int bisiesto(int);   //declaración o prototipo de la función

int main()
{
int anio;
cout<< " Ingrese numero de anio ";
cin >> anio;

if (bisiesto(anio))     //llamada a la función
cout << "Bisiesto" << endl;
     else
     cout << "No es bisiesto" << endl;
getch();
return(0);
}

int bisiesto(int a)   //definición de la función
{
 if ( a%4==0  &&  a%100!=0  ||  a%400==0)
       return 1;
   else
       return 0;
}
```

OJO:
eliminar el espacio antes del //

Modificar la resolución de la función, haciendo que devuelva un valor booleano, y dejando UNA sola sentencia return.

Ejercitación:

Escriba un ejemplo de funciones que correspondan al siguiente esquema de llamadas:



Ejercicio 1: Realice el seguimiento

```
//Pasaje de parámetros por valor o por  copia

#include <iostream.h>
#include <conio.h>

int fResta(int uno, int dos)
{
   int aux;
   aux=uno-dos;

   return aux;

}

int main(){
   int uno, dos, resta;

   uno=1;
   dos=2;
   resta=  fResta(uno,dos);    //  llamada a la función fResta
   cout<<"La resta es: "<<resta;

   getch ();
   return 0;
}
```

Ejercicio 2: Realice el seguimiento

```
// Pasaje de parámetros por valor o por copia

#include <iostream.h>
#include <conio.h>

int fSumIncrementa (int x, int y)
{
   int aux;
   aux=x+y;
   x=x+1;
   y=y+1;

   cout<<"Mensaje desde la funcion:   y vale: "<<y<<"  x vale: "<<x;
   cout<<endl;
   return aux;
}

int main(){
   int uno=1;
   int dos=2;
   int resu=0;

   cout<<"La suma es: "<< fSumIncrementa (uno, dos);       //  llamada a la función
   cout<<endl;
    cout<<"Mensaje desde el main: "<<endl;
   cout<<"Uno vale: "<<uno<<endl;
   cout<<"Dos vale: "<<dos;

   getch ();
   return 0;
}
```

Ejercicio 3: Realice el seguimiento y encuentre los errores

```
// Pasaje de parámetros por valor y copia

#include <iostream.h>
#include <conio.h>

void fHace(char c, int b){
    cout<<"Mensaje desde la funcion: c= "<<c<<"  b= "<<b;

    if = 'a' return b;
     else    return 1;
}

int main(){
   int x;
   char y;
   cout<< "El resultado es= « << fHace(x,y);
   cout<< "b vale: "<< b;

 getch ();
 return 0;
}
```

```
// Pasaje de parámetros por valor o por copia

#include <iostream.h>
#include <conio.h>

int fdos(int par2)
{
   int resu;
   resu=par2*2;
   return resu;
}

int funo(int par1)
{
   int doble;
   doble= fdos(par1)+10;

   return doble;
}


int main ()
{
   int a=6;
   int b=7;
   int resu;

   resu= funo(a) + fdos(b);

   cout<<"Resu vale: "<<resu;

   getch();
   return 0;
}
```

Ejercicio 4: Realice el seguimiento

Tip:
Si una función llama a otra, la que es llamada debe ser **definida antes**.