

Capítulo 6: Modularidad Usando Funciones

Prof. Wilfredo Soto

sotow1@suagm.edu

Objetivos

2

- ❖ En este capítulo, aprenderás sobre:
 - ✓ Declaraciones de funciones y parámetros
 - ✓ Devolviendo un solo valor
 - ✓ Devolviendo múltiples valores
 - ✓ Conversión de coordenadas rectangulares a coordenadas polares
 - ✓ Alcance variable
 - ✓ Categorías de almacenamiento variable
 - ✓ Errores comunes de programación

Declaraciones de funciones y parámetros

3

- ▶ La interacción con una función incluye:
 - Pasar datos a una función correctamente cuando se llama
 - Devolver valores de una función cuando deja de funcionar
- ▶ Se llama a una función dando el nombre de la función y los argumentos que pasan entre paréntesis siguiendo el nombre de la función

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

Esto identifica a la función llamada

Esto transmite datos a la función

Figura 6.1 Llamar y transmitir datos a una función.

Declaraciones de funciones y parámetros (*continuación*)

- Antes de llamar a una función, se debe declarar para que funcione y lo haga
- El establecimiento de declaraciones para una función se conoce como, prototipo de la función
- El prototipo de función le dice a la función que se llama:
 - Tipo de valor que se devolverá formalmente, si corresponde
 - Tipo de datos y orden de los valores que la función que llama debe transmitir a la función llamada.
- Los prototipos de funciones se pueden colocar con las instrucciones de declaración de variables de la función, encima del nombre de la función que llama o en un archivo de encabezado separado

Declaración de función (*continuación*)

5

- La forma general de las instrucciones de prototipo de función es:

returnDataType functionName (list of argument data types);

donde el tipo de datos se refiere al tipo del valor que será devuelto de manera formal por la función.

Ejemplos de prototipos de función:

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

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

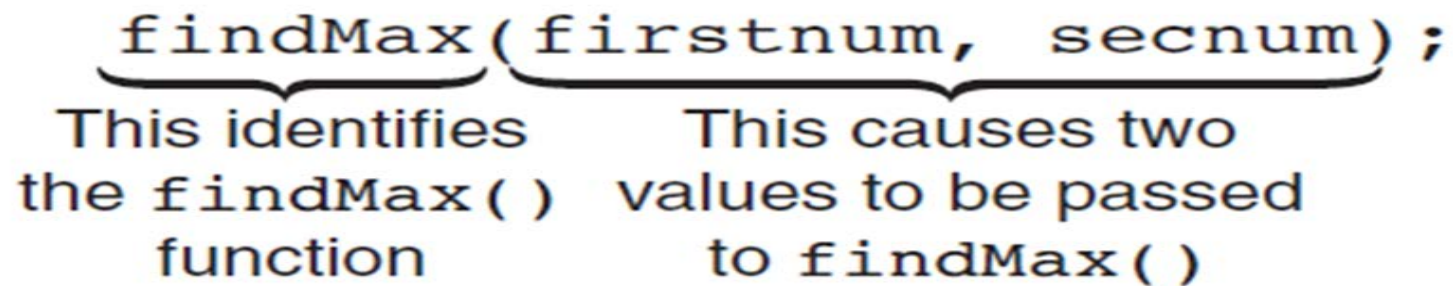
```
void desplegar(double, double);
```

Llamar a una función

- ▶ Los requisitos para llamar a una función incluyen:
 - Usar el nombre de la función
 - Encerrar los datos pasados a la función entre paréntesis siguiendo el nombre de la función, utilizando el mismo orden y tipo declarado en el prototipo de función

Llamar a una función (*continuación*)

- Los elementos encerrados entre paréntesis se llaman argumentos de la función llamada



The diagram shows the code `findMax(firstnum, secnum);` with two curly braces underneath. The first brace is under `findMax` and the second is under `(firstnum, secnum)`. Below the first brace is the text "This identifies the findMax () function". Below the second brace is the text "This causes two values to be passed to findMax ()".

```
findMax(firstnum, secnum);
```

This identifies the `findMax ()` function

This causes two values to be passed to `findMax ()`

Figura 6.2 Llamar a `findMax()` y transmitirle dos valores.

Llamada a una función (*continuación*)

8

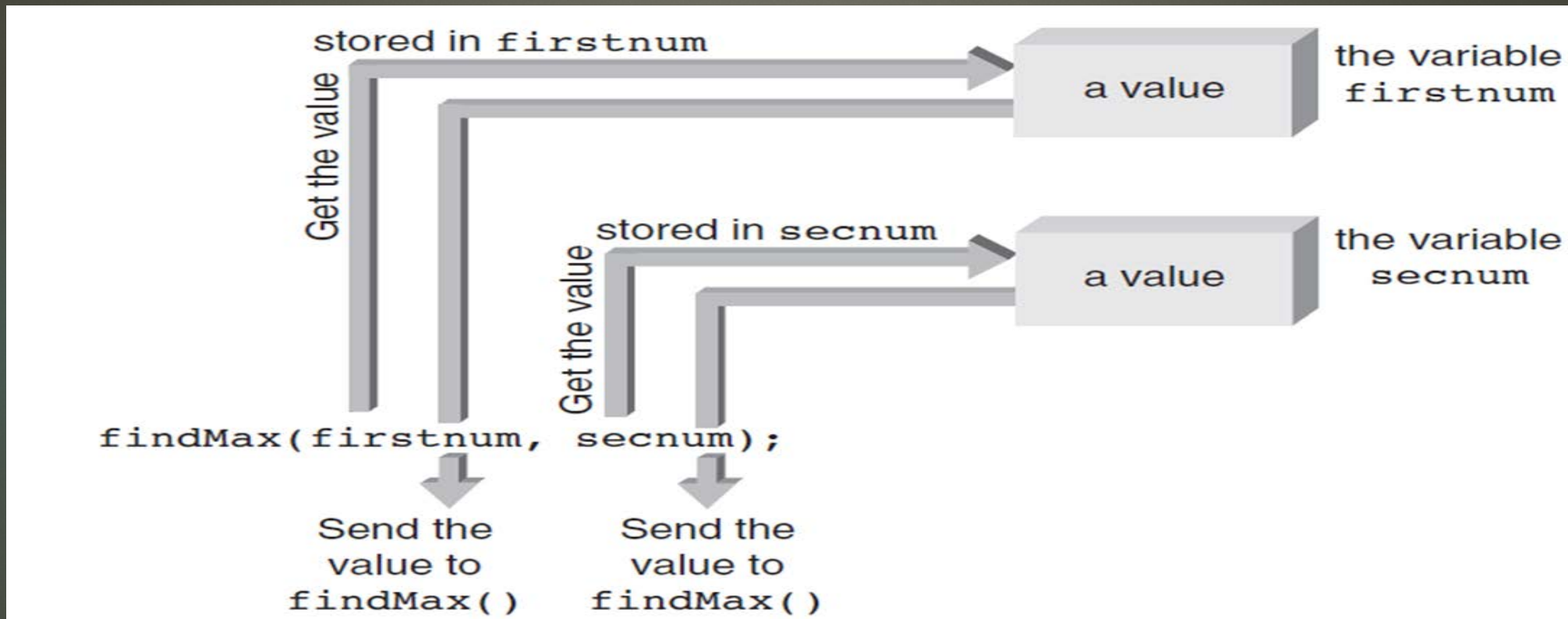


Figura 6.3 `findMax()` recibe valores actuales.

Definición de una Función

- ▶ Cada función de C ++ consta de dos partes:
 - **Encabezado de la función**
 - **Cuerpo de la función**
- ▶ El propósito del *encabezado de la función*:
 - Identificar el tipo de dato del valor devuelto por la función, proporcionarle un nombre a la función y especificar el número, el orden y el tipo de los argumentos esperados por la función
- ▶ El propósito del *cuerpo de función*:
 - Operar sobre los datos transmitidos y devolver en forma directa, cuando mucho, un valor a la función que llama.

Definición de una Función

10

- ▶ En vista que findMax() no devolverá de manera formal ningún valor y recibirá dos argumentos en número entero, puede usarse la siguiente línea de encabezado:

void findMax(int x, int y) ← sin punto y coma

- ▶ La primera parte del procedimiento de llamada ejecutado por la computadora implica ir a las variables *primernum* y *segundonum* y recuperar los valores almacenados. Estos valores son transmitidos luego a findMax() y almacenados al final en los parametros x & y (vease la figura 6.5).

Cuerpo de la función

- Un cuerpo de función comienza con una llave de apertura, { , contiene las declaraciones necesarias y otras instrucciones de C++, y termina con una llave de cierre, }.

```
tipoDatoQueDevuelve  nombreDeFunción  (lista parámetros)  
{  
  declaraciones de constantes simbólicas  
  declaraciones de variables  
  otras instrucciones de C++  
}
```

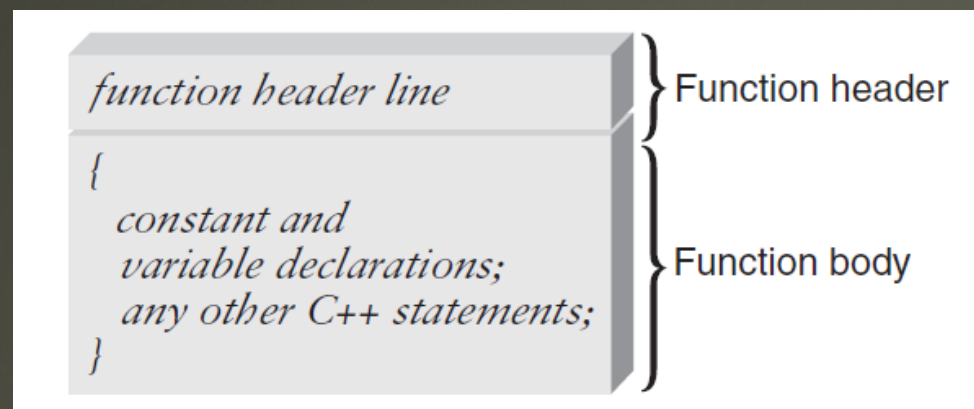
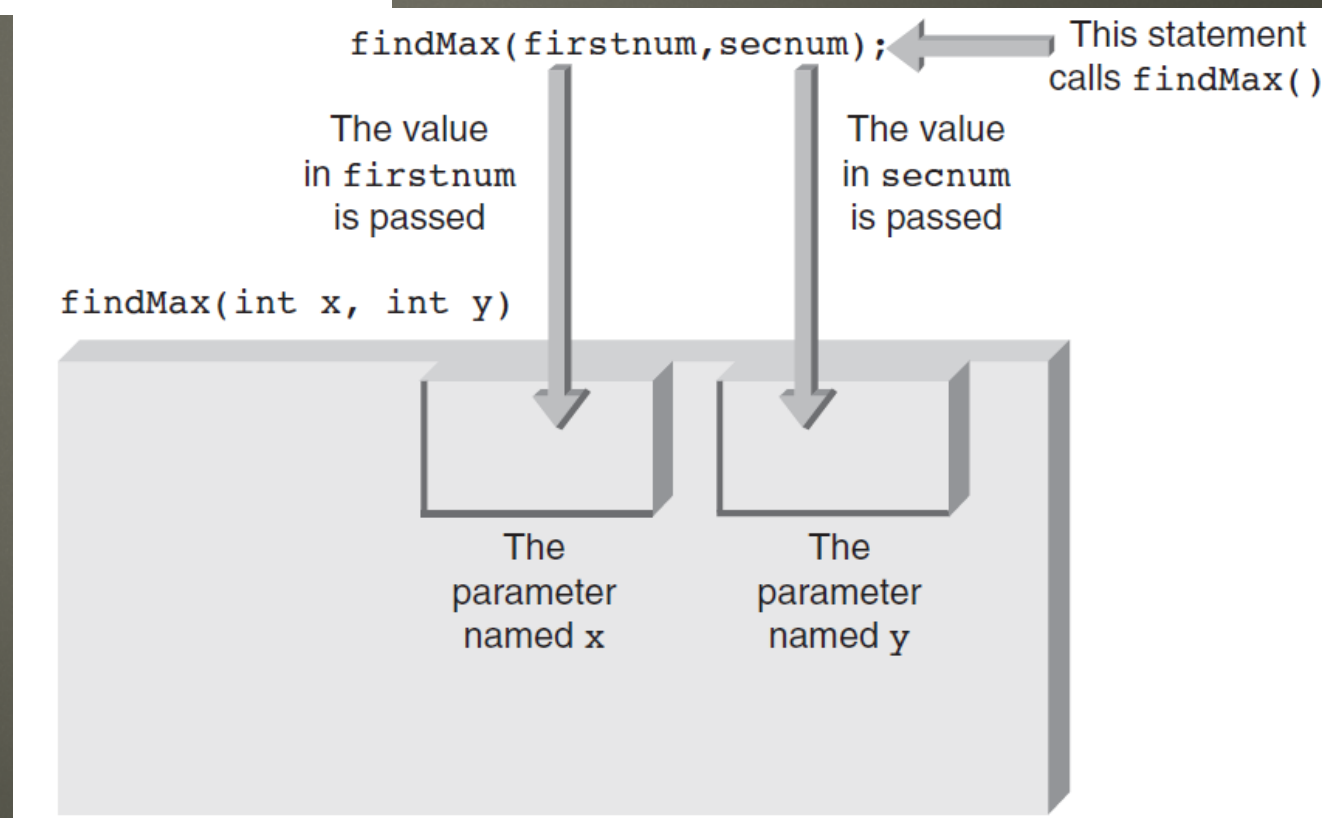


Figura 6.4 El formato general de una función

Figura 6.5 Almacenamiento de valores en parámetros x & y



Regla general para colocar instrucciones en un programa C++:

- ▶ Todas las directivas de preprocesador, constantes, variables y funciones nombradas deben declararse o definirse antes de que puedan ser utilizadas
- ▶ Aunque esta regla permite colocar directivas de preprocesador y declaraciones en todo el programa, al hacerlo se produce una estructura de programa deficiente

Funciones con listas de parámetros vacías

14

- ▶ Aunque las funciones útiles que tienen una lista de parámetros vacía son extremadamente limitadas, pueden ocurrir
- ▶ El prototipo de función para tal función requiere escribir la palabra clave ***void*** o nada en absoluto entre paréntesis seguido al nombre de la función

▶ Ejemplos:

▶ `int display () ;`

▶ `int display (void) ;`

Ordenamiento de Instrucciones

15

► Buena Forma De Programación:

```
directivas del preprocesador
prototipos de función
int main()
{
    constantes simbólicas
    declaraciones de variables
    otras instrucciones ejecutables
    return valor
}
definiciones de función
```



Programa 6.2

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

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

int main()
{
    int primernum, segundonum;

    cout << "\nIntroduzca un número: ";
    cin >> primernum;
    cout << "¡Estupendo! Por favor introduzca un segundo numero: ";
    cin >> segundonum;

    encontrarMax(primernum, segundonum); // aquí se llama a la función

    return 0;
}

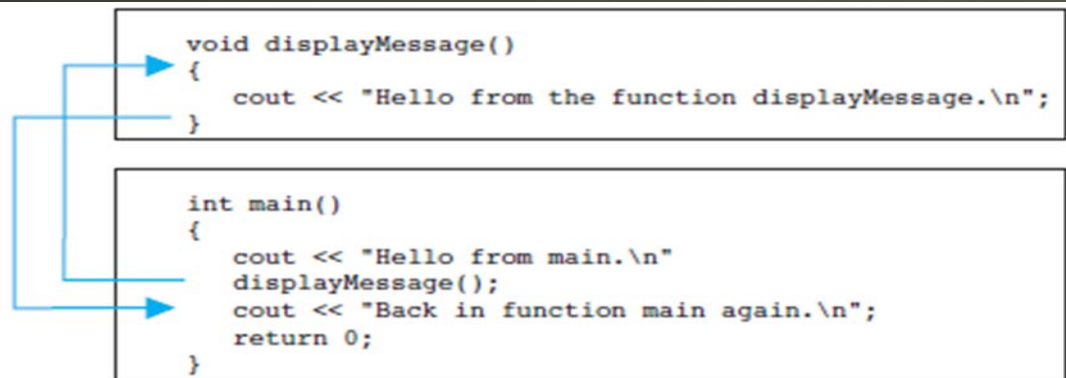
// en seguida está la función encontrarMax()

void encontrarMax(int x, int y)
{
    // inicio del cuerpo de función
    int numMax; // declaración de variable

    if (x >= y) // encontrar el número máximo
        numMax = x;
    else
        numMax = y;

    cout << "\nEl máximo de los dos números es "
         << numMax << endl;

    return;
} // fin del cuerpo de función y fin de la función
```



```

1 // This program has two functions: main and displayMessage
2 #include <iostream>
3 using namespace std;
4
5 //*****
6 // Definition of function displayMessage *
7 // This function displays a greeting.    *
8 //*****
9
10 void displayMessage()
11 {
12     cout << "Hello from the function displayMessage.\n";
13 }
14
15 //*****
16 // Function main *
17 //*****
18
19 int main()
20 {
21     cout << "Hello from main.\n";
22     displayMessage();
23     cout << "Back in function main again.\n";
24     return 0;
25 }

```

Program Output

```

Hello from main.
Hello from the function displayMessage.
Back in function main again.

```

```

1 // The function displayMessage is repeatedly called from a loop.
2 #include <iostream>
3 using namespace std;
4
5 //*****
6 // Definition of function displayMessage *
7 // This function displays a greeting.    *
8 //*****
9
10 void displayMessage()
11 {
12     cout << "Hello from the function displayMessage.\n";
13 }
14
15 //*****
16 // Function main *
17 //*****
18
19 int main()
20 {
21     cout << "Hello from main.\n";
22     for (int count = 0; count < 5; count++)
23         displayMessage(); // Call displayMessage
24     cout << "Back in function main again.\n";
25     return 0;
26 }

```

Program Output

```

Hello from main.
Hello from the function displayMessage.
Hello from the function displayMessage.
Hello from the function displayMessage.
Hello from the function displayMessage.
Hello from the function displayMessage.
Back in function main again.

```


Argumentos predeterminados

17

- ▶ C ++ proporciona argumentos predeterminados en una llamada de función para mayor flexibilidad
 - ▶ Uso primario: para ampliar la lista de parámetros de las funciones existentes sin requerir ningún cambio en las listas de parámetros de llamada ya utilizadas en un programa
 - ▶ Enumerado en el prototipo de función y transmitido automáticamente a la función llamada cuando los argumentos correspondientes se omiten de la llamada de función
 - ▶ Ejemplo: prototipo de función con argumentos predeterminados

```
void example (int, int = 5, double = 6.78)
```

Reutilizando nombres de funciones (*overloading*)

- ▶ C ++ proporciona la capacidad de usar el mismo nombre de función para más de una función
 - Conocido como ***function overloading***
- ▶ Un solo requisito para crear más de una función con el mismo nombre:
 - El compilador debe poder determinar qué función usar según los tipos de datos de los parámetros (no el tipo de datos del valor de retorno, si corresponde)
 - Cuál de las funciones se llama depende del tipo de argumento suministrado en el momento de la llamada

Plantilla de funciones (*Function Templates*)

19

- ▶ Plantilla de función: función única completa que sirve como modelo para una familia de funciones
 - La función de la familia que realmente se crea depende de la función específica llamada
- ▶ Generaliza la escritura de funciones que realizan esencialmente la misma operación, pero en diferentes tipos de datos de parámetros
- ▶ Permite escribir una función general que maneje todos los casos pero donde el compilador puede establecer parámetros, variables e incluso el tipo de retorno en función de la llamada a la función real

Plantilla de funciones

(Function Templates)

20



Program 6.3

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

template <class T>
void showabs(T number)
{
    if (number < 0)
        number = -number;
    cout << "The absolute value of the number is "
         << number << endl;
    return;
}

int main()
{
    int num1 = -4;
    float num2 = -4.23f;
    double num3 = -4.23456;

    showabs(num1);
    showabs(num2);
    showabs(num3);

    return 0;
}
```

template <class T>

Esta línea, la cual se llama *prefijo de plantilla*, se usa para informar al compilador que la función que sigue inmediatamente es una plantilla que usa un tipo de datos nombrado *T*.

Dentro de la plantilla de función se usa la *T* de la misma manera que cualquier otro tipo de datos, como int, float, double, etc.

```
El valor absoluto del numero es 4
El valor absoluto del numero es 4.23
El valor absoluto del numero es 4.23456
```

Puede utilizarse en cambio cualquier letra o identificador que no sea una palabra clave.
Ej .

template <class TIPOD>
void abs(TIPOD numero)

Devolviendo un valor único

21

- ▶ La función que recibe un argumento pasado por valor no puede alterar inadvertidamente el valor almacenado en la variable utilizada para el argumento
- ▶ La recepción de funciones pasadas por argumentos de valor puede procesar los valores que se le envían de cualquier manera y devolver uno y solo un valor "legítimo" directamente a la función de llamada.

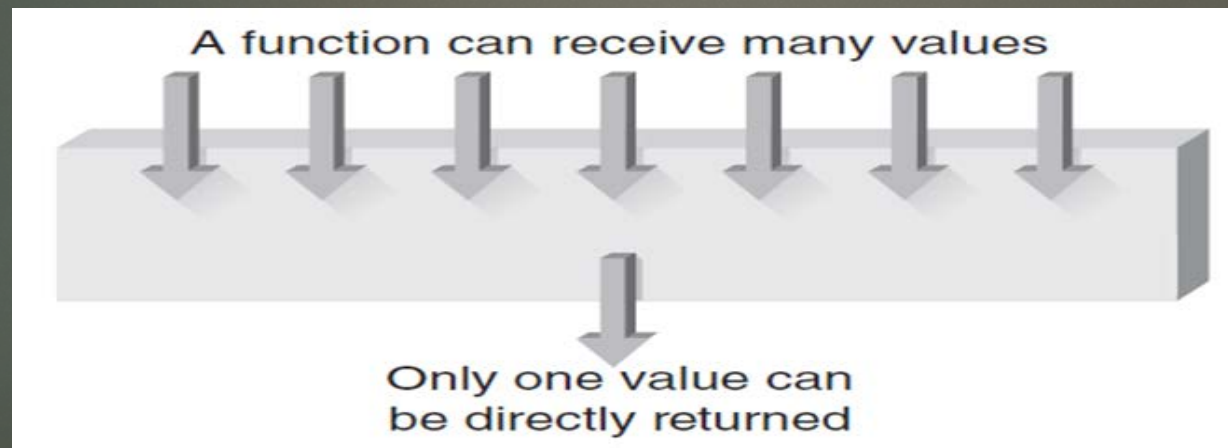


Figura 6.10 Una función devuelve directamente como máximo un valor

Inline Functions (*Funciones Inl ne*)

22

- Llamar a una funci n coloca una cierta cantidad de sobrecarga en una computadora como:
 - ❖ Colocar valores de argumento en una regi n de memoria reservada (llamada stack (o pila)) a la que la funci n tiene acceso
 - ❖ Pasando el control a la funci n
 - ❖ Proporcionando una ubicaci n de memoria reservada para cualquier valor devuelto (de nuevo, usando el stack para este prop sito)
 - ❖ Volviendo al punto correcto en el programa que llama la funci n

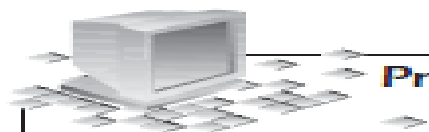
Inline Functions (*Funciones Inl ne*)

(*cont.*)

- ▶ El pago de sobrecarga asociada a llamar a una funci n se justifica cuando se invoca una funci n muchas veces esto Puede reducir sustancialmente el tama o de un programa
- ▶ Para funciones peque as que no se llaman muchas veces, la sobrecarga de pasar y devolver valores puede no estar garantizada
- ▶ Para Funciones *Inl ne*:
 - Agrupa l neas de c digo repetidas juntas bajo un nombre de funci n com n
 - Haga el compiladce or coloque este c digo en el programa donde sea que se llame a la funci n

Ejemplo: Función Inline

24



Programa 6.7

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

inline double convertir_temp(double in_temp) // una función inline
{
    return (5.0/9.0) * (in_temp - 32.0);
}

int main()
{
    const CONVERSIONES = 4; // numero de conversiones que se harán
    int cuenta;
    double fahren;

    for(cuenta = 1; cuenta <= CONVERSIONES; cuenta++)
    {
        cout << "\nIntroduzca una temperatura en grados Fahrenheit: ";
        cin >> fahren;
        cout << "El equivalente en grados Celsius es "
             << convertir_temp(fahren) << endl;
    }

    return 0;
}
```


Inline Functions (*Funciones Inl ne*)

(*cont.*)

- ▶ Ventaja: aumento en la velocidad de ejecuci n
 - ❖ Debido a que la funci n en l nea se expande e incluye en cada expresi n o enunciado que la llama, no se pierde tiempo de ejecuci n debido a la sobrecarga de devoluci n y llamada que requiere una funci n no en l nea.
- ▶ Cada vez que se hace referencia a una funci n en l nea, el c digo completo se reproduce y almacena como parte integral del programa.
- ▶ Una funci n no en l nea se almacena en la memoria solo una vez
- ▶ Las funciones en l nea se deben usar solo para funciones peque as que no se llaman extensamente en un programa

Devolución de valores múltiples

26

- ▶ En una invocación de función típica, la función llamada recibe valores de su función de llamada, almacena y manipula los valores pasados, y devuelve directamente como máximo un valor
 - ❖ Pasar por valor: cuando los datos se pasan de esta manera
 - ❖ Pasar por referencia: otorgar a una función llamada acceso directo a las variables de su función:
 - Como función de transmisión por referencia, en vista que la función llamada puede hacer referencia, o tener acceso, a la variable cuya dirección se ha transmitido. C++ proporciona dos tipos de parámetros de dirección, referencias y apuntadores.

Pasar y usar los parámetros de referencia

27

- ▶ Desde el lado del emisor, llamar a una función y pasar una dirección como un argumento que se acepta como parámetro de referencia es lo mismo que llamar a una función y pasar un valor
- ▶ Si un valor o una dirección se pasa realmente depende de los tipos de parámetros declarados

Pasar y usar los parámetros de referencia (*continuación*)

28

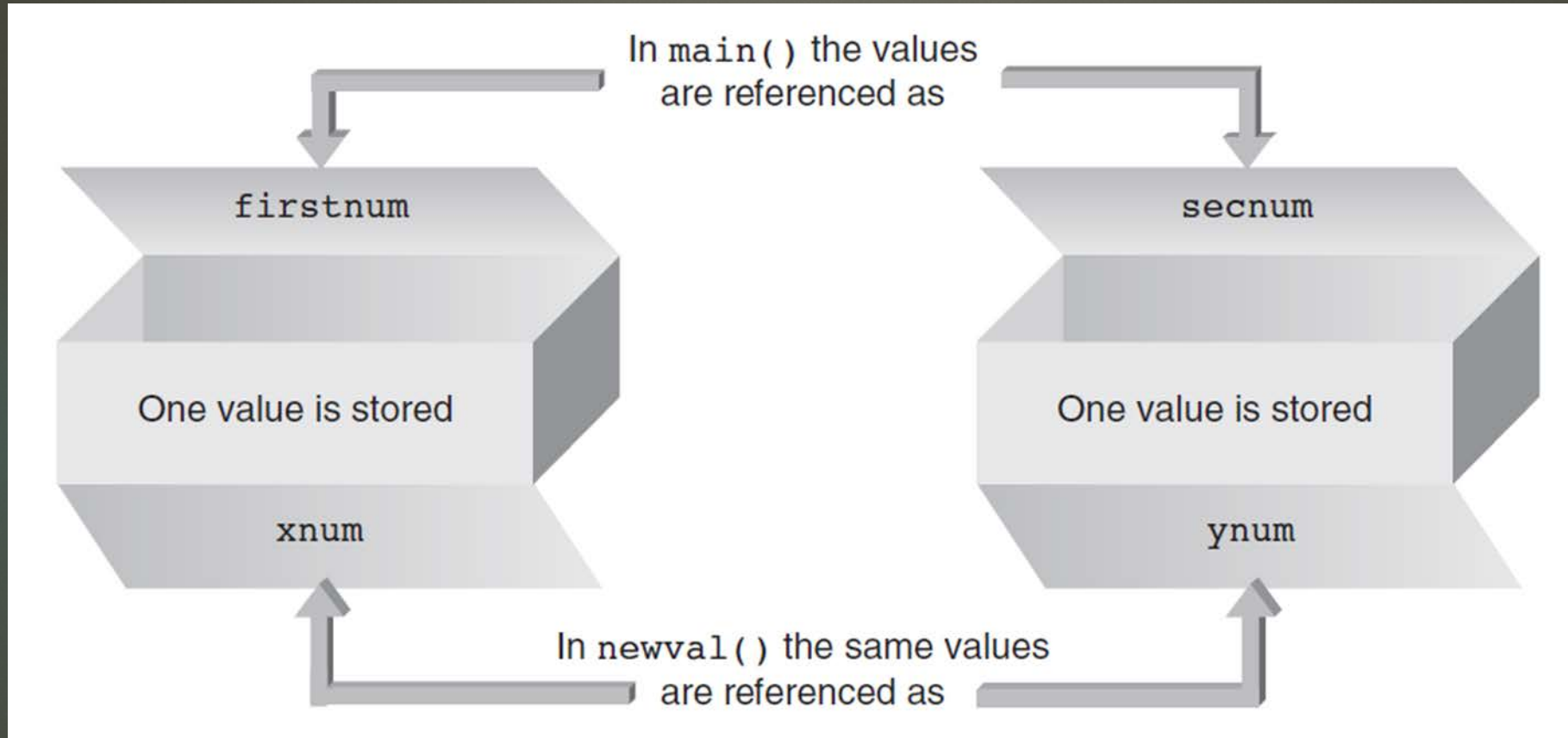


Figura 6.11 La equivalencia de argumentos y parámetros en el Programa 6.8

Pasar y usar los parámetros de referencia (continuación)

29



Program 6.8

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

void newval(double&, double&); // prototype with two reference parameters

int main()
{
    double firstnum, secnum;

    cout << "Enter two numbers: ";
    cin >> firstnum >> secnum;
    cout << "\nThe value in firstnum is: " << firstnum << endl;
    cout << "The value in secnum is: " << secnum << "\n\n";
    newval(firstnum, secnum); // call the function
    cout << "The value in firstnum is now: " << firstnum << endl;
    cout << "The value in secnum is now: " << secnum << endl;

    return 0;
}

void newval(double& xnum, double& ynum)
{
    cout << "The value in xnum is: " << xnum << endl;
    cout << "The value in ynum is: " << ynum << "\n\n";
    xnum = 89.5;
    ynum = 99.5;

    return;
}
```

Enter two numbers: 22.5 33.0

The value in firstnum is: 22.5

The value in secnum is: 33

The value in xnum is: 22.5

The value in ynum is: 33

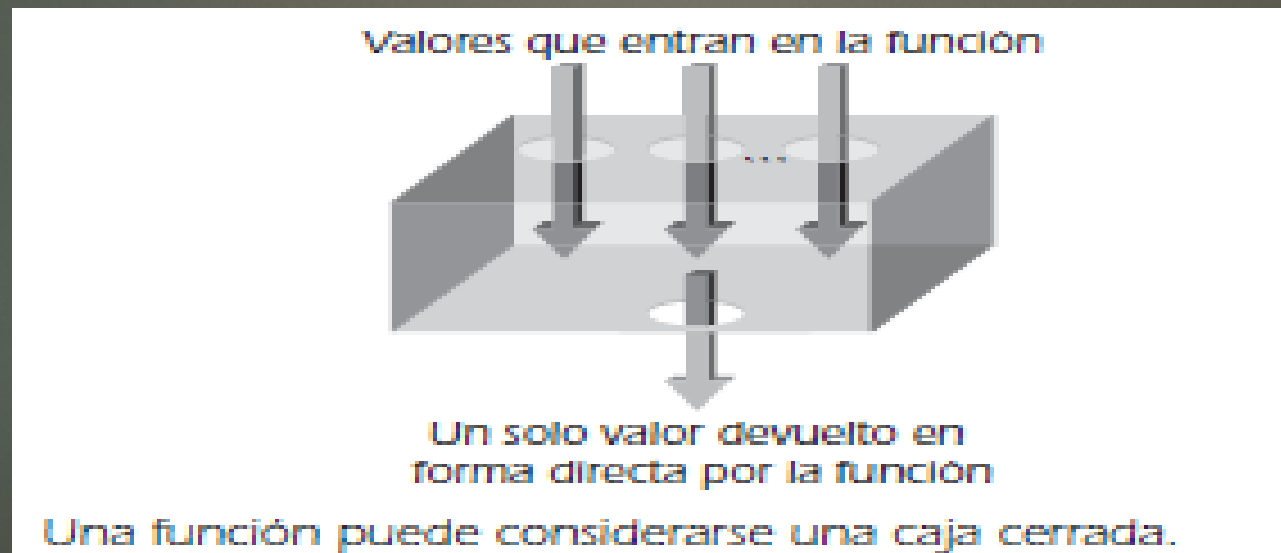
The value in firstnum is now: 89.5

The value in secnum is now: 99.5

Variable Scope (*Alcance variable*)

30

- Una función puede considerarse una caja cerrada, con ranuras en la parte superior para recibir valores y una sola ranura en la parte inferior para devolver un valor



Variable Scope (*Alcance variable*)

(*continuación*)

- *Variables locales*: variables creadas en una función que están convencionalmente disponibles solo para la función
- *Alcance (Scope)*: Sección del programa donde el identificador es válido o "conocido"
- Una variable con *alcance local* es simplemente una con ubicaciones de almacenamiento separadas por una declaración dentro de la función que las declaró
- ▶ Una variable con **alcance global**, por lo general denominada **variable global**, el almacenamiento se crea mediante una instrucción de declaración localizada fuera de cualquier función.



Program 6.15

32

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

int firstnum;    // create a global variable named firstnum

void valfun();   // function prototype (declaration)

int main()
{
    int secnum;    // create a local variable named secnum

    firstnum = 10;    // store a value in the global variable
    secnum = 20;      // store a value in the local variable

    cout << "From main(): firstnum = " << firstnum << endl;
    cout << "From main(): secnum = " << secnum << endl;

    valfun();        // call the function valfun

    cout << "\nFrom main() again: firstnum = " << firstnum << endl;
    cout << "From main() again: secnum = " << secnum << endl;
    return 0;
}

void valfun()    // no values are passed to this function
{
    int secnum;    // create a second local variable named secnum

    secnum = 30;    // affects only this local variable's value

    cout << "\nFrom valfun(): firstnum = " << firstnum << endl;
    cout << "From valfun(): secnum = " << secnum << endl;

    firstnum = 40;    // changes firstnum for both functions

    return;
}
```

From main(): firstnum = 10

From main(): secnum = 20

From valfun(): firstnum = 10

From valfun(): secnum = 30

From main() again: firstnum = 40

From main() again: secnum = 20

Estas variables pueden ser utilizadas por todas las funciones que se colocan físicamente después de la declaración de la variable global. Esto se muestra en el programa 6.15, donde se utiliza a propósito el mismo nombre de variable dentro de ambas funciones contenidas en el programa.

Operador de resolución de alcance

33

- ▶ Cuando una variable local tiene el mismo nombre que una variable global, todas las referencias al nombre de la variable hechas dentro del alcance de la variable local se refieren a esta variable local.



Programa 6.18

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

double numero = 42.8;    // una variable global llamada número

int main()
{
    double numero = 26.4;    // una variable local llamada número

    cout << "El valor de número es " << numero << endl;

    return 0;
}
```

Cuando se ejecuta el programa 6.18, se despliega la siguiente salida.

El valor de número es 26.4

Operador de resolución de alcance (*continuación*)

- Para hacer referencia a una variable global cuando una variable local del mismo nombre está en el alcance, use el operador de resolución de alcance de C ++, que es ::

```
#include <iostream>
using namespace std;
double number = 42.5; // a global variable named number
int main()
{
    double number = 26.4; // a local variable named number
    cout << "The value of number is " << ::number << endl;
    return 0;
}
```


Mal uso de las variables globales

35

- ▶ Las variables globales permiten a los programadores "saltar" las salvaguardas normales proporcionadas por las funciones
- ▶ En lugar de pasar variables a una función, es posible hacer todas las variables globales: *no haga esto*
 - El uso indiscriminado de variables globales destruye las salvaguardas que C++ proporciona para hacer que las funciones sean independientes y estén aisladas entre sí.
 - Usar solo variables globales puede ser especialmente desastroso en programas grandes con muchas funciones creadas por el usuario

Categorías de almacenamiento variable

36

- ▶ El alcance de una variable puede considerarse como el espacio en el programa donde la variable es válida
- ▶ Además de la dimensión espacial representada por su alcance, las variables también tienen una dimensión temporal. La dimension temporal se refiere al tiempo que las ubicaciones de almacenamiento son reservadas para una variable.
- ▶ La dimension temporal se conoce como la “vida” de la variable.
- ▶ Cuándo y durante cuanto tiempo se guardan las ubicaciones de almacenamiento de una variable antes de que se liberen se puede determinar por la categoría de almacenamiento de la variable

Categorías de almacenamiento variable (*continuación*)

- ▶ Las cuatro categorías de almacenamiento disponibles son:
 - auto
 - static
 - extern
 - register

```
auto int num;          // auto storage category and int data type
static int miles;      // static storage category and int data type
register int dist;      // register storage category and int data type
extern int volts;       // extern storage category and int data type
auto float coupon;     // auto storage category and float data type
static double yrs;     // static storage category and double data type
extern float yld;       // extern storage category and float data type
auto char inKey;       // auto storage category and char variable type
```

Categorías de almacenamiento de variables locales

- ▶ Las variables locales solo pueden ser miembros de las categorías de almacenamiento `auto`, `static` o `register`.
- ▶ El almacenamiento para variables automáticas locales es reservado o creado en forma automática cada vez que se llama una función que declara variables automáticas.
 - ❖ Siempre que la función no haya devuelto el control a su función de llamada, todas las variables automáticas locales a la función están "vivas".
- ▶ Una variable `static` local no se crea y se destruye cada vez que la función que declara se llama
 - ❖ Las variables `static` locales siguen existiendo durante la vida útil del programa

Categorías de almacenamiento de variables locales (*continuación*)

- ▶ La mayoría de las computadoras tienen algunas áreas de almacenamiento de alta velocidad, llamadas registros, ubicadas en la CPU que también se pueden usar para almacenamiento variable
 - ▶ Debido a que los registros están ubicados en la CPU, se puede acceder a ellos más rápido que las áreas de almacenamiento de memoria normales ubicadas en la unidad de memoria de la computadora.

Clases de almacenamiento de variables globales

- ▶ Las variables globales son creadas por declaraciones de definición externas a una función
- ▶ Por su naturaleza, las variables globales no van y vienen con el llamado de una función
- ▶ Después de crear una variable global, existe hasta que el programa en el que se ha declarado haya terminado de ejecutarse
- ▶ Las variables globales pueden declararse con la categoría de almacenamiento `static` o `extern`, pero no ambas.

```
Ej. extern int suma;  
    extern double voltios;  
    static double corriente;
```


Errores Comunes de Programación

41

- ▶ Pasar tipos de datos incorrectos
- ▶ Errores que ocurren cuando la misma variable se declara localmente tanto en la llamada como en las funciones llamadas
- ▶ Omitir el prototipo de la función antes o dentro de la función de llamada.
- ▶ Terminar un encabezado de función con un punto y coma
- ▶ Olvidar incluir el tipo de datos de los parámetros de una función en el encabezado de la función

Resumen

- ▶ Se llama a una función dando su nombre y pasándole cualquier dato entre paréntesis después del nombre
- ▶ El tipo de devolución de una función es el tipo de datos del valor que devuelve la función
- ▶ Los argumentos pasados a una función cuando se llama deben cumplir con los parámetros especificados por el encabezado de la función
- ▶ Las funciones se pueden declarar para todas las funciones que llaman por medio de un prototipo de función
- ▶ Cada variable tiene una categoría de almacenamiento, que determina cuánto tiempo se conserva el valor de la variable

Ejercicio #1 de práctica para el salón

43

En un programa que funcione. Asegúrese que su función es llamada desde `main()` y Pruebe la función transmitiéndole varios datos.

Escriba una función nombrada *revisar()* que tenga tres parámetros. El primer parametro debiera aceptar un numero entero, el segundo parametro un numero de precisión doble y el tercer parametro un numero de precisión doble. *El cuerpo de la función debiera desplegar solo los valores de los datos transmitidos a la función cuando es llamada.*

(*NOTA:* Cuando se rastrean errores en las funciones, es muy útil hacer que la función despliegue los valores que se le han transmitido. Con bastante frecuencia, el error no esta en lo que el cuerpo de la función hace con los datos, sino en los datos recibidos y almacenados.)

Ejercicio #2 de práctica para el salón

- El volumen, v , de un cilindro esta dado por la formula

$$v = \pi r^2 l$$

donde r es el radio del cilindro y l es su largo. Usando esta formula, escriba una función C++ nombrada *vol_cil()* que acepte el radio y el largo de un cilindro (use cin) y devuelva su volumen. Incluya la función escrita en el ejercicio en un programa que funcione. Asegúrese que su función es llamada desde *main()* y devuelve en forma correcta un valor a *main()*. **Haga que *main()* use una instrucción cout** para desplegar el valor devuelto. Pruebe la función transmitiéndole varios datos.

OJO:

Dentro de la función *vol_cil()* debera llamar otra función para buscar el valor de pi dado por la ecuación $\pi = 2.0 * \text{asin}(1.0)$