Programación de Ordenadores

Ingeniería Química

5

David Pelta

Depto de Ciencias de la Computación e I.A.

Universidad de Granada

FUNCIONES Y PROCEDIMIENTOS

- Programación modular.
- Funciones.
- Parámetros formales y actuales.
- Procedimientos.
- Paso de parámetros por valor y por referencia.

Motivación I

- Una forma natural de atacar problemas grandes es dividirlo en sub-problemas que se puedan resolver de forma "independiente" y luego combinarse.
- En programación, esta tecnica se refleja en el uso de sub-programas: conjunto de instrucciones que realizan una tarea específica.
- En C++ los sub-programas se denominan funciones.
- Recibe valores de entrada (parámetros) y proporciona un valor de salida (valor de retorno). La función se *llama* o *invoca* cuando deseamos aplicarla.
- Analogía: un jefe (el que llama la función) solicita a un empleado (la función), que realice una tarea y devuelva los resultados una vez que finalice.

Motivación II

La utilización de subprogramas permite:

- *Reducir la complejidad del programa ("divide y vencerás").*
- *Eliminar código duplicado.*
- *Limitar los efectos de los cambios (aislar aspectos concretos).*
- *Ocultar detalles de implementación (p.ej. algoritmos complejos).*
- *Promover la reutilización de código*
- *Mejorar la legibilidad del código.*
- *Facilitar la portabilidad del código.*

Funciones

Desde el punto de vista matemático, una ***función*** es una operación que a partir de uno o mas valores (***argumentos***), produce un valor denominado ***resultado*** o ***valor de la función***.

Ejemplo:

$F(x) = x / (1+x^2)$

F es el nombre de la función y *x* es el argumento.

Para evaluar F se necesita darle valor a *x.*

Si $x = 3$, entonces

$F(3) = 3 / (1+3^2)$

$= 0.3$

Funciones

Una función puede tener varios argumentos, aunque el ***resultado*** o ***valor de la función*** es único.

Ejemplo:

$F1(x,y) = x * y$

$F2(x,y,z) = x^2 + y^2 + z^2$

Los lenguajes proveen una serie de funciones predefinidas que facilitan la tarea al programador. Por ejemplo, las incluídas en la librería matemática.

Para utilizarlas, debemos escribir

nombre de la función(argumentos)

sqrt(900.0)

la función *sqrt*, toma un argumento de tipo *double* y devuelve como resultado un valor de tipo *double*

- Hemos visto cuales son las motivaciones para la utilización de subprogramas (funciones en C++)
- Durante el curso, ya hemos utilizado funciones "provistas por el lenguaje", especialmente, las incluídas en la librería matemática
- Ahora tenemos la posibilidad de definir nuestras propias funciones.

Declaración de funciones

Declaración o Prototipo de la función

<tipo> nombre_func(<lista de parámetros>);

<tipo>: especifica el tipo del valor que devuelve la función (el tipo que tendrá el resultado)

nombre_func: el nombre de la función

<lista de parámetros> lista que indica cuantos argumentos y de que tipo se necesitan para utilizar la función.

La declaración de la función termina con un punto y coma.

Definición de funciones

Definición de la función: *describe el funcionamiento interno de la misma: es decir, el algoritmo que se aplica para calcular el valor que se debe devolver.*

Se distinguen dos partes en la definición

1. ***la cabecera de la función***: que es idéntica al prototipo
2. ***el cuerpo de la función:*** serie de declaraciones y sentencias que deben encerrarse entre llaves (estructura similar al cuerpo del programa principal)

Ejemplos

Prototipos

- double promedio(double v1, double v2);
- bool esNroPar(int nro);
- double sqrt(double x);
- char toupper(char c);
- bool esMayor(int a, int b);
- bool aprobado(int notaPromedio);

Ejemplos

```
double promedio(double v1, double v2)
  {return((v1+v2)/2.0);
  }

bool esNroPar(int nro)
  {if(nro%2 == 0)
     return(true);
   else
    return(false);
  }

bool esMayor(int a, int b)
  {return(a > b);
  }
```

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

double precioFinal(double costoUnitario, int cantItems);

int main()
{double costo, precio;
 int cantidad;

 cout << " Ingrese la cantidad de items a comprar: ";
 cin >> cantidad;
 cout << " Ingrese el costo por unidad: ";
 cin >> costo;

 precio = precioFinal(costo, cantidad);

 cout << " Total a Pagar (iva incluido): " << precio;

 return(0);
}

double precioFinal(double costoUnitario, int cantItems)
{const double IVA = 0.16;
 double subTotal;

 subTotal = costoUnitario * cantItems;

 return(subTotal + subTotal*IVA);
}
```

Parámetros Formales

Declaración de la función o prototipo

Llamado a la función

Cabecera de la función

Cuerpo de la función

Parámetros Formales y Parámetros Actuales

Que ocurre cuando hacemos:

```
precio = precioFinal(5.0, 8 );
```

```
double precioFinal(double costoUnidad,int cantItems)
{const double IVA = 0.16;
 double subTotal;

 subTotal = costoUnitario * cantItems;
  return(subTotal + subTotal*IVA);
}
```

Se copia el valor de los parámetros actuales en los formales

- Se copia el valor de los argumentos o parámetros actuales en los parámetros formales.
- Se transfiere el flujo de ejecución a la función invocada.
- Cuando la función termina su ejecución, devuelve el control al programa principal, que continuará en la línea siguiente a la invocación

Ejemplos I

```
int sumatoria(int inicio, int fin)
{int suma,i;
 suma = 0;
 for(i=inicio,i <= fin, i=i+1)
  suma = suma + i;
 return(suma);
}

int potencia(int n, int k)
{int acum,i;
 acum = 1;
 for(i=1,i <= k; i=i+1)
   acum = acum * n;
 return(acum);
}
```

Ejemplos II

```
int cuadrado(int n);
int cubo(int n);

int main ()
{int k = 3;
 cout << cuadrado(k);
 cout << cubo(k);
 return(0);
}
```

```
int cuadrado(int n)
   {return(n*n);
   }
```

```
int cubo(int n)
  {return(n*cuadrado(n));
  }
```

Ejercicios

- Defina una función que calcule el máximo de dos valores.
- Muestre como se puede utilizar para calcular el máximo de tres valores. Y con cuatro?
- Defina e implemente una función que informe si un año es bisiesto o no (*son bisiestos todos los años divisibles por 4, excluyendo los que sean divisibles por 100. Sin embargo, si lo son los divisibles por 400*)

Funciones que no retornan ningún valor

- Llamadas también "Procedimientos"
- En la declaración se indica:

```
void nombre_func(<lista de parámetros>);
```

- Útiles para mostrar información:

```
void limpiarPantalla();
void mostrarValores(int v1, int v2);
```

Ejemplo: imprimir k asteriscos
imprimir k simbolos
dibujo de pinos

Reglas de Alcance

El buen uso de la programación modular requiere que los módulos (funciones) sean independientes.

Esto se consigue intentando satisfacer dos condiciones:

- Cada módulo se diseña sin conocimiento del diseño de otros módulos
- La ejecución de un subprograma particular no tiene por que afectar a los valores de las variables de otros subprogramas.

Dado que se permite el anidamiento en el llamado a funciones, es necesario establecer mecanismos para evitar problemas con los identificadores definidos en varias partes del código.

Conceptos de **variables local**
variable global

Variables Locales

Son aquellas que se declaran en el cuerpo de la función. Solo son "visibles" o "usables" dentro de la función donde se han declarado.

Dos funciones diferentes, pueden utilizar los mismos nombres de variables sin "interferencias" ya que se refieren a posiciones diferentes de memoria.

```
int main ()
{int k = 3;
 cout << cuadrado(k);
 cout << k;
 return(0);
}
```

```
int cuadrado(int n)
{int k;
 k = n*n;
 return(k);
}
```

La variable *k* sigue valiendo 3 luego del llamado a *cuadrado(k)*

Constantes y Variables Globales

Si definimos una constante

const double PI = 3.14159

dentro de una función (incluyendo main), el valor *PI* será "local" a dicha función.

¿Como hacer para que sea visible en todas las funciones?

Declararla como constante global: no incluirla en el cuerpo de ninguna función.

También se pueden definir variables globales, aunque su utilización no está recomendada.

Ejemplo

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

const double PI=3.14159;
//calcula el area de un circulo de radio "radio"
double area(double radio);
// calcula el volumen de un circulo de radio "radio"
double volumen(double radio);
int main ()
{double r = 2.5;
 cout << area(r);
 cout << volumen(r);
 return(0);
}
```

```
double area(double radio)
  {return(PI*radio*radio);
  }
double volumen(double radio)
 {double tmp = pow(radio,3);
  return((4.0/3.0)*PI*tmp)
 }
```

Recomendaciones para la definición de funciones

- Escriba funciones como si fueran "cajas negras": el usuario debe saber QUE hace la función y no COMO lo hace.
- La declaración de la función y un comentario adecuado deben ser suficientes para saber como usarla.
- Todas las variables utilizadas en el cuerpo de la función deben estar definidas en el cuerpo de la función.

Ejemplos

- Dado el prototipo de función

  int miFuncion(int x, char y), y las variables a y b variables de tipo char, k de tipo int e inicializadas correctamente.

Indique cuales de las siguientes llamadas son válidas

b=F(a,'8') F(b,1) F(1,a) k=F(k+1,b) k=F(98,'k')

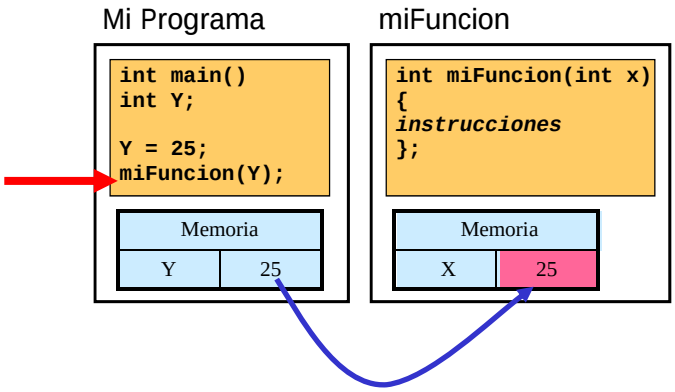
Paso de Parámetros a Funciones

- Paso de Parámetros por valor
- Paso de Parámetros por referencia
- Ejemplos de Utilización

Paso de Parámetros por Valor

- Es el mecanismo que hemos visto hasta ahora. El valor de los argumentos se copia en los parámetros formales.
- Los argumentos que se "pasan" por valor, también reciben el nombre de parámetros de entrada.
- Los parámetros formales funcionan como variables locales.
- Los argumentos pueden ser variables o valores literales *cuadrado(4)*, *cuadrado(n)*

Parámetros por Valor

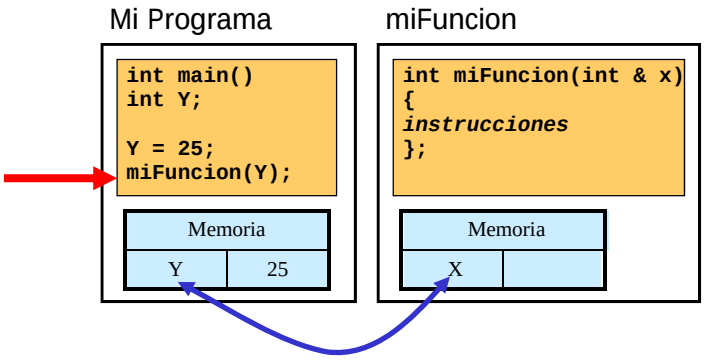


La función "miFuncion" trabaja sobre una copia del valor del parámetro actual

Paso de Parámetros por Referencia

- Un parámetro por referencia sirve para que la función comunique valores calculados al programa que la invocó.
- También se denominan parámetros de salida o entrada/ salida
- El valor de un parámetro por referencia puede ser leído y modificado dentro de la función y ésta modificación queda reflejada en el parámetro real cuando acaba la función.
- Para usar parámetros por referencia el identificador del parámetro formal debe estar precedido por el símbolo clave **&**
- El parámetro actual debe ser una variable, no una expresión

Parámetros por Referencia



- La variable X es un "alias" para la variable Y.
- X e Y hacen referencia a la misma posición de memoria
- Todos los cambios que hagamos dentro de la función *"miFuncion"* sobre la variable X se reflejarán en el parámetro actual correspondiente.

Ejemplo

```
#include <iostream>
#include <cmath>

const double PI=3.14159;

void AreaVol(double radio, double &area, double &volumen);
//calcula area y volumen de un circulo de radio "radio"
int main ()
   {double r = 2.5;
    double a, v;
    Area_Vol(r,a,v);
    cout << "Area:" << a << ", Volumen:" << v;
    return(0);
   }

void AreaVol(double radio, double &area, double &volumen)
   {area = PI * radio * radio;
    volumen = (4.0/3.0)* PI * pow(radio,3);
   }
```

	Parámetros por valor	Parámetros por referencia
Propósito del parámetro	Usado para transmitir un valor al cuerpo de la función	Como por valor, pero puede afectar a la variable actual
Forma del parámetro actual	Una expresión, variable o literal	Una variable
Tipo del parámetro actual	El mismo tipo que el parámetro formal	El mismo tipo que el parámetro formal
Tareas realizadas antes de ejecutar el cuerpo de la función	Creación de una variable local copia del valor del parámetro actual	Ninguna
Variable realmente accedida al usar el parámetro formal	La variable local	La variable que es el parámetro actual

Ejercicios

- Implemente una función que transforme un valor de temperatura expresado en grados Celsius a grados Farenheit (Fahrenheit = Celsius * 9/5 + 32).
- Implemente una función que calcule el factorial de un número dado.
- Implemente una función comb(n,m) que calcule el número combinatorio $\binom{m}{n} = \frac{m!}{n!(m-n)!}$
- Diseñe una función que devuelva la suma de los digitos del número natural suministrado como parámetro, así como el número de dígitos procesados.