Alfredo Vellido : www.lsi.upc.edu/~avellido

Fonaments d'Informàtica

Semana 6. Subprogramas

Objetivos

- Entender el **concepto de subprograma** y aprender a aplicarlo en la programación en C++.
- Distinguir entre **acciones** **y** **funciones**.
- Comprensión del **paso de parámetros** a acciones y funciones en C++

Subprogramas o procedimientos (*procedures*)

- **Objetivo:**

Encapsular: definir y asociar nombre y código a "pequeñas tareas" estandarizables dentro de un programa

Por ejemplo:

- máximo de tres números reales
- cálculo de la 'longitud' de un número (su número de cifras)
- cambio de moneda

Subprogramas o procedimientos

- **Ventajas:**
  - Mejora la **legibilidad** del programa
  - Ayuda a **estructurar mejor** los programas
  - **Evita repetir** el mismo código en el mismo programa
  - Se puede **reutilizar el código** en diferentes programas
  - Sólo hay que comprobar la corrección una vez
  - Permite **construir los programas por niveles de abstracción**, a base de resolver pequeños problemas y combinarlos

Ejemplo

```
...
float valor_absolut(float a)
{
 if ( a < 0.0 ) a = -a;
 return a;
}

int main () {
float mireal;
cin >> mireal;
mireal = valor_absolut(mireal);

[...]
```

Paso de parámetros en C++

- El **intercambio de información** entre un subprograma y la llamada al mismo se efectúa a través de **parámetros**. La info que un subprograma puede recibir y enviar son los PARÁMETROS FORMALES del subprograma.
  - **Parámetros formales**: aparecen en la cabecera de la definición de una acción / función. Se definen por tipo y nombre.
  - **Parámetros actuales**: Los parámetros concretos de llamada (instanciación de los formales)

```
...
float valor_absolut(float a)
{
if ( a < 0.0 ) a = -a;
return a;
}

int main () {
float mireal;
cin >> mireal;
mireal = valor_absolut(mireal);
...
```

Paso de parámetros en C++

- **Cuidado!!!**
  - El nombre del subprograma debe de ser correcto
  - El número de parámetros de llamada ha de ser **IGUAL** al número de parámetros formales
  - El **tipo** de los parámetros ha de corresponderse 1 a 1

```
...
float valor_absolut(float a)
{
if ( a < 0.0 ) a = -a;
return a;
}

int main () {
float mireal;
cin >> mireal;
mireal = valor_absolut(mireal);
...
```

Paso de parámetros en C++

Los parámetros formales se pueden clasificar en:

- **Entrada**. Un parámetro de entrada es un valor que el subprograma necesita para resolver el problema que le incumbe. **En este caso, el paso de parámetros es por valor.**
- **Salida**. Un parámetro de salida es una variable que no ha de tener necesariamente un valor inicial, y en la que el subprograma dejará un valor (resultado). **En este caso, el paso de parámetros se hará por referencia**, lo que se indica con el símbolo **'&'** delante del nombre del parámetro.

Paso de parámetros en C++

Los parámetros formales se pueden clasificar en:

- **Entrada/Salida**. Un parámetro de entrada/salida es una variable que ha de tener un valor inicial y que el subprograma puede modificar. **En este caso, el paso de parámetros es por referencia**, lo que se indica, de nuevo, con el símbolo **'&'** delante del nombre del parámetro.

Paso de parámetros en C++ (2)

- Los parámetros se le pueden pasar a un subprograma **por valor** (su valor: una copia: no es modificable), o **por referencia** (su dirección: es modificable), o **por referencia constante** (lo que se pasa es la dirección, pero se hace no modificable para evitar cambios indeseados).

CLASE DE PARÁMETRO	PASO EN C++	SINTAXIS
Entrada	Por valor Por referencia const.	tipo nombre const tipo& nombre
Salida	Por referencia	tipo& nombre
Entrada/Salida	Por referencia	tipo& nombre

Funciones

- Una función es un subprograma que **retorna un valor,** además de poder tener salida(s) y cero ó más entradas.
- Analogía con las funciones matemáticas: f(x,y)= 4xy+2x
- **Sintaxis**:

```
tipo_retorno nom_func (<lista_de_parametros>)
{<bloque de instrucciones>
  return <expresion>;
}
```

Subprogramas: acciones y funciones

Funciones

- **Sintaxis**:

```
tipo_retorno nom_func (<lista_de_parametros>)
  {<bloque de instrucciones>
    return <expresion>;
  }
```

lista de parámetros: **tipop1 nomp1, tipop2 nomp2, ...**

- **return** debe devolver un valor del tipo "**tipo_retorno**"
- **Llamada**: `a = nom_func (<lista_de_variables>);`

Funciones (2)

- Una **función** ***puede ser declarada en un punto del programa y definida en otro***. Si la definimos después de su llamada dentro del programa, deberemos haberla declarado antes.

```
[...]
tipo_retorno nom_func (<parámetros formales>);

// ... código ... int main() {…}

tipo_retorno nom_func (<parámetros formales>)
 {
 <bloque de instrucciones>
 return <expresion_retorno>
 }
```

Funciones (3)

- **Ejemplo 2**: Escribid una función que, dado un radio como parámetro, retorne la longitud de la circunferencia

```
double longit_circunf (double r)
  {
   return 2*M_PI*r;
  }
```

Ahora, usando esta fc., escribid un programa que, introducido un radio por teclado, nos saque por pantalla la longitud de la circunferencia …

Funciones (3b)

- **Ejemplo 1**: Escribid una función que calcule y retorne cuántas cifras tiene un número entero pasado como parámetro.

```
int longitud (int n)
   {int longit=0;
    while (n!=0)
    {longit++;
     n = n/10;}
    return longit;
   }
```

Acciones

- Una acción es un subprograma que retorna un valor **vacío (*void*).** No es completamente intercambiable con una función: Necesitamos poder modificar el valor de los parámetros: concepto de **parámetros de salida y entrada/salida.**
- **Son subprogramas que pueden modificar el estado del programa, acción o función desde donde se les llame.**

Subprogramas: acciones y funciones

Acciones

- **Sintaxis**:

```
void nombreacc (<parámetros formales>)
  {
   <bloque de instrucciones>
  }
```

- **Lista de parámetros**:
  - parámetros de **entrada**: `tipo nombre`
  - parámetros de **salida ó e/s**: `tipo& nombre`
- **Llamada**: `nombreacc (lista_de_variables);`

Acciones (2)

- Una **acción** puede ser ***declarada en un punto del programa y definida en otro***. Si la definimos después de su llamada dentro del programa, debemos haberla declarado antes.

- **Sintaxis**:

```
[...]
void nombreacc (<parámetros formales>);

// ... código, e.g. main () {...}

void nombreacc (<parámetros formales>)
{
 <bloque de instrucciones>
}
```

Acciones (3)

- **Ejemplo 1**: acción que intercambie los valores de dos enteros (entrada/salida)

```
void intercambiar (int& i, int& j)
{int k = i;
 i = j;
 j = k;}
```

Ahora, usando esta acción, escribid un programa que introducidos dos enteros por teclado, nos saque por pantalla sus valores intercambiados …

Acciones (3)

- **Ejemplo 2**: Dado un radio como parámetro, escribid una acción que calcule y devuelva al `main` el área/longitud del círculo/circunferencia

```
void area_long (double r, double& are, double& lon)
{ are = M_PI * pow(r,2);
  lon = 2 * M_PI * r;}
```

Subprogramas: acciones y funciones

Acciones (4)

- Escribir una **acción** para obtener y devolver a `main` las raíces de una ecuación de segundo grado (si estas no son reales, ha de indicarlo de alguna manera), habiendo recibido los coeficientes.

```
void raices_g2(float a,float b,float c,float& r1,float& r2,bool& sol)
{
 double numerador2;
 numerador2 = pow(b,2) - 4.0*a*c;

 if (numerador2 < 0) sol = false;
 else
  {r1 = (-b + sqrt(numerador2))/(2.0*a);
   r2 = (-b - sqrt(numerador2))/(2.0*a);
   sol= true;}
}
```

Subprogramas: acciones y funciones

Subprogramas: Más cosas sobre paso de parámetros ... Imaginad que tenemos definidas las siguientes función y acción (izq.) ... ¿Qué saldrá por pantalla?

```
[...]
int dobla1 (int x)
 { x=2*x; return x; }
void dobla2 (int& x)
 { x=2*x; }
```

```
int main(){
 int y = 5;
 int z = 6;
 z = dobla1(y); cout << z;
 z = dobla1(y*y); cout << z;
 z = dobla1(25); cout << z;
 dobla2(z); cout << z;
 dobla2(2*z); cout << z;
 dobla2(25); cout << z;
[...]
```

Subprogramas: acciones y funciones

Subprogramas: Más cosas sobre paso de parámetros ... Imaginad que tenemos definidas las siguientes función y acción ... ¿Qué saldrá por pantalla?

```
[...]
int dobla1 (int x)
 { x=2*x; return x; }
void dobla2 (int& x)
 { x=2*x; }
```

```
int main(){
 int y = 5;
 int z = 6;
 z = dobla1(y); cout << z;    //  ---> 10
 z = dobla1(y*y); cout << z; //  ---> 50
 z = dobla1(25); cout << z;  //  ---> 50
 dobla2(z); cout << z;       //  ---> 100
 dobla2(2*z); cout << z; //  ---> NO COMPILA
 dobla2(25); cout << z;  //  ---> NO COMPILA
[...]
```