ING. TÉCNICA DE TELECOMUNICACIONES: SISTEMAS ELECTRÓNICOS
UNIVERSITAT DE VALÈNCIA
Técnicas Informáticas - Curso 2006-2007 PRÁCTICA 5 (2 sesiones)

Objetivo de la práctica:

1. Aplicar el método de diseño descendente para obtener programas modulares.
2. Adquirir destreza en el paso de parámetros a subprogramas y reconocer la conveniencia de aplicar el tipo de paso por valor o por referencia.
3. Reforzar el concepto de ámbito de las variables al aplicarlo en el diseño de subprogramas.
4. Diferenciación entre funciones y procedimientos.

Conceptos básicos

La construcción de subprogramas es la consecuencia directa de aplicar la técnica de diseño descendente de programas. En esta técnica un problema complejo se va descomponiendo sucesivamente en problemas más sencillos cuya combinación resuelve el problema original, de forma que cada subproblema es convertible en un subprograma de forma directa y sencilla. A los subprogramas que devuelven un valor se les denomina **FUNCIONES.** Si el subprograma no devuelve ningún valor se le denomina **PROCEDIMIENTO**.

Existen dos lugares en el programa, donde se trabaja con un subprograma: 1) en la codificación del subprograma, donde indicaremos qué conjunto de instrucciones queremos que realice ese subprograma; 2) en la llamada al subprograma para que se ejecute en un determinado punto del programa principal (o dentro de otro subprograma).

Codificación de un subprograma

```
<tipo_que_devuelve> <nombre_del_subprograma>(lista_de_parámetros_separados por_coma)
{
      /*ZONA DE DECLARACION DE VARIABLES LOCALES*/
      /*ZONA DE CODIGO*/
      return <valor de retorno> //SI NO DEVUELVE NADA ESTA INSTRUCCIÓN SOBRA
}
```

Ejemplo de una función que calcula el máximo de dos números

```
float máximo(float num1, float num2) {
      float aux;
      if(num1 > num2)
            aux = num1;
      else
            aux = num2;
      return aux;
}
```

Ejemplo de una función que calcula la edad conocido el año de nacimiento:

```
int edad(int anyo_nac) {
      int tuedad;
      tuedad = 2006-anyo_nac;
      return tuedad;
}
```

Ejemplo de una procedimiento que imprime la tabla de multiplicar del número que se le pase como parámetro:

```
void tablamult(int num){
int i;
for (i=1; i<=10; i++)
  cout<<num<<"x"<<i<<"es"<<i*num<<endl;
}
```

Un procedimiento tiene a **`void`** como el tipo de dato que devuelve, es decir, no devuelve nada. Las funciones pueden formar parte de una expresión, es decir, se puede usar como un operando más de la expresión. Los procedimientos son sentencias y se usan como tales, es decir, al no devolver nada no se puede usar como parte derecha de una asignación o como un operando en una expresión.

Llamada de subprogramas

Una vez codificado el subprograma, éste se puede llamar desde cualquier punto del programa, bien en otro subprograma o bien en el subprograma principal.

`Ejemplo`

```
01 int main(){
02 float a, b, resul;
03 cout << "Introduce dos valores en punto flotante"\n";
04 cin >> a >> b;
05 resul = maximo(a,b);
06 tablamult(resul);
07 }
```

En la línea 05, se hace una llamada la función `maximo()`, pasándole dos parámetros a y b, en la línea 06 se llama a un procedimiento que imprime la tabla de multiplicar del número máximo.

Parámetros Formales y Reales

Por parámetros formales entendemos los parámetros que utilizamos en la codificación del subprograma. Por ejemplo, en la función `máximo` los parámetros `num1` y `num2` reciben el nombre de **parámetros formales.** En cambio cuando la función es llamada en la línea 05, allí se les da un valor concreto `a` y `b` y son denominados **parámetros reales**. El nombre de los parámetros formales y reales puede coincidir, esto no genera ninguna ambigüedad pues aparecen en trozos diferentes de programas.

Ámbito de las variables en subprogramas

Tanto las variables declaradas dentro del subprograma como los parámetros de un subprograma tienen un ámbito local al subprograma, es decir, su zona de validez son las instrucciones que pertenecen al código del subprograma. Ningún otro subprograma es capaz de acceder a las variables locales de un subprograma. A esto se le llama "encapsulación" de los datos y es beneficioso para el programador pues es una forma de evitar un mal uso de las variables. Cualquier intento de acceder a variables declaradas en otros subprogramas generará un mensaje de error al compilar.

Estructura de un programa

/*ZONA DE LOS INCLUDES*/
/*ZONA DE DECLARACIÓN DE COSTANTES*/
/*ZONA DE DECLARACION DE VARIABLES GLOBALES*/
/*ZONA DE DECLARACION DE LOS PROTOTIPOS DE LAS FUNCIONES*/
/*CODIFICACIÓN DE LA FUNCIÓN PRINCIPAL (MAIN)*/
/* CODIFICACIÓN DELRESTO DE FUNCIONES*/

Paso de parámetros por valor y paso por referencia

Cuando en la llamada a un subprograma pasamos a éste un **parámetro por valor**, estamos **pasando una copia** del valor a una variable local del subprograma. Esta copia **se destruye** cuando acaba su ámbito, es decir, cuando acaba la ejecución del subprograma y por tanto, al ser una copia, la variable que le dio el valor, no modifica su contenido. En el **paso por referencia**, en cambio, no pasamos una copia de la variable, sino que **pasamos la variable** en sí, de tal manera que cuando acaba la llamada del subprograma, la variable que fue pasada como parámetro real **conserva** las modificaciones realizadas por el subprograma.

```
void paso_parametros(int param1, int& param2)
{
      param1 = 5;
      param2 = 7;
}
```

El parámetro param1 está pasado por valor. El parámetro param2 está pasado por referencia. Observar que la diferencia está en que el de referencia tiene un '&' delante. Si hiciésemos la siguiente llamada:

```
int main()
{
      int x,y;
      x=8;
      y=9;
      paso_parametros(x,y);
      cout<< "x = " << x <<endl << "y = " << y<< endl;
      ...
}
```

La salida por pantalla sería:

```
x = 8
y = 7
```

Funciones recursivas

Existen situaciones 'curiosas' en donde el problema se resuelve sencillamente si una función se llama así misma dentro de su código. En matemáticas aparecen muchos ejemplos de funciones recursivas. Dos ejemplos son la función potencia y la función factorial. Las funciones recursivas consumen, en general, una gran cantidad de recursos, por lo que deben ser usadas sólo en aquellos casos en donde una implementación recursiva se adapte de forma intuitiva y natural al problema que queremos resolver. Siempre existe ante un algoritmo recursivo, una versión iterativa que realiza la misma tarea..

Recuerda: fact(n) = n * (n-1) * (n-2) * …. *2*1

es decir **fact(n) = n * fact(n-1)** **si n > 1**

fact(1)=1 **si n = 1**

Ejemplo de la función potencia implementada de forma recursiva:

```
int potencia(int base, int expo)
{
      int res;

      if (expo==1)
            return (base);
      else
      {     res = base * potencia(base, expo-1);
            return (res);
      }
}
```

EJERCICIOS

NOTA: En cada ejercicio: reflexiona y discute con tu compañero sobre papel en cuántos subprogramas vais a dividir cada ejercicio y que tarea realizará cada subprograma.

BLOQUE 1: Funciones .

Ejercicio1. Haz un programa que mediante un menú permita: 1) Calcular el área y perímetro de un cuadrado conocido su lado; 2) Calcular el área y la longitud de una circunferencia conocido su radio; 3) Calcular el área y el perímetro de un triangulo equilátero conocido su lado; 0) Acabar. El usuario introducirá la longitud del lado o el radio en cada caso. Se ha de verificar que se introducen valores positivos mayores que cero. El programa se deberá ejecutar hasta que el usuario introduzca la opción 0. Divide el problema en subprogramas implementando una función para cada una de figuras geométricas.

Ejercicio2. Haz un programa que mediante un menú permita realizar las siguientes operaciones: 1) Introducir dos números enteros; 2) Calcular su máximo; 3) Calcular el mínimo; 4) Calcular la media; 5) Calcular la suma de los dos; 6) Imprimir la tabla de multiplicar de los dos números; 0) Acabar. El programa se deberá ejecutar hasta que el usuario introduzca la opción 0. Divide el problema en subprogramas implementando una función para cada una de las operaciones matemáticas que se piden y que serán llamadas desde la función `main()`.

Ejercicio3. Al programa anterior añádele una nueva opción que calcule el máximo común divisor de los dos números.
Explicación: El máximo común divisor (mcd) de dos números `P` y `Q` es el mayor entero `D` que divide a ambos. Un algoritmo muy conocido para calcularlo es el de Euclides. Éste utiliza dos variables, que contienen inicialmente a cada uno de los números, y trata de hacer que su contenido sea el mismo. Para ello, irá restando la menor a la mayor hasta que ambas contengan el mismo valor. En dicho momento, el valor obtenido en cualquiera de ellas es el máximo común divisor de los dos números iniciales. Por ejemplo, si P = 18 y Q = 12, el algoritmo hará que P y Q vayan tomando los siguientes valores:

Inicialmente	P = 18 y Q = 12	(P > Q => P := P - Q)
Después	P = 6 y Q = 12	(Q > P => Q := Q -P)
Después	P = 6 y Q = 6	(P = Q => El mcd es 6)

BLOQUE 2: ¿Paso por valor o paso por referencia?

Ejercicio 4. Los siguientes programas son incorrectos o no dan el resultado esperado. Indica el por qué y escríbelos de la manera correcta.

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

void es_par(void)
{
 if((valor_leido % 2) == 0)
   cout<<"El valor"<<valor_leido<<"es par\n";
 else
  cout<<"El valor"<<valor_leido<<"es impar\n";
}

int main(void)
{
 int valor_leido;
 cout << "Introduce un valor";
 cin>>valor_leido;
 es_par();
 system("PAUSE");
 return 0;
}
```

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

void pide_carácter(char c)
{
  do
  {
    cout<<"Introduce una letra minuscula\n";
    cin >> c;
  }
  while((int(c)<int('a'))||((int(c)>int('z'));
}

int main(void)
{
  char mi_letra = '?';
  pide_carácter(mi_letra);
  cout<<"La letra leida es"<<mi_letra<<endl;
  return 0;
}
```

Ayuda: de qué tipo tendría que ser el paso de parámetros (valor o referencia) para la función.

Ejercicio5. Haz una función que pida al usuario su DNI y año de nacimiento. Éstas variables se le pasaran como parámetros a la función y ésta tendrá que asignarle los valores introducidos por el usuario desde teclado. ¿El paso de parámetros debería ser por valor o por referencia? ¿Por qué? Una vez leídos el DNI y el año de nacimiento, el programa imprimirá sus datos personales con el siguiente formato:

`Hola, tu DNI es 25403982, tu NIF 25403982E y tienes 20 años.`

Puedes utilizar la función `edad` vista en la página 1. Reutiliza el código de la práctica 3 que obtenía la letra del NIF a partir del DNI para crear una función que pasándole el DNI devuelva la letra. El programa se deberá ejecutar hasta que se introduzca la opción de acabar.

Ejercicio6. Diseñar un procedimiento que permute (intercambie) el valor de dos variables tipo carácter que se le pase como parámetros. ¿El paso de parámetros debería ser por valor o por referencia? ¿Por qué? Implementa la función `main()` que lea dos caracteres desde teclado, llame al procedimiento permutar e imprima los nuevos valores en pantalla. Prueba el paso por valor y por referencia para cerciorarte de que has implementado el tipo de paso de parámetros correcto.

Ejercicio7. Utiliza la función permutar que acabas de implementar en el ejercicio anterior para escribir un programa que lea una palabra formada por tres caracteres (leer carácter a carácter) y se muestre en pantalla la palabra con los caracteres en orden inverso. Por ejemplo: para la entrada luz debería imprimir zul.

Ejercicio8. Al ejecutar el programa anterior ¿qué ocurre con las palabras `oso, ana, asa`?. A estas palabras, que se leen igual de izquierda a derecha que de derechas a izquierdas, se les llama *palíndromas*. Haz un programa que lea una palabra de tres letras en el teclado y escriba en pantalla si es palíndroma o no. Reflexiona sobre cómo descompondrías el programa en subprogramas más sencillos y reutiliza funciones ya implementadas en los otros ejercicios. El programa se debe ejecutar hasta que el usuario introduzca la palabra `fin`.

Ejercicio9: Haz un programa que lea un número complejo (parte real y parte imaginaria) desde el teclado y calcule: 1) Su modulo r ; 2) Su fase θ; 3) Sus coordenadas en polares con el formato ($r\, e^{i\theta}$). Haz una función para leer desde el teclado y otras para las funciones matemáticas.
Ayuda: Si **z = a +b i** es el número complejo entonces su modulo **r** = sqrt(a^2+b^2) y su fase **θ** =arcotangente(b/a)

Ejercicio10: Haz un programa que lea dos números complejos (parte real y parte imaginaria) desde el teclado y calcule: 1) Su suma; 2) Su resta; 3) El producto; 4) Y el cociente. Haz una función para leer desde el teclado y otras para las funciones matemáticas.
Ayuda: para calcular el producto y el cociente mejor transformar los números complejos a coordenadas polares con la función del ejercicio anterior y realizar las operaciones en coordenadas polares.

BLOQUE 3: Funciones Recursivas

Ejercicio11. Sobre la implementación de la función potencia en la página 4, haz la traza del programa sobre papel y comprueba cuantas veces sería llamada la función potencia y cuáles serían los valores de los parámetros en cada llamada. ¿Cuál es el valor final devuelto para la llamada `potencia(2,3)`?. Implementa una versión iterativa de la función potencia.

Ejercicio12. Implementa una versión recursiva de la función factorial. Recuerda

fact(n) = n * fact(n-1) **si n > 1**
fact(1)=1 **si n = 1**