CFGM SISTEMAS MICROINFORMÁTICOS Y REDES
SISTEMAS OPERATIVOS MONOPUESTO
TEMA 4 - PRÁCTICA 02

Conceptos

Concepto y definición de una función.

Conforme los programas se diseñan para abordar problemas más complejos éstos se hacen más largos, es decir, requieren muchas más líneas de código. La experiencia ha demostrado que para los programadores los programas largos son difíciles de entender, depurar y de mantener a lo largo de su vida útil, aún siendo éstos los autores de los mismos.

Las anteriores dificultades dieron lugar a la concepción de nuevos modelos de programación que mejoraron la inteligibilidad, resultando en lo que conocemos como programación modular. La programación modular se basa en la estrategia de *divide y vencerás*. Esta estrategia promulga que para resolver un problema complejo la mejor aproximación será dividir éste en una serie de problemas más simples, más fáciles de abordar. Una vez resueltos estos problemas más sencillos, los resultados de los mismos se unirán de tal forma que nos ofrezcan una solución para el problema complejo.

Por tanto, la programación modular se basa en la división de programas complejos en un número de pequeños módulos. Cada módulo resolverá una parte pequeña del problema, quedando el problema complejo resuelto mediante la combinación de los resultados de cada uno de los módulos. Esta división permite que el programador se concentre en problemas más fácilmente abarcables, y se especialice en la resolución de los mismos, obviando los detalles del problema complejo que resolverá finalmente el programa, lo que redundará en la corrección, eficiencia y facilidad de depuración y mantenimiento de cada uno de los módulos por separado, y por tanto del programa final.

Los programas divididos en módulos suelen tener uno principal, que es el que coordina las llamadas a cada uno de los demás módulos. El programa principal al llamar a los módulos provoca que el hilo de la ejecución salte y pase por el código del módulo invocado. Cuando un módulo ha terminado su ejecución, provoca que el hilo de la ejecución vuelva al punto del programa donde dicho módulo fue invocado.

El lenguaje de programación C es un lenguaje que se basa en el paradigma modular. Las funciones en el lenguaje C representan módulos, que pueden parametrizarse, y devolver un valor de cualquier tipo soportado por el lenguaje. Las funciones en lenguaje C se definen empleando la siguiente estructura:

```
tipoDevuelto nombreFuncion (tipo1 parametro1,tipo2 parametro2, ...){
        codigoDeLaFuncion
}
```

donde:

- *nombreFuncion*: es el nombre de la función que se está definiendo. El nombre de la función ha de ser único en un programa, es decir, no puede coincidir con el nombre de una variable o función ya declarada ni con una palabra reservada.

- *tipoDevuelto*: es el tipo de dato del valor que devolverá la función como resultado de la misma. Se ha de destacar que una función sólo puede devolver un único valor de un tipo determinado. Las funciones que devuelven varios valores no se permiten. Cuando no se desea que una función devuelva un valor se indicará el tipo *void*.

- *tipoN parametroN*: representa el tipo y el nombre respectivamente del parámetro N-ésimo. Se ha de destacar que el número de parámetros que puede contener en su definición una función no es fijo, pudiendo desde no contener ningún parámetro hasta el número que sea necesario.

- *codigoDeLaFuncion*: es el código que contiene las instrucciones necesarias para que la función cumpla con su cometido.

Entre las funciones que componen un programa existe una función principal llamada *main*. El nombre de esta función es fijo, y su definición tiene normalmente la siguiente forma:

```
void main(void){
        ...
}
```

Esta función es la que contiene el código que se ejecutará directamente al iniciar el programa, y desde ella se invocarán las funciones en las que se haya descompuesto dicho programa.

Las definiciones de las funciones se dividen en dos partes fundamentales, siendo éstas, *cabecera* y *cuerpo*.

La cabecera de una función comprende la parte de la definición que indica la información necesaria para invocar la misma en cualquier programa. Esta información es el nombre de la función, el tipo de dato del valor que devuelve la misma y el número y tipo de argumentos que acepta. Por tanto la cabecera comprende desde que se inicia la definición hasta que se cierra el paréntesis después de la definición del último argumento.

El cuerpo de una función comprende la parte que indica qué procesamiento realizará la función, y por tanto comprende desde la llave de apertura del bloque de código, el código de la función y la llave de cierre del bloque de código asociado a la función.

Cuando un programa se compone de varias funciones se emplean dos maneras de organizar las mismas:

- Se incluye la definición de todas las funciones en un solo fichero de código. Esta opción es para proyectos pequeños, y no es recomendable, aunque por simplicidad a veces se recurre a ella.

- Cada definición se hace en un fichero fuente, coincidiendo el nombre de este fichero normalmente con el nombre de la función. Esto es una buena práctica de programación, aunque en pequeños proyectos no se suele emplear ya que requiere el uso de ficheros cabecera y guiones de compilación.

En el caso de que se use un único fichero de código fuente se ha de tener en cuenta que las definiciones de la funciones se pueden colocar en cualquier orden, siempre que éstas hayan sido declaradas en un punto anterior del fichero fuente al primer punto del código donde se invoca a las mismas.

Se ha de diferenciar entre el concepto de declaración y el concepto de definición. La definición de una función incluye la cabecera de la misma, así como el cuerpo de ésta. La declaración de una función se realiza para indicar cuáles son sus características para que pueda ser invocada, siendo éstas las que aparecen en la cabecera de la función, pudiéndose omitir de manera opcional el nombre de los argumentos. Una función puede ser declarada simplemente incluyendo en un fichero de código fuente la cabecera de la misma seguida de un punto y coma. Esta estructura se denomina *prototipo de la función*. Por ejemplo podemos declarar la función suma, que devuelve un tipo entero y recibe dos argumentos de tipo entero, con el siguiente prototipo:

```
int suma(int,int);

void main(void){
        ...
}
```

En un prototipo no es obligatorio indicar el nombre de los parámetros, y aunque no es habitual la inclusión de dichos nombres, se considera por varios autores una buena práctica de programación. En la presente actividad práctica y de desarrollo no incluiremos los nombres de los parámetros en los prototipos.

Normalmente se siguen dos estrategias para localizar la definición de las funciones en un solo fichero fuente:

1. Se declaran todas las funciones antes de la definición de la función principal *main*. Siempre se ha de observar que la definición de una función se ha realizado antes de la definición de las funciones que invocan a ésta.

2. Se define la función principal y a continuación se definen el resto de funciones. Esta opción es muy utilizada cuando se emplean estrategias descendentes en los procesos de ingeniería del software (se resuelven los problemas desde lo más general a lo más específico). La opción requiere que previamente a la definición de la función *main* se hayan declarado las funciones que se utilizan en el cuerpo de dicha función.

En general, y como buena práctica de programación, en primer lugar en el fichero fuente se incluirán las declaraciones de todas las funciones que aparecen en el mismo, excepto la función principal. A continuación, se define la función principal, y posteriormente se definirán el resto de funciones que han sido declaradas previamente.

Devolución de resultados por parte de las funciones.

Como ya se ha indicado en la sección anterior las funciones pueden devolver, como resultado de su procesamiento, un único valor del tipo determinado en su declaración.

En el lenguaje C se emplea la palabra reservada *return*, incluyendo a la derecha de dicha palabra reservada el valor que deseamos que devuelva la función. Esta palabra reservada se emplea dentro del cuerpo de las funciones de la siguiente forma:

```
return valor;
```

donde *valor* se sustituye con el valor que se desea devolver, pudiendo ser éste cualquier expresión, siempre y cuando ésta sea del mismo tipo que el tipo de dato devuelto indicado en la declaración de la función.

Invocación de funciones.

Una función definida en un programa se invoca según la siguiente estructura:

```
nombreFuncion(arg1,arg2,...);
```

donde *nombreFuncion* es el nombre de la función que se desea invocar, y *argN* es el argumento N-ésimo que se desea pasar a la función.

Se ha de distinguir claramente entre los conceptos de *argumentos formales* y *argumentos reales*. Los argumentos formales son los argumentos que se han indicado en la definición de la función, siendo el nombre que se les ha dado válido sólo en el ámbito de la función. Los argumentos reales son los argumentos que se pasan a una función cuando ésta es invocada. Los argumentos reales pueden ser variables, o cualquier otro tipo de expresión, y su nombre, en el caso de variables, no tiene por qué coincidir con el nombre dado a los argumentos formales.

Cuando se invoca a una función se hace, si no se indica lo contrario, una copia del valor de cada argumento real en el valor de la variable que representa cada argumento formal. A esta variedad de paso de argumentos se le llama *paso de parámetros por valor* o *paso de argumentos por copia*. Se ha de destacar que una vez terminada la ejecución de la función no se realiza una copia del valor de argumento formal al argumento real. Por tanto, si durante la ejecución de la función se modifica el valor de un argumento formal, el valor del argumento real no se verá alterado.

Una vez terminada la ejecución de la función, si ésta ha devuelto un valor, la llamada a dicha función es sustituida por el valor devuelto en la expresión o sentencia en la que participe dicha llamada.

Experimentos

E1. a) Analice el código del siguiente programa e identifique las funciones que aparecen en el mismo ¿Qué objetivo tiene cada función?

```
#include<stdio.h>
#include<math.h>

float cuadrado(float);
float negativo(float);
float raiz(float,float,float);
float resultado1(float,float,float);
float resultado2(float,float,float);

void main(void){
     float a,b,c;
     float r1,r2;

     printf("Calculadora de soluciones para ecuaciones de segundo grado\n");

     printf("Parametro a:");
     scanf("%f",&a);

     printf("Parametro b:");
     scanf("%f",&b);
```

```
     printf("Parametro c:");
     scanf("%f",&c);

     r1 = resultado1(a,b,c);
     r2 = resultado2(a,b,c);

     printf("Las soluciones a la ecuacion son x=%f y x=%f",r1,r2);
}

float cuadrado(float n){
      return n*n;
}

float negativo(float n){
      return -1.0 * n;
}

float raiz(float a,float b,float c){
      float b2;
      b2 = cuadrado(b);

      return sqrt( b2 - 4.0 * a * c);
}

float resultado1(float a,float b,float c){
      float negb;
      float r;
      negb = negativo(b);
      r = raiz(a,b,c);

      return (negb + r) / (2.0 * a);
}

float resultado2(float a,float b,float c){
      float negb;
      float r;
      negb = negativo(b);
      r = raiz(a,b,c);

      return (negb - r) / (2.0 * a);
}
```

b) Ejecute el siguiente código paso a paso y observe qué ocurre con el hilo de la ejecución cuando se invoca una función. Pruebe las diferentes acciones que ofrece el depurador (*Step over, Step into, Step out*) cuando se realice una invocación a una función.

E2. a) Analice el siguiente código e identifique el objetivo de la función *f*.

```
#include<stdio.h>

void f(int);

void main(void){
     int n = 10;

     printf("El valor de n antes de llamar a la funcion es %d\n",n);

     f(n);

     printf("El valor de n despues de llamar a la funcion es %d",n);

}

void f(int n){
     printf("El valor de n a la entrada de la funcion es %d\n",n);

     n = n * 2;

     printf("El valor de n al finalizar la funcion es %d\n",n);
}
```

b) Ejecute paso a paso el código y observe el valor que toma la variable *n* a través del proceso de ejecución ¿Por qué *n* toma esos valores?

Ejercicios

EJ1. Escriba un programa que pida al usuario dos números. El programa determinará cual de ellos es el mayor y cual el menor, mostrando por pantalla el resultado. Para ello cree un programa principal y dos funciones que acepten cada una dos argumentos, devolviendo la primera el valor del mayor de los argumentos, y la segunda devolviendo el menor de los dos argumentos.

EJ2. Escriba varias funciones para calcular datos de una circunferencia en función de su radio, siendo éstos:

- Diámetro de la circunferencia.
- Longitud de la circunferencia.
- Superficie de la circunferencia.

Así mismo, escriba un programa principal que solicite al usuario un radio determinado, invoque las distintas funciones para calcular los datos relacionados con la circunferencia y muestre dichos datos por pantalla.

Problemas

P1. Escriba una función que reciba tres enteros indicando una fecha, de forma que el primero será el día, el segundo el mes y el tercero el año, e imprima esta fecha con un determinado formato en la pantalla. Dicha función recibirá dos argumentos más, el primero indicará el carácter que se empleará de separador entre cada cifra (p.ej. ' / ' ó ' - '). El segundo será un número entero que indicará el orden en que se mostrarán las tres partes de la fecha, de forma tal que:

- 1: Indicará formato español dd/mm/aaaa (por ejemplo: 24/5/2005).
- 2: Indicará formato español abreviado dd/mm/aa (por ejemplo: 24/5/05).
- 3: Indicará formato inglés mm/dd/aaaa (por ejemplo: 5/24/2005).
- 4: Indicará formato inglés abreviado mm/dd/aa (por ejemplo: 5/24/05).
- 5: Indicará formato japonés aaaa/mm/dd (por ejemplo: 2005/5/24).
- 6: Indicará formato japonés abreviado aa/mm/dd (por ejemplo: 05/5/24).

Como ayuda cree una función que reciba el valor de un año y devuelva su valor abreviado (por ejemplo: al recibir 2005 devolverá 05). Además es recomendable que cree una serie de constantes simbólicas para representar más sencillamente los distintos valores numéricos empleados para indicar el formato de la fecha (por ejemplo: #define ESP 1).

Cree un programa principal que le pida al usuario el día, mes, año y carácter separador y le muestre un menú que le permite seleccionar el formato de fecha que desee. A continuación se mostrará por pantalla la fecha en el formato elegido.

P2. Escriba un programa que nos permita operar con tiempos expresados en horas, minutos y segundos. El programa recibirá dos cantidades de tiempo expresadas en horas, minutos y segundos, y devolverá como salida la suma de dichas cantidades expresada también en horas, minutos y segundos.

Pare ello el programa operará siempre con cantidades de tiempo expresadas en segundos, y por tanto dispondrá de al menos las siguientes funciones:

- Una función que reciba tres argumentos (horas, minutos y segundos) y devuelva la cantidad de segundos correspondiente.
- Una función que reciba una cantidad en segundos, y devuelva el valor entero de horas que representa.
- Una función que reciba una cantidad en segundos, y devuelva el valor entero de minutos que representa.
- Una función que reciba una cantidad en segundos e imprima por pantalla la cantidad de horas, minutos y segundos a que corresponde.

P3. Escriba un programa que funcione como una calculadora. El programa mostrará un menú en el que se nos permitirá elegir la operación matemática deseada (suma, resta, multiplicación, división) o salir del programa. Una vez elegida la opción el programa pedirá al usuario los dos valores, y devolverá el resultado de la operación, esperando la pulsación de una tecla para volver a mostrar el menú inicial. El programa deberá estar modularizado de forma que haya una función para cada una de las siguientes tareas:

- Salida del menú por pantalla y lectura de la opción elegida. La opción elegida se devolverá como valor de retorno.

- Petición de cada valor. La función tendrá un argumento que nos permita indicar si el valor es el primero o el segundo, para mostrar el mensaje adecuado, y nos devolverá como valor de retorno el valor introducido por el usuario.
- Procesamiento de la opción elegida. La función aceptará tres argumentos (la opción elegida y los dos valores) y devolverá como valor de retorno el resultado de la operación matemática elegida.
- Muestra por pantalla el resultado (el resultado será el argumento).
- La función principal, que organizará las llamadas al resto de funciones.