Tema 4

Funciones

En este capítulo presentamos las funciones como herramientas para el diseño de programas estructurados, fiables y fácilmente comprensibles y modificables. Al mismo tiempo se introduce la técnica de programación descendente (Top-Down) que tanta relación tiene con dichas herramientas.

Índice general

4.1. Justificación

Cuando un programa comienza a ser largo y complejo, como suele ocurrir con la mayoría de los problemas reales, no es apropiado tener un único texto con sentencias una tras otra. La razón es que no se comprende bien qué hace el programa debido a que se intenta abarcar toda la solución a la vez. Asimismo el programa se vuelve monolítico y difícil de modificar. También es corriente que aparezcan trozos de código muy similares entre sí repetidos a lo largo de todo el programa.

Para solucionar estos problemas y proporcionar otras ventajas adicionales a la programación, los lenguajes de alto nivel suelen disponer de una herramienta que permite estructurar el programa principal como compuesto de subprogramas (rutinas) que resuelven problemas parciales (subproblemas) del problema principal. A su vez cada uno de estos subproblemas puede estar resuelto por otra conjunción de problemas parciales, etc... Las funciones son mecanismos de estructuración que permiten ocultar los detalles de la solución de un problema y resolver una parte de dicho problema en otro lugar del código.

Junto con las herramientas que hemos llamado funciones aparecen varios conceptos importantes como los de variable local, ámbito, programación descendente, etc...

Supongamos que queremos calcular la media de votos que los distintos candidatos han obtenido en unas elecciones. Se pretende mejorar la entrada y salida de datos separándolas mediante una línea de de guiones:

```
------------------------------------------------------------------
Cuantos candidatos hay ? ... 3
------------------------------------------------------------------
Candidato numero: 1
Teclee el numero de votos para este candidato: ... 27
------------------------------------------------------------------
```

```
Candidato numero: 2
Teclee el numero de votos para este candidato: ... 42
-----------------------------------------------------------------
Candidato numero:3
Teclee el numero de votos para este candidato: ... 3
-----------------------------------------------------------------
Numero medio de votos conseguidos por candidato es: 2.4E+1
-----------------------------------------------------------------
```

Una primera aproximación a la solución de este problema es el siguiente código:

```
//
//  Autor:
//  Fecha: 28-10-93         version: 1.0
//-------------------------------------------------------
//  Programa ejemplo que calcula la media de votos
//  (Realizado sin funciones)
//-------------------------------------------------------

#include <iostream>

using namespace std;

const unsigned int LONGITUDLINEA = 65;
const unsigned int MAXIMONUMEROCANDIDATOS = 50;

void main() {

    unsigned int i;
    float media;
    unsigned int numeroDeVotos, totalDeVotos;
    unsigned int numeroDeCandidatos, indCand;

    for(i=1;i<= LONGITUDLINEA;i++){
        cout <<'-';
    }
    cout << endl;
```

```
cout << "Cuantos candidatos hay?";
cin >> numeroDeCandidatos;
cout << endl;
if (numeroDeCandidatos>MAXIMONUMEROCANDIDATOS){
    numeroDeCandidatos=MAXIMONUMEROCANDIDATOS;
}

totalDeVotos = 0;
for (indCand = 1; indCand <= numeroDeCandidatos;indCand++){
    for(i=1;i<= LONGITUDLINEA;i++){
        cout <<'-';
    }
    cout << endl;
    cout << "Candidato numero: ";
    cin >> numeroDeCandidatos;
    cout << endl;
    cout << "Teclee el numero de votos para este candidato ";
    cin >> numeroDeVotos;
    cout << endl;
    totalDeVotos = totalDeVotos + numeroDeVotos;
}
for(i=1;i<= LONGITUDLINEA;i++){
    cout <<'-';
}
cout << endl;

media = float(totalDeVotos) / float(numeroDeCandidatos);
cout << "Numero medio de votos conseguidos por candidato es: ";
cout << media;
cout << endl;
for(i=1;i<= LONGITUDLINEA;i++){
    cout <<'-';
}
cout << endl;
```

```
}
```

Como podemos observar, el código de esta solución se ve oscurecido por la aparición repetitiva de las sentencias que se usan para dibujar la línea. Si tenemos en cuenta que estas sentencias no son ni siquiera una parte fundamental de la solución del problema. que se mezclan con las otras y que repetimos código similar, llegaremos a la conclusión de que esta solución no es la mejor. Incluso podemos ver que si deseamos ampliar la E/S de datos y separar con líneas de guiones debemos repetir más veces el mismo código.

En su concepción más simple una función es una construcción que permite dar nombre a un conjunto de declaraciones y sentencias asociadas que se usan para resolver un subproblema dado. Usando funciones la solución es más corta, comprensible y fácilmente modificable.

```
//
//  Autor:
//  Fecha: 28-10-93         version: 2.0
//-------------------------------------------------------
//  Programa ejemplo que calcula la media de votos
//  (Realizado con funciones)
//-------------------------------------------------------
//

#include <iostream>

using namespace std;

const unsigned int LONGITUDLINEA = 65;
const unsigned int MAXIMONUMEROCANDIDATOS = 50;

void DibujarLinea() {

    int i;
    for(i=1;i<=LONGITUDLINEA;i++){
        cout <<'-';
    }
    cout << endl;
}
```

```
void main() {

    unsigned int i;
    float media;
    unsigned int numeroDeVotos, totalDeVotos;
    unsigned int numeroDeCandidatos, indCand;

    DibujarLinea();

    cout << "Cuantos candidatos hay?";
    cin >> numeroDeCandidatos;
    cout << endl;
    if (numeroDeCandidatos>MAXIMONUMEROCANDIDATOS){
        numeroDeCandidatos=MAXIMONUMEROCANDIDATOS;
    }
    totalDeVotos = 0;

    for (indCand = 1; indCand <= numeroDeCandidatos;indCand++){
        DibujarLinea();
        cout << "Candidato numero: ";
        cin >> numeroDeCandidatos;
        cout << endl;
        cout << "Teclee el numero de votos para este candidato ";
        cin >> numeroDeVotos;
        cout << endl;
        totalDeVotos = totalDeVotos + numeroDeVotos;
    }

    DibujarLinea();

    media = float(totalDeVotos) / float(numeroDeCandidatos);
    cout << "Numero medio de votos conseguidos por candidato es: ";
    cout << media;
    cout << endl;
    DibujarLinea();
```

```
}
```

4.2. Programación descendente

No repetir código no es la única razón para estructurar un programa usando funciones. Puesto que un subproblema puede codificarse como una función, un problema complejo puede subdividirse en subproblemas más simples, quienes a su vez pueden ser de nuevo subdivididos hasta llegar a la descripción de subproblemas muy simples que se puedan codificar como funciones escritas en C++.

La filosofía que acabamos de presentar se denomina refinamiento progresivo, programación descendente (*top-down*) o bien la técnica de divide y vencerás. La técnica de ocultar los detalles de implementación de una función se denomina abstracción procedural. De esta forma la estructura de la solución diseñada queda patente en el programa.

De hecho, todo los programas en C++ tienen una función básica, la función `main`.

4.3. Declaración y llamada a funciones

Como el resto de entidades en C++, las funciones deben declararse antes de ser usados. La declaración sirve para definir qué trabajo realiza la función y cómo lo lleva a cabo.

Toda función debe tener asociado un comentario respecto a lo que hace y el algoritmo que utiliza para ello.

El uso de funciones es un claro exponente de la programación estructurada, en el sentido que los errores se pueden localizar en la función que los produce, la solución al problema es más comprensible y clara, y en el futuro, con una buena documentación, será posible efectuar cambios en el programa que deben permitir que éste continue funcionando correctamente.

Una vez declarada, la función puede ser llamada (invocada) en el programa. Para ello basta incluir su nombre (y parámetros en el caso de que los tuviese) como si se tratara de una sentencia más. Cuando se alcanza la llamada, el control pasa a la primera sentencia de dicha función y cuando ésta acaba de ejecutar su cuerpo el control vuelve a la siguiente instrucción escrita tras la llamada a la función.

4.4. Localidad, ámbito y visibilidad

En una función existe un apartado destinado a la declaración de valores. En este lugar (entre la cabecera y el cuerpo de la función) se pueden declarar tipos, constantes y variables Todas las entidades declaradas dentro de una función se dice que son locales a dicha función, ya que no son visibles (referenciables o utilizables) en ningún otro lugar del programa (ni desde otras funciones).

C++ no permite la declaración de funciones dentro de funciones, es decir, no está permitido el anidamiento de funciones.

No obstante, una función puede usar entidades globales a todo el programa. Se dice que estas entidades son visibles en la función. Las entidades globales son las definidas antes del main del programa. Se dice entonces que las entidades así usadas son no locales a dicha función. El uso de entidades no locales en las funciones hace que un programa deje de estar estructurado al dar lugar a la aparición de los llamados efectos laterales (variables no locales modificadas en el interior de una función). Este tipo de situaciones debe eliminarse o bien restringirse al mínimo.

La vida de una entidad local es la misma que la de la función donde se la ha declarado. Por tanto una variable local se crea cada vez que se ejecuta la función y se destruye al terminar la ejecución de su cuerpo (por tanto una variable local no mantiene el último valor que se asignó en la anterior llamada a la función). Las variables locales no tienen valores iniciales por defecto; es el programador el encargado de proporcionárselos.

Se puede definir en una función un elemento que tenga el mismo nombre que otro global (por ejemplo una variable). En este caso se produce lo que se denomina ocultamiento. Dentro de la función nos referimos al elemento local y no al global con el mismo nombre. De hecho, el elemento global no es visible y no se puede tener acceso a él.

En general todos los lenguajes exigen declarar cualquier entidad antes de usarla (aunque no sea local). Estos lenguajes normalmente permiten diferir la especificación del cuerpo de una función y únicamente declarar su cabecera (nombre y lista de parámetros). Para esta situación, en C++ se distingue entre la declaración de una función y su definición.

La declaración de una función, también denominada *prototipo*, indica el nombre de la función y la lista de parámetros que recibe y acaba con un punto y coma. En la lista de parámetros se incluyen el tipo y el identificador de cada parámetro. Los parámetros están separados por comas.

La definición repite los datos de la declaración, pero en vez del punto y coma aparece un bloque se declaraciones e instrucciones con las tareas que ejecuta la función. En

general, las situaciones donde es necesario usar declaraciones previas aparecen con poca frecuencia:

```
// Solo la cabecera ha sido declarada.
// Esto permite al compilador comprobar la correccion de todas
// las llamadas al procedimiento DibujarLinea (solo necesita el
// nombre y la lista de parametros)
void DibujarLinea();

    ...

void P() {
        ...
        DibujarLinea();
        ...
}

    ...

//Aqui se realiza la definicion completa.
//La cabecera debe coincidir con la que se ha declarado previamente
void DibujarLinea() {
        ...
}
```

4.5. Funciones con parámetros

Una función puede resultar mucho más útil si se puede variar su funcionamiento de una llamada a otra. Esto se consigue añadiéndole parámetros. Por ejemplo, sería interesante poderle indicar el tamaño de la línea que queremos dibujar a la función que definimos anteriormente, `DibujarLinea`.

Esto se puede conseguir mediante la parametrización de la función. Se añade una lista de valores, los parámetros, que variarán el funcionamiento de la función dependiendo de cuáles sean los valores concretos con los que se llame a la función.

Esta lista de parámetros está entre paréntesis tras el nombre de la función. Cada uno de los parámetros esta precedido por su tipo y separado del siguiente parámetro por una

coma. Si la función no necesita parámetros, simplemente se escriben los paréntesis.

```
//
//  Autor:
//  Fecha: 28-10-93         version: 3.0
//-------------------------------------------------------
//  Programa ejemplo que calcula la media de votos
//  (Realizado con funciones con parametros)
//-------------------------------------------------------
//

#include <iostream>

using namespace std;

const unsigned int LONGITUDLINEA = 65;
const unsigned int MAXIMONUMEROCANDIDATOS = 50;

void DibujarLinea(int numGuiones) {

    int i;
    for(i=1;i<=numGuiones;i++){
        cout <<'-';
    }
    cout << endl;
}

void main() {

    unsigned int i;
    float media;
    unsigned int numeroDeVotos, totalDeVotos;
    unsigned int numeroDeCandidatos, indCand;
    unsigned int guiones;

    cout << "Introduzca la longitud de las lineas de separacion: " << endl;
    cin >> guiones;
```

```
    if (guiones > LONGITUDLINEA){
        guiones = LONGITUDLINEA;
    }
    DibujarLinea(guiones);

    cout << "Cuantos candidatos hay?";
    cin >> numeroDeCandidatos;
    cout << endl;
    if (numeroDeCandidatos > MAXIMONUMEROCANDIDATOS){
        numeroDeCandidatos = MAXIMONUMEROCANDIDATOS;
    }
    totalDeVotos = 0;

    for (indCand = 1; indCand <= numeroDeCandidatos;indCand++){
        DibujarLinea(guiones);
        cout << "Candidato numero: ";
        cin >> numeroDeCandidatos;
        cout << endl;
        cout << "Teclee el numero de votos para este candidato ";
        cin >> numeroDeVotos;
        cout << endl;
        totalDeVotos = totalDeVotos + numeroDeVotos;
    }

    DibujarLinea(guiones);

    media = float(totalDeVotos) / float(numeroDeCandidatos);
    cout << "Numero medio de votos conseguidos por candidato es: ";
    cout << media;
    cout << endl;
    DibujarLinea(guiones);
}
```

Los parámetros que se declaran en la cabecera de una función se denominan parámetros formales, mientras que los parámetros que aparecen en una llamada a la función se denominan parámetros reales o actuales.

En el ejemplo de la función `DibujaLinea(int numGuiones)`, el valor del parámetro formal númeroGuiones es consultado pero no es modificado.

Vamos a distinguir dos tipos de parámetros formales, parámetros por valor y parámetros por referencia.

4.5.1. Parámetros por valor

Los parámetros por valor se usan para proporcionar valores al interior de una función, y así modificar la tarea que lleva a cabo la función. Por esta razón se les suele denominar parámetros de entrada.

El parámetro real correspondiente a un parámetro por valor debe ser una expresión que se evalúe a un valor del tipo del parámetro formal. El valor de un parámetro por valor puede ser leído, como ya hemos visto, pero también puede ser modificado dentro del cuerpo de la función. Sin embargo, esta modificación no se conserva en el parámetro real al acabar la ejecución de la función. Cuando un parámetro real se pasa por valor, el parámetro formal es una copia y es sobre esta copia sobre la que se hacen las modificaciones. El parámetro real queda inalterado.

```
void dibujarLineas(int ancho, int numLineas){
    int nColumna, nFila;

    for (nFila=1; nFila <= numLineas; nFila++){
        for (nColumna=1; nColumna <= ancho; nColumna++){
            cout << "-" ;
        }
        cout << endl;
    }
}
```

La llamada `dibLineas(65,1)` genera una línea con 65 guiones. La llamada `dibLineas(0,5)` dibuja 5 líneas en blanco y por último la llamada `dibLineas(5,0)` no hace nada. No es una buen norma de estilo modificar el valor de una variable por valor para evitar errores si el programador supone que esta modificación se va a conservar. Se puede indicar que un parámetro es de sólo lectura anteponiendo la palabra clave `const` al tipo del parámetro formal. El compilador genera un error si detecta que se intenta modificar un parámetro constante.

```
void dibujarLineas(const int ancho, const int numLineas){
```

```
    int nColumna, nFila;

    for (nFila=1; nFila <= numLineas; nFila++){
        for (nColumna=1; nColumna <= ancho; nColumna++){
            cout << "-" ;
        }
        cout << endl;
    }
}
```

4.5.2. Parámetros por referencia (variables)

Un parámetro por referencia sirve para que la función comunique valores calculados al programa que la invocó. También se denominan parámetros de 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. A diferencia del paso de parámetros por valor, el parámetro formal no es una copia del real, sino una referencia a él. Por tanto, para el compilador la zona de memoria representada por el parámetro formal y actual es la misma y al terminar la ejecución de la función el parámetro actual conserva los cambios de valor.

Para usar parámetros variables el identificador del parámetro formal debe estar precedido por el símbolo clave `&` y el parámetro actual debe ser una variable, no una expresión cualquiera. Esto se debe a que el parámetro real debe conservar la modificación que se produzca en la ejecución de la función.

```
void raices(const double a, const double b, const double c,
            double &raiz1, double &raiz2)
{
// Se supone un discriminante positivo
    double discriminante;

    discriminante = sqrt( b*b-4.0*a*c );
    raiz1 = (-b + discriminante) / (2.0*a);
    raiz2 = (-b - discriminante) / (2.0*a);
}
```

Una posible llamada es `raices(1.0, 2.0, 1.0, a, b)`, donde a y b son dos variables de tipo `double`. Los parámetros formales reciben valores que se reflejarán en los

parámetros actuales usados en la llamada a la función. Los tipos de la variable formal y actual deben coincidir. Si se desea que un parámetro actual sea tanto de entrada como de salida entonces debe declararse como variable el parámetro formal.

```
void intercambiar(int & x, int &y) {
    int temp;

    temp = x;       // Se lee el valor de x
    x    = y;       // Se escribe un valor para x
    y    = temp;
}
```

Tras ejecutar

```
a = 5;
b = 8;
intercambiar( a, b );
```

el valor de la variable `a` es ahora `8` y el de `b` es `5`.

El ámbito de un parámetro formal se ciñe a la función en cuya cabecera se declara, igual que si se tratase de una variable local declarada en su zona de declaraciones. No existe ninguna relación entre los identificadores de los parámetros formales y actuales. La transferencia de valores se efectúa por posición en la lista de parámetros y no por el nombre que éstos tengan. Por ejemplo, el resultado de ejecutar `a=5; b=3; p(a,b);` es el mismo que el de ejecutar `b=5; a=3; p(b,a);` siempre que la función `p` no intente cambiar el valor de las variables que recibe como parámetros.

	Parámetros por valor	Parámetros por referencia
Propósito del parámetro	Usado para transmitir un valor al cuerpo de la función	Como por valor pero además puede afectar a la variable actual
Forma del parámetro actual	Una expresión	Una variable
Tipo del parámetro actual	El mismo tipo que el parámetro formal	El mismo tipo que el parámetro formal
Tareas llevadas a cabo antes de ejecutar el cuerpo de la función	Creación de una variable local y 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

4.6. Valores devueltos por las funciones

Una función puede llevar a cabo una serie de instrucciones y devolver información al programa que la llamó mediante el cambio de los valores de los parámetros por referencia. Además, una función también puede devolver un valor como resultado de su ejecución.

Para ello, en la declaración de la función se indica el tipo del valor que devuelve la función antes del nombre de la función. Para indicar que una función no devuelve ningún valor, se usa el tipo void.

Para devolver este valor se usa la instrucción `return`, devuelve el valor de la expresión que aparece tras ella.

Una función puede aparecer dondequiera que pueda usarse una sentencia. Si la función devuelve valor, la llamada a la función puede aparecer como operando de alguna expresión. El valor de la función se usa para calcular el valor total de la expresión.

```
double maximo(double x, double y){
    double resultado;

    if (x > y){
        resultado = x;
    }
    else{
        resultado = y;
```

```
    }
    return resultado; // Devolucion de valor al punto de llamada
}
```

El uso de esta función debe ser algo como:

```
x = 2.0 + maximo(3.0,p)/6.90,
```

de forma que la llamada devolverá el mayor valor entre `3.0` y el contenido de la variable `double p` y con dicho valor se evaluará el resto de la expresión.

Toda función debe ejecutar una sentencia `return`, pero puede contener en su cuerpo una más de una sentencia de este tipo. Tras la ejecución de una sentencia `return` acaba la ejecución de la función devolviendo el valor de la expresión que se indique tras ser evaluada. El valor resultante de dicha expresión debe ser del tipo declarado en la cabecera.

Es sintácticamente correcto colocar una sentencia `return` en cada una de las dos ramas de una sentencia condicional o dentro del cuerpo de un bucle, pero es una política que puede producir errores inesperados.

Si, por ejemplo, se olvida la sentencia `return` en una de las dos ramas de un `if`, el compilador no genera ningún mensaje de error, pero si, a la hora de ejecutarlo, el flujo del programa llega a esa rama, el programa acaba con un error. Para evitar ese tipo de situaciones, sólo permitiremos una única sentencia `return`, que será la última sentencia de la función. En el cuerpo de la función usaremos una variable local del mismo tipo que se va a devolver para almacenar temporalmente el resultado de la función.

También por cuestión de estilo, las funciones no deben asignar valores a variables globales y las que devuelven un valor no deben recibir parámetros variables ya que su tarea es calcular un valor a partir de otros valores.

4.7. Sobrecarga de funciones

En C++ se pueden definir varias funciones con el mismo nombre. Por ejemplo:

```
void f(int & i, int & j);
int  f(const float fl);
```

Esta definición múltiple se denomina *sobrecarga* y, para que sea correcta, cada una de las funciones tiene que tener una lista de parámetros diferente para que el compilador pueda saber a cuál de ellas se está haciendo referencia en cada caso. Por ejemplo, en:

```
{
   ...
  f(5, 4);
  d = f(38.45);
   ...
}
```

se puede saber que, en la primera instrucción se hace referencia a la primera función porque se pasan dos parámetros enteros. En el segundo caso se hace referencia a la segunda función porque sólo se pasa como parámetro un número real.

Esta característica del lenguaje ya ha sido usada. Concretamente, la operación >>, definida sobre la variable especial `cin` se define sobre diferentes tipos, como `int`, `char` o `float`, por ejemplo. Cuando el programa ejecuta una de estas instrucciones, sabe qué versión escoger por el tipo del parámetro que se le pasa. Igual ocurre con la operación << sobre `cout`.

4.8. Valores por defecto de los parámetros

Al definir una función se pueden establecer cuáles son los valores que tomarán por defecto los parámetros formales de la función si el usuario no los especifica. Por ejemplo, en la función:

```
void ff(const int i, const float fl = 0.0, const char cc = 'a')
{
   ...
}
```

establece como valores por defecto `0.0` para el segundo parámetro (`fl`) y `'a'` para el tercero (`cc`). Como se ve en el ejemplo, se pueden establecer valores por defecto sólo para unos cuantos parámetros, pero siempre han de ser los últimos.

A la hora de llamar a la función se puede no dar explícitamente valores a los parámetros que ya los tienen por defecto. Las siguientes llamadas son equivalentes:

```
{
   ...
  ff(5, 0.0, 'a');
  ff(5, 0.0);
  ff(5);
   ...
}
```

Se puede dejar de dar sólo algunos de los valores para los parámetros con valores por defecto pero, otra vez, han de ser los últimos.

Por supuesto, también se puede llamar a la función con otros valores distintos.

Ejercicios

1. Diseñar una función lógica que nos diga si un número es impar.

2. Escribe una función que tome 3 parámetros: dos de tipo natural y uno de tipo carácter. La función deberá sumar, restar, multiplicar o dividir los valores de los dos primeros parámetros dependiendo del código indicado en el tercer parámetro, y devolver el resultado.

3. El coeficiente binomial `c(n,k)` tiene el valor:
   $$\begin{cases} \frac{n(n-1)(n-2)...(n-k+1)}{k(k-1)(k-2)...(1)} & 1 \leq k \leq n \\ 1 & k = 0 \\ indefinido & k > n \end{cases}$$
   Por ejemplo, c(4,0)=1 y c(10,4)=210. Escriba una función con la siguiente cabecera

   ```
   int binomio(const int n, const int k);
   ```

   que devuelva el valor de `c(n,k)` o bien el valor 0 en caso de estar indefinido. Después modifique el programa para que utilice variables `double` en vez de `int`.

4. Defínase una función llamada `sumaDeDigitos` que reciba como parámetro un número `int` y devuelva la suma de los dígitos que lo componen. Componga con ella un programa donde se lea el número, se calcule esta suma y se presente en pantalla

el resultado. Un ejemplo de ejecución es, ante la entrada 102 resultado 3, para 1800 el resultado es 9, etc...

5. Codificar un programa que resuelva las ecuaciones siguientes simultáneas:

```
ax + by + c = 0      y        px + qy + r = 0
```

Realizar una función encargada de la solución de las ecuaciones que devuelva el valor de x e y en base al resto de parámetros de las ecuaciones. La función debe también tener un parámetro variable de tipo `bool` que tome el valor true en el caso de que las ecuaciones tengan solución y `false` en caso contrario. El programa debe leer los valores, resolver las ecuaciones usando la función diseñada e imprimir en pantalla los valores en caso de existir o un mensaje de error si no es posible resolverlas. Resolver con dicho módulo las ecuaciones siguientes:

```
3x + 2y - 7 = 0     3x + 2y -  7 = 0     3x + 2y -  7 = 0
9x - 5y + 1 = 0     9x + 6y - 21 = 0     9x + 6y - 20 = 0
```

6. Dado que no es aconsejable utilizar la igualdad entre número reales por ser dependiente de la precisión, diseñar una función que devuelva el valor true cuando sus dos argumentos reales son iguales para una tolerancia dada, y false cuando no lo sean. Utilizar para ello el siguiente prototipo:

```
bool IgualesR (const float v1, const float v2, const float tolerancia)
```

7. Diseña una función que, a partir de tres datos representando una fecha, por ejemplo, día, mes y año, compruebe si constituyen una fecha válida del siglo XX. Para ello, se tendrá en cuenta que los meses de Abril, Junio, Septiembre y Noviembre tienen 30 días; todos los demás meses, exceptuando Febrero, tienen 31 días. Febrero tiene 29 días si el año es bisiesto y 28 días si el año es no bisiesto. Por ejemplo:

Día	Mes	Año	Salida
23	11	1948	válido
31	11	1990	inválido
0	11	1990	inválido
1	13	1991	inválido
29	2	1991	inválido
25	12	1890	inválido

Nota: identificar los subproblemas del problema a ser resuelto, y resolver cada uno de ellos mediante una función en C++.

8. Diseña una función que muestre el calendario para un mes en el siguiente formato:

Lunes	Martes	Miércoles	Jueves	Viernes	Sábado	Domingo
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30						

Los datos para la función serán el número de días en el mes y el día de la semana en la que comienza ese mes. En el ejemplo mostrado en la figura, habría que pasarle los siguientes datos de entrada:
`MostrarCalendario (30, Domingo)`.

9. Diseña un programa que lea como datos de entrada una fecha dada por el mes y el año, y muestre el calendario para el mes en el formato del ejercicio anterior. Este problema se divide en los siguientes subproblemas:

   a) Leer los valores para mes y año, realizando una validación de dichos datos.

   b) Averiguar el día de la semana en el que comienza dicho mes.

   c) Averiguar cuántos días tiene dicho mes.

   d) Mostrar el calendario.

   Implementar las funciones necesarias en C++ para resolver este ejercicio.

10. Escribe un programa mediante una función que acepte un número de día, mes y año y lo visualice en el formato dd/mm/aa. Por ejemplo, para los valores 5, 11, 2001, deberá visualizarse 05/11/01.

11. Escribir un programa que permita el cálculo del mcd (Máximo Común Divisor) de dos números por el algoritmo de Euclides. (Dividir a entre b, se obtiene el cociente q y el resto r; si es 0, b es el mcd; si no, se divide b entre r, y así sucesivamente hasta encontrar un resto cero; el último divisor es el mcd.)

12. Escribir una función `potenciaentera(base, exponente)` que devuelva el valor de $base^{exponente}$. Por ejemplo, `potenciaentera(3,4) = 3 * 3 * 3 * 3`. Suponga

que `exponente` es un entero positivo, no cero, y `base` es un entero. La función `potenciaentera` deberá utilizar for para controlar el cálculo. No utilizar ninguna función de biblioteca.

13. Escribe una función que reciba un número natural L y dibuje un triángulo de asteriscos con base y altura L. Por ejemplo, para si L=5 debería dibujarse la figura adjunta.

```
    *
   * *
  * * *
 * * * *
* * * * *
```