

SISTEMAS INFORMÁTICOS

GITT, GIT, GIST Y GIEC

PRÁCTICA 2:

Operaciones con bits

Funciones

Sentencias de control

UAH, Departamento de Automática, ATC-SOL

OBJETIVOS

- Comprender las máscaras de bits
- Aprender a estructurar un programa en funciones
- Comprender y manejar sentencias de control de flujo

TEMPORIZACIÓN

- Inicio de la práctica: Semana del 2 de octubre
- Tiempo de desarrollo de la práctica: 2 semanas. Finalizarla antes del examen parcial 1
- Evaluación: ver fechas en el aula virtual

EJERCICIOS A REALIZAR

Cree un nuevo proyecto denominado *Practica2*, siguiendo los pasos explicados en la práctica 1. Después añada al proyecto el fichero *practica2.c* y complételo como se indica a continuación:

```
// practica2.c
#define _CRT_SECURE_NO_DEPRECATE
#include <stdio.h>
#include <stdlib.h>

// DECLARACIONES DE FUNCIONES
int menu(); // declaración de la función "menu"

int main ()
{
    // DEFINICIONES DE VARIABLES
    int opcion; // opción elegida del menú

    printf ("PRÁCTICA 2\n");
    printf ("=====\n");
    opcion = menu(); // llama/invoca a la función "menu"

    while (opcion != 10) // sentencia repetitiva
    {
        switch (opcion) // sentencia condicional
        {
            case 1:
                printf("Ejercicio 1: desplazar\n"); // llama a la función "printf"
                // Llamar a la función "desplazar"

                break;

            case 2:
                printf("Ejercicio 2: par o impar\n");
                // Llamar a la función "parImpar"

                break;

            case 3:
                printf("Ejercicio 3: operaciones lógicas y a nivel de bits\n");
                // Llamar a la función "operacionesLogicasBits"

                break;

            case 4:
                printf("Ejercicio 4: poner a cero un bit\n");
                // Llamar a la función "ponerACeroBit"

                break;

            case 5:
                printf("Ejercicio 5: factorial\n");
                // Llamar a la función "factorial"

                break;
```

```
case 6:
    printf("Ejercicio 6: numero de bits\n");
    // Llamar a la función "numeroBits"

    break;

case 7:
    printf("Ejercicio 7: cambiar bits\n");
    // Llamar a la función "cambiarBits"

    break;

case 8:
    printf("Ejercicio 8: intercambiar\n");
    // Llamar a la función "intercambiar"

    break;

case 9:
    printf("Ejercicio 9: media aritmetica\n");
    // Llamar a la función "mediaAritmetica"

    break;

default:
    printf ("ERROR: Opcion incorrecta.\n");
    break;
}
system("pause"); // llamar a la función "system"
opcion = menu(); // llamar a la función "menu"
}
```

// DEFINICIONES DE FUNCIONES

```
int menu() // definición de la función "menu"
{
    int op;
    do // sentencia repetitiva
    {
        system("cls");
        printf ("\n"
            "\t01.- Ejercicio 1\n"
            "\t02.- Ejercicio 2\n"
            "\t03.- Ejercicio 3\n"
            "\t04.- Ejercicio 4\n"
            "\t05.- Ejercicio 5\n"
            "\t06.- Ejercicio 6\n"
            "\t07.- Ejercicio 7\n"
            "\t08.- Ejercicio 8\n"
            "\t09.- Ejercicio 9\n"
            "\t10.- Salir\n"
            "\n"
            "Elija una opcion: ");

        scanf ("%d", &op); // llamar a la función "scanf"
```

```
    if (op < 1 || op > 10) // sentencia condicional
    {
        // printf("Opción no válida\n");
        printf("Opci%cn no v%clida\n", 0xA2, 0xA0);
        system("pause"); // detenerse hasta pulsar una tecla
    }
}
while (op < 1 || op > 10); // condición

return op;
}
```

Ejecute el programa (Ctrl + F5) y pruebe todas las opciones, incluso opciones no válidas como -1, 0, 11, etc.

```
01.- Ejercicio 1
02.- Ejercicio 2
03.- Ejercicio 3
04.- Ejercicio 4
05.- Ejercicio 5
06.- Ejercicio 6
07.- Ejercicio 7
08.- Ejercicio 8
09.- Ejercicio 9
10.- Salir
```

Elija una opcion:

Ejecute el programa paso a paso. Para ello, empiece pulsando la tecla F10. Continúe paso a paso (F10). Cuando llegue a la sentencia `opcion = menu();` continúe pulsando F10. Detenga la ejecución. Vuelva a empezar (F10). Continúe paso a paso (F10). Cuando llegue a la sentencia `opcion = menu();` continúe pulsando F11. ¿Cuál es la diferencia en F10 y F11?

Continúe paso a paso y escriba cuál es el flujo de ejecución que ha seguido el programa, después de haber elegido alguna de las opciones y, finalmente, salir.

Pruebe la diferencia que hay entre utilizar una u otra sentencia `printf` de las siguientes:

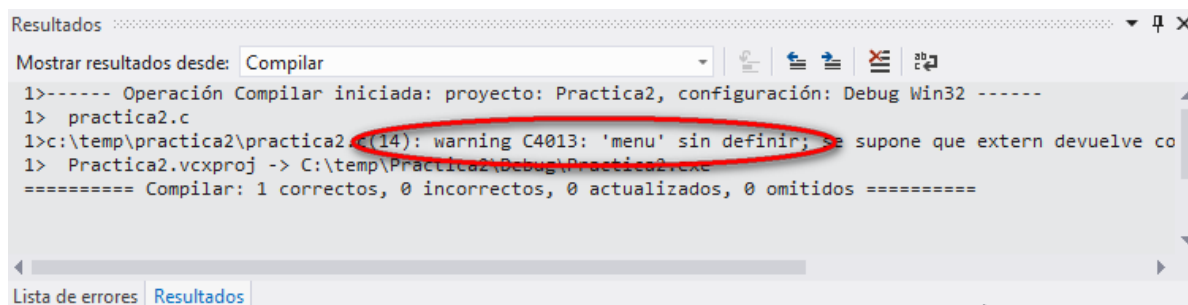
```
printf("Opción no válida\n");
printf("Opci%cn no v%clida\n", 0xA2, 0xA0);
```

Pruebe a quitar `\n` de la sentencia `printf` anterior que haya utilizado.

Pruebe a quitar (coméntela: `//`) la sentencia `system` siguiente:

```
system("pause"); // detenerse hasta pulsar una tecla
```

Comente (`//`) la línea `int menu();` que escribió antes de la función `main`. Compile de nuevo el programa y observe la ventana de resultados:



Resultados

Mostrar resultados desde: Compilar

```
1>----- Operación Compilar iniciada: proyecto: Practica2, configuración: Debug Win32 -----
1> practica2.c
1>c:\temp\practica2\practica2.c(14): warning C4013: 'menu' sin definir; se supone que extern devuelve co
1> Practica2.vcxproj -> C:\temp\Practica2\Debug\Practica2.exe
===== Compilar: 1 correctos, 0 incorrectos, 0 actualizados, 0 omitidos =====
```

Lista de errores Resultados

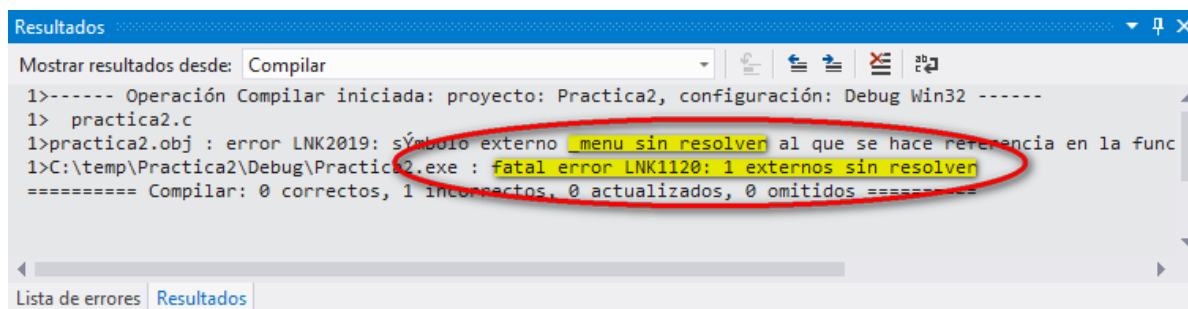
¿Qué significa ese aviso? ¿Cómo lo soluciona? Por analogía, ¿qué contiene el fichero `stdio.h`? Ábralo: clic con el botón secundario del ratón > abrir documento... Busque (menú Editar > Buscar...) las declaraciones de `printf` y `scanf`. ¿En qué líneas están?

Comente la línea `#include <stdio.h>`, compile de nuevo el programa y observe la ventana de resultados. ¿Solución?

A continuación, comente (`/* */`) la definición de la función `menu` como se indica a continuación:

```
/*
int menu() // definición de la función "menu"
{
    ...
}
*/
```

Compile de nuevo el programa y observe la ventana de resultados:



Resultados

Mostrar resultados desde: Compilar

```
1>----- Operación Compilar iniciada: proyecto: Practica2, configuración: Debug Win32 -----
1> practica2.c
1>practica2.obj : error LNK2019: símbolo externo 'menu' sin resolver al que se hace referencia en la func
1>C:\temp\Practica2\Debug\Practica2.exe : fatal error LNK1120: 1 externos sin resolver
===== Compilar: 0 correctos, 1 incorrectos, 0 actualizados, 0 omitidos =====
```

Lista de errores Resultados

¿Qué significa ese error? ¿Cómo lo soluciona?

Ejercicio 1

Escribir una función `desplazar` que visualice el valor de un número entero desplazado hacia la derecha `n` posiciones. La función recibirá como parámetro el número a desplazar y el valor del desplazamiento. La función devolverá el nuevo entero.

A modo de ejemplo, piense en algo que ya conoce: $x = \log(y)$; la función `log` recibe como parámetro el valor `y` (valor del cual se quiere calcular el logaritmo) y devuelve como resultado `x` (el logaritmo de `y`). ¿Quién proporciona a `log` el valor `y`? Usted. ¿A quién devuelve `log` el valor calculado? A usted. Piense ahora que usted es la función `main`, así sabrá lo que tiene que hacer `main`.

Según lo expuesto, en DEFINICIONES DE FUNCIONES añade la definición de la función `desplazar`:

```
int desplazar(int n, int nPos)
{
    int nDesplazado;
    /* añade el código que falta */
    return nDesplazado;
}
```

En DECLARACIONES DE FUNCIONES añade la declaración de la función `desplazar`:

```
int desplazar(int, int);
```

La función `main`, para el caso 1, debe preguntar el número a desplazar y el valor del desplazamiento. Como salida visualizará el nuevo entero.

```
case 1:
    printf("Ejercicio 1:\n"); // llama a la función "printf"
    // Solicitar datos introducidos por el teclado
    printf("Introduzca un numero: ");
    scanf("%d", &y);
    printf("Posiciones a desplazar a la derecha: ");
    scanf("%d",&z);
    // Llamar a la función "desplazar"
    x = desplazar(y, z);
    // Mostrar el resultado
    printf("El numero desplazado es: %d\n\n", x);
    break;
```

El código anterior utiliza tres variables, `x`, `y` y `z`, que tenemos que definir en DEFINICIONES DE VARIABLES de `main`:

```
int main ()
{
    // DEFINICIONES DE VARIABLES
    int opcion; // opción elegida del menú
    int x = 0, y = 0, z = 0;
```

Compile el programa y pruebe el caso 1.

```
Elija una opcion: 1
Ejercicio 1: desplazar
Introduzca un numero: 32
Posiciones a desplazar a la derecha: 3
El numero desplazado es: 4
```

Para el resto de los ejercicios que se proponen a continuación tiene que seguir los mismos pasos, incluso puede reutilizar las variables de `main` que le sean válidas.

Ejercicio 2

Escribir una función `parImpar` que compruebe si un número entero es par o impar. La función recibirá como parámetro el número y devolverá 0 si es par y 1 si es impar.

En DEFINICIONES DE FUNCIONES añade la definición de la función `parImpar`:

```
int parImpar(int n)
{
    int resultado;
    /* añade el código que falta */
    return resultado;
}
```

En DECLARACIONES DE FUNCIONES añade la declaración de la función `parImpar`:

```
int parImpar(int);
```

La función `main`, para el caso 2, debe preguntar el número a verificar. Como salida visualizará un mensaje indicando si el número es par o impar.

```
case 2:
    printf("Ejercicio 2: par o impar\n");
    // Solicitar datos introducidos por el teclado
    printf("Introduzca un numero: ");
    scanf("%d",&y);
    // Llamar a la función "parImpar"
    x = parImpar(y);
    // Mostrar el resultado
    if (x != 0)
        printf("El numero %d es impar\n\n", y);
    else
        printf("El numero %d es par\n\n", y);
    break;
```

El código anterior utiliza dos variables, `x` e `y` de tipo `int` que ya tenemos definidas en DEFINICIONES DE VARIABLES de `main`:

```
int main ()
{
    // DEFINICIONES DE VARIABLES
    int opcion; // opción elegida del menú
    int x = 0, y = 0, z = 0;
```

Compile el programa y pruebe el caso 2.

```
Elija una opcion: 2
Ejercicio 2: par o impar
Introduzca un numero: 33
El numero 33 es impar
```

Ejercicio 3

Escribir una función `operacionesLogicasBits` que permita verificar los resultados de las operaciones especificadas en el código mostrado a continuación. Primero, escriba en un folio los resultados que usted calcule al ejecutar mentalmente ese código. Después ejecute el programa paso a paso (F10) y verifique los resultados. Después, detrás de cada sentencia `r? = ...` añada un comentario con el valor correcto.

```
void operacionesLogicasBits()
{
    int a = 8, b = 0, c = 15, d = 93, e, r1, r2, r3, r4, r5, r6;

    r1 = a && b || c && !d;
    r2 = !a || b && c || d;
    r3 = a < b || !c > d;
    r4 = a + b > d - c;
    r5 = a && b && !c || !(a && b) && c;

    a = 0x12; b = 0x56; c = 0x9a ; d = 0x0f ; e = 0360;

    r1 = a & b | c;
    r2 = c & 0177;
    r3 = ~a | b ^ c;
    r4 = e & c;
    r5 = r4 & ~077;
    r6 = (a & d) << 4 | (a & e) >> 4;
}
```

En DECLARACIONES DE FUNCIONES añade la declaración de la función `operacionesLogicasBits`:

```
void operacionesLogicasBits();
```

La función `main`, para el caso 3, sólo tiene que llamar a la función `operacionesLogicasBits`.

```
case 3:
    printf("Ejercicio 3: operaciones logicas y a nivel de bits\n");
    // Llamar a la función "operacionesLogicasBits"
    operacionesLogicasBits();
    break;
```

Ejercicio 4

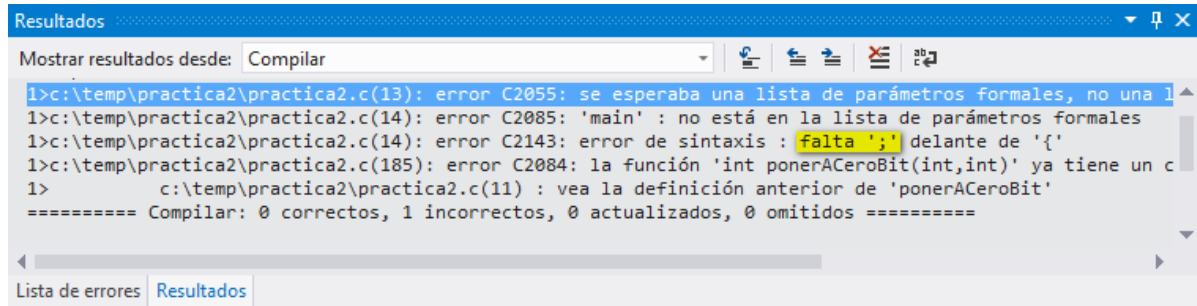
Escribir una función `ponerACeroBit` que ponga a cero un bit en una posición concreta de un número entero. La función recibirá como parámetros el número y la posición del bit a resetear. La función devolverá el nuevo valor obtenido.

Por ejemplo, si introducimos el número 28 (en binario 11100b) y el bit que queremos cambiar es el de la posición 3 (la posición más a la derecha es la 0), el resultado en binario será 10100b (20).

En DEFINICIONES DE FUNCIONES añade la definición de la función `ponerACeroBit`:

```
int ponerACeroBit(int n, int nPosBit)
{
    int resultado, mascara;
    mascara = 1 << nPosBit;
    /* añade el código que falta */
    return resultado;
}
```


En DECLARACIONES DE FUNCIONES añada la declaración de la función `ponerACeroBit`. Cuando añada esta declaración, haga una prueba: omita el punto y coma que tiene que poner al final de la declaración, compile el programa y observe la ventana de resultados:



Observará que el compilador nos informa de que falta un punto y coma, pero no indica exactamente dónde, porque no puede saberlo; al no poner un punto y coma al final de la declaración de la función, quizás esperaba encontrar una `{`, ya que también podemos definir las funciones antes de `main` (en este caso, no sería necesaria la declaración).

```
int ponerACeroBit(int n, int nPosBit)
{
    ...
}
```

La función `main`, para el caso 4, debe preguntar el número entero a modificar y por la posición del bit a modificar. Como salida visualizará el número entero modificado. Este apartado se hace análogamente a como hicimos el caso 1.

Compile el programa y pruebe el caso 4.

```
Elija una opcion: 4
Ejercicio 4: poner a cero un bit
Introduzca un numero: 28
Posicion del bit a poner a cero: 3
El numero modificado es: 20
```

Ejercicio 5

Escribir una función `factorial` que calcule el factorial de un número entero positivo entre 0 y 16. La función recibirá como parámetro el número y devolverá como resultado el factorial, o el valor de -1 si el número no está entre 0 y 16.

En DEFINICIONES DE FUNCIONES añada la definición de la función `factorial`:

```
long factorial(long n)
{
    long i = 0, factorial = 1;
    if (n < 0 || n > 16) return -1;
    /* añada el código que falta */
    return factorial;
}
```

En DECLARACIONES DE FUNCIONES añade la declaración de la función `factorial`.

La función `main`, para el caso 5, debe preguntar el número del cual se quiere calcular su factorial y verificar que es positivo. Como salida visualizará el factorial del número o un mensaje "No es posible calcular el factorial" si la función devuelve el valor -1.

```
case 5:
    printf("Ejercicio 5: factorial\n");
    do
    {
        printf("Introduzca un numero entero positivo: ");
        scanf("%ld", &yl);
    }
    while (yl < 0);
    // Llamar a la función "factorial"
    xl = factorial(yl);
    // Mostrar el resultado
    if (/* añade el código que falta */)
        printf("No es posible calcular el factorial\n");
    else
        printf("El factorial de %ld es: %ld\n\n", yl, xl);
    break;
```

El código anterior utiliza dos variables, `xl` e `yl` de tipo `long`. Añádalas a las que ya tenemos definidas en DEFINICIONES DE VARIABLES de `main`.

Compile el programa y pruebe el caso 5.

```
Elija una opcion: 5
Ejercicio 5: factorial
Introduzca un numero entero positivo: 99
No es posible calcular el factorial
Presione una tecla para continuar . . .
```

```
Elija una opcion: 5
Ejercicio 5: factorial
Introduzca un numero entero positivo: -3
Introduzca un numero entero positivo: 6
El factorial de 6 es: 720
```

Ejercicio 6

Escribir una función `numeroBits` que pregunte al usuario un número entero positivo y devuelva el número mínimo de bits necesario para representar ese número en binario. Por ejemplo, dado el número 12, necesitamos 4 bits para representarlo (porque $2^3 < 12$ y $2^4 > 12$).

En DEFINICIONES DE FUNCIONES añade la definición de la función `numeroBits`:

```
int numeroBits(int n)
{
    int dosElevadoN = 2, nbits = 1; // 2^1 = 2
    while (dosElevadoN <= n)
        /* añade el código que falta */
}
```

En DECLARACIONES DE FUNCIONES añada la declaración de la función `numeroBits`.

La función `main`, para el caso 6, debe preguntar el número del cual se quiere calcular su mínimo número de bits y verificar que es positivo. Como salida visualizará el número y el número mínimo de bits necesario. Este apartado se hace análogamente a como hicimos el caso 5.

Compile el programa y pruebe el caso 6.

```
Elija una opcion: 6
Ejercicio 6: numero de bits
Introduzca un numero entero positivo: -1
Introduzca un numero entero positivo: 8
El numero de bits para representar 8 es: 4
```

Ejercicio 7

Escribir una función `cambiarBits` que aplique a un número entero positivo la transformación siguiente: invertir (cambiando 1 por 0 y viceversa) los n bits correspondientes a ese número que estén a partir de la posición p (la posición 0 es la posición más a la derecha). No se podrá cambiar más bits que los que hay entre las posiciones p y 0 ($p+1$). La función devolverá el número transformado o -1 si el número de bits a cambiar no es válido.

Por ejemplo, si introducimos el número 28 (en binario 11100b) y el número de bits que queremos invertir es 3 a partir de la posición 3, el resultado en binario será 10010b (18).

En DEFINICIONES DE FUNCIONES añada la definición de la función `cambiarBits`.

En DECLARACIONES DE FUNCIONES añada la declaración de la función `cambiarBits`:

```
int cambiarBits(int, int, int);
```

La función `main`, para el caso 7, debe preguntar el número entero a modificar, la posición a partir de la cual se van a invertir los bits y el número de bits a invertir. Como salida visualizará el número modificado o un mensaje "Número de bits no valido" si la función devuelve el valor -1. Este apartado se hace análogamente a como hicimos el caso 4, 5, 6, etc.

Si necesita declarar alguna variable, hágalo.

Compile el programa y pruebe el caso 7.

```
Elija una opcion: 7
Ejercicio 7: cambiar bits
Introduzca un numero entero positivo: -1
Introduzca un numero entero positivo: 28
Posicion del bit inicial a invertir: 3
Numero de bits a invertir: 3
El numero modificado es: 18
Presione una tecla para continuar . . .
```

```
Elija una opcion: 7
Ejercicio 7: cambiar bits
Introduzca un numero entero positivo: 28
```

Posicion del bit inicial a invertir: 3
Numero de bits a invertir: 5
Numero de bits no valido

Ejercicio 8

Escribir una función `intercambiar` que intercambie dos valores con decimales. La función recibirá como parámetros los dos números de tipo **double** y no devolverá nada.

En DEFINICIONES DE FUNCIONES añade la definición de la función `intercambiar`.

En DECLARACIONES DE FUNCIONES añade la declaración de la función `intercambiar`:

```
void intercambiar(double *, double *);
```

La función `main`, para el caso 8, debe preguntar por los dos valores de tipo **double** y mostrará los valores intercambiados.

Si necesita declarar alguna variable, hágalo.

Compile el programa y pruebe el caso 8.

```
Elija una opcion: 8
Ejercicio 8: intercambiar
Introduzca el valor de d1: 34.5
Introduzca el valor de d2: -3.14
Valores intercambiados: d1 = -3.14, d2 = 34.5
```

Ejercicio 9

Escribir una función `mediaAritmetica` que calcule la media aritmética de 4 números. Los números pueden tener decimales. La función devolverá la media aritmética.

La función `main`, para el caso 9, debe preguntar por los valores de tipo **double** y mostrar la media aritmética de los mismos.

Compile el programa y pruebe el caso 9.

```
Elija una opcion: 9
Ejercicio 9: media aritmetica
Numero: 3.5
Numero: 4.5
Numero: 5
Numero: 6
Media: 4.75
```

EJERCICIOS OPCIONALES

Constantes, variables y operadores

NOTA: En esta sección se presenta el código de varios programas y se mencionan diversos mensajes de advertencia (warnings) que pueden aparecer o no en función del compilador que se use. La materia expuesta en los mismos, también formará parte de los exámenes.

Ejercicio 10

En el siguiente programa se muestran diferentes aspectos del trabajo con variables y constantes de tipo carácter. Depure el programa y conteste a las siguientes preguntas.

```
#include <stdio.h>

int main ()
{
    unsigned char car1;
    char car2;

    printf ("Constantes y variables de tipo caracter\n");

    car1 = 0x35 + 96 - ' ';           /* (a) */
    car2 = "z";                      /* (b) */

    printf ("\nEjemplo 1:\n");
    printf ("Caracteres:\t%c\t%c\n", car1, car2);
    printf ("Valores:\t%d\t%d\n", car1, car2);

    car1 = 0x9C;                     /* (c) */
    car2 = car1;

    printf ("\nEjemplo 2:\n");
    printf ("Caracteres:\t%c\t%c\n", car1, car2); /* (d) */
    printf ("Valores:\t%d\t%d\n", car1, car2); /* (e) */

    return 0;
}
```

- Al compilar se da un error de compilación en la línea indicada como (b). ¿Cuál es la causa del error? Indique cual sería la forma correcta de escribir dicha línea de código y corrija la en el mismo.
- ¿Qué valor se asignará a car1 en la línea indicada como (a)? ¿Qué carácter ASCII representa dicho valor?
- ¿Qué carácter ASCII estará almacenado en la variable car1 cuando se ejecute la línea indicada como (c)? (Consulte la tabla ASCII)
- ¿Se obtiene el resultado previsto al ejecutarse la sentencia (d)? ¿Cómo es que al ejecutarse la sentencia (e) las variables car1 y car2 parecen no contener lo mismo? Explique qué ocurre.

Ejercicio 11

El siguiente programa muestra aspectos del trabajo con constantes y variables enteras y de coma flotante. Depure el programa y conteste a las siguientes preguntas.

```
#include <stdio.h>

int main ()
{
    char car1, car2;
    unsigned int ent1;
    short int ent2;
    long lon;
    float real;
    double dob;

    printf ("Constantes y variables de tipo entero y real\n");

    ent1 = 300 * 1.5 + 0x14;          /* (a) */
    ent2 = 301 * 1.5 + 0x14;          /* (b) */

    printf ("\nEjemplo 1:\n");
    printf ("ent1 = %d\tent2 = %d\n", ent1, ent2);    /* (c) */

    car1 = ent1;                      /* (d) */
    car2 = ent2 - ent1;               /* (e) */

    printf ("\nEjemplo 2:\n");
    printf ("car1 = %d\tcar2 = %d\n", car1, car2);

    lon = 200000;                     /* (f) */
    ent1 = 200000;                    /* (g) */
    ent2 = 200000;                    /* (h) */

    printf ("\nEjemplo 3:\n");
    printf ("lon ocupa %d bytes.\t", sizeof(lon));
    printf ("Su valor es: %ld\n", lon);
    printf ("ent1 ocupa %d bytes.\t", sizeof(ent1));
    printf ("Su valor es: %d\n", ent1);
    printf ("ent2 ocupa %d bytes.\t", sizeof(ent2));
    printf ("Su valor es: %d\n", ent2);

    real = lon;                       /* (i) */

    printf ("\nEjemplo 4:\n");
    printf ("real ocupa %d bytes.\t", sizeof(real));
    printf ("Su valor es: %f\n", real);

    real = 12345.6789;                /* (j) */
    dob = 12345.6789;                 /* (k) */

    printf ("\nEjemplo 5:\n");
    printf ("real = %f\n", real);
    printf ("dob = %f\n", dob);

    return 0;
}
```

1. La compilación genera dos *warnings* en las líneas (a) y (b):

- (a): '=' : conversión de 'double' a 'unsigned int'; posible pérdida de datos
- (b): '=' : conversión de 'double' a 'short'; posible pérdida de datos

¿Cuál es el motivo? Calcular los valores de las expresiones a la derecha del operador de asignación en ambos casos empleando, mediante una depuración, una ventana de *Inmediato* si se desea. ¿Imprime la línea (c) el resultado esperado? Si la respuesta es negativa, explicar lo ocurrido.

2. ¿De qué manera se pueden evitar el *warning* en la línea (a) del apartado anterior, suponiendo que ent1 debe seguir siendo una variable de tipo int? Corrija el código y compruebe que ya no se genera el *warning*.
3. Las líneas (d) y (e) no generan ningún *warning*. ¿Qué valores toman las variables `car1` y `car2`? Trate de explicar por qué se asigna dicho valor a `car1`, y en consecuencia, qué hace el compilador al realizar este tipo de asignaciones.
4. ¿De qué tipo es la constante en la línea (f)? ¿Se realiza algún tipo de conversión? ¿Cuál? ¿Y en las líneas (g) y (h)? ¿Por qué hay un *warning* en la línea (h)?

(h): '=' : truncamiento de 'int' a 'short'

Explique el valor que toma la variable `ent2`.

5. ¿Por qué se genera un *warning* en la línea (i)?

(i): '=' : conversión de 'long' a 'float'; posible pérdida de datos

Ponga un ejemplo en el que se ponga de manifiesto el motivo de dicho *warning*. ¿De qué tipo capaz de almacenar números reales tendría que ser la variable real para que no se generase dicho *warning*? Compruébelo.

6. ¿Por qué sucede el *warning* de la línea (j)?

(j): '=' : truncamiento de 'double' a 'float'

Explique el resultado que se obtiene por pantalla. ¿Por qué no sucede lo mismo con la variable `dob`?

Ejercicio 12

El siguiente programa muestra aspectos del trabajo con variables de tipos numéricos (enumerados incluidos) y los problemas que surgen cuando se mezclan tipos en las operaciones. Depure el programa y conteste a las siguientes preguntas.

```
#include <stdio.h>

enum dia_semana
{
    Lunes, Martes, Miercoles, Jueves, Viernes, Sabado, Domingo
};

int main ()
{
    enum dia_semana Hoy = Lunes;

    int a = 100, b = 7, c, d;
    float f = 7.0, g;
```

```
printf ("Hoy es %d\n\n", Hoy);          /* (a) */
Hoy = (enum dia_semana)3;              /* (b) */
printf ("Ahora la variable Hoy es %d\n\n", Hoy);

c = b / a;                             /* (c) */
d = f / a;                             /* (d) */
g = b / a;                             /* (e) */

printf ("c = %d, d = %d, g = %f\n", c, d, g);

c = (f * a) % 13;                      /* (f) */

printf ("Ahora c = %d\n", c);

return 0;
}
```

1. ¿Por qué da error de compilación la línea (f)? Corríjala mediante una conversión explícita (cast) para que se pueda compilar el programa y dé el resultado correctamente. No modifique el tipo de las variables.
2. ¿Qué valor se imprime por pantalla en la línea (a)? ¿Por qué? ¿Qué valor toma la variable `Hoy` en la línea (b)? Recuerde que en un tipo enumerado la variable sólo puede tomar valores definidos en la propia enumeración.
3. Razone cuál es el resultado de las operaciones de las líneas (c), (d) y (e) y qué valores toman las variables asignadas. Modifique la línea (e) para que se pueda obtener el resultado más preciso posible en la división.
4. Cree un tipo mediante la palabra clave `typedef` basado en el tipo `enum dia_semana`, y modifique el programa para emplearlo en lugar de `enum dia_semana`. El tipo debe llamarse `tDiaSemana`.

Ejercicio 13

En el siguiente programa se muestra el uso de los operadores de asignación. Justifique los resultados obtenidos al operar.

```
#include <stdio.h>

int main ()
{
    int a = 5, b = 0, c = 10, d = 3, e;

    a *= d++;
    b = d--;
    c /= d++;
    e = b++ + ++d;

    printf (" a = %d\n", a);
    printf (" b = %d\n", b);
    printf (" c = %d\n", c);
    printf (" d = %d\n", d);
    printf (" e = %d\n", e);

    return 0;
}
```