Informática. Curso 2007/2008.
Práctica 2. Iniciación al C

Fecha inicio: 28/02/2008

Fecha entrega: 12/03/2008

Objetivos de la práctica:

- Aprender a compilar programas en C a partir de código fuente en varios archivos.
- Analizar y comprender funciones y programas sencillos ya implementados.
- Comprender tipos y expresiones de C.
- Implementar programas sencillos en C.

IMPORTANTE:

El fichero P2_0000.tgz contiene material necesario para el desarrollo de esta práctica. Descárgatelo y descomprímelo haciendo `tar xvfz P2_0000.tgz`. Esto generará una carpeta con el nombre P2_0000 y diversas subcarpetas con el esquema de trabajo aconsejado para la realización de la práctica. Es conveniente que cambies el nombre de la carpeta sustituyendo el 0000 por tu número de grupo.

EJERCICIO 1

En la carpeta `ej1` se incluyen los siguientes archivos:

`ej1/p1.c`	Programa que imprime tabla de N!
`ej1/p2.c`	Programa que imprime tabla de combinaciones C(N,M)
`ej1/aritmetica.c`	Funciones útiles para aritmética (código fuente)
`ej1/aritmetica.h`	Funciones útiles para aritmética (cabeceras)

Lee atentamente el contenido de los 4 archivos anteriores y analiza los algoritmos que implementan.

a. Compilación y linkado. Ejecuta los siguientes comandos para compilar y crear los programas p1 y p2. Después de ejecutar cada comando, ejecuta `ls -l` para verificar qué archivos se han creado y sus permisos.

```
# COMPILE
gcc -c -ansi -Wall aritmetica.c
gcc -c -ansi -Wall p1.c
gcc -c -ansi -Wall p2.c

# LINK
gcc p1.o aritmetica.o -o p1
gcc p2.o aritmetica.o -o p2
```

Para no tener que teclear todos los comandos anteriores cada vez que queremos volver a compilar, es útil introducirlos en un fichero de texto. Crea un archivo nuevo llamado `compila`, que contenga todos los comandos anteriores. Ahora, cuando queramos compilar simplemente llamaremos a nuestro fichero de compilación ejecutando `./compila`

Importante: antes de poder ejecutar el fichero `compila` es necesario asignarle permisos de ejecución. Para ello haremos uso del comando `chmod` (`chmod a+x compila`).

Nota: si incluyes la sentencia `set -e` al principio de compila, parará en caso de error.

En la memoria:

- Explica la diferencia entre `COMPILE` y `LINK`
- Explica las opciones `-c`, `-ansi` y `-Wall`
- Incluye el resultado de ejecutar `ls -l` en la carpeta ej1 y explica el contenido de la primera columna del listado

b. Ejecución. Ejecuta el programa `p1` con N=5, redirigiendo la salida al fichero `p1_5.txt`. Para redirigir la salida se utiliza el operador > en el momento de ejecutar el programa. Para el caso anterior haríamos `./p1 > p1_5.txt`. Cuando el programa nos pida el valor del número, introduciremos 5. El programa creará el fichero `p1_5.txt`, que contiene los resultados. Mira el contenido de este fichero para comprobar que el programa funciona bien.

Ejecuta el programa p2 con N=10 y M=2. Redirige la salida al fichero `p2_10_2.txt` y comprueba los resultados.

En la memoria:

- Incluye el contenido de los archivos con los resultados e indica si son correctos
- Explica la función fact (máximo 5 líneas)
- Explica la diferencia entre fact y factf (máximo 5 líneas)
- Explica las funciones comb1 y comb2 (máximo 5 líneas por función)

c. Ejecuta el programa `p1` con N=40 y redirige la salida al fichero `p1_40.txt`. Ejecuta el programa p2 con N=20 y M=5, y redirige la salida al fichero `p2_20_5.txt`

En la memoria:

- Analiza qué ocurre con los valores de fact. Explica las causas de los errores.
- Analiza qué ocurre con los valores de factf y factd.
- Analiza los resultados de comb1 y comb2 y explica las causas de los errores.

d. El programa `p3.c` tiene varios errores y warnings de compilación. Debes corregirlos todos hasta que compile y funcione correctamente.

En la memoria:

- Lista los errores y warnings que se han producido al compilar
- Explica para cada uno de ellos las causas y cómo lo has solucionado

EJERCICIO 2. Tipos básicos y operadores

Tipos de datos básicos en C

El lenguaje C dispone de los siguientes tipos de datos para representar números:

- char e int para números enteros en representación binaria
- float y double para números reales en representación de coma flotante

Los valores booleanos también se representan con números. El 0 se considera falso y cualquier otro número se considera verdadero.

Cada tipo de datos numérico puede representar un rango de valores que depende de su tamaño en memoria. Por ejemplo, una variable de tipo char ocupa 1 byte (8bits), y por tanto puede representar 256 valores distintos.

Modificadores de los tipos de datos numéricos

Se pueden aplicar modificadores a los tipos de datos anteriores para indicar si llevan signo (signed o unsigned) o para que usen más o menos memoria (short o long). Los modificadores aplicables a cada tipo son:

- [signed | unsigned] char
- [short | long] [signed | unsigned] int

- float
- [long] double

La cantidad exacta de bytes empleados para representar cada tipo de datos numérico varía dependiendo de la arquitectura del ordenador y del compilador. De todas formas ANSI establece las siguientes reglas:

- `sizeof(short int) <= sizeof(int)    <= sizeof(long int)`
- `sizeof(float)     <= sizeof(double) <= sizeof(long double)`

Nota: en C, la función sizeof() indica el número de bytes ocupado por un tipo de datos o una variable. Por ejemplo, `sizeof(int)` devuelve el número de bytes ocupado por una variable de tipo `int`.

Operaciones con enteros y decimales

Los operadores aritméticos y lógicos (`+,-,/,*,==,&&,||,...`) se pueden aplicar tanto a enteros como a reales. Los dos operandos deben ser del mismo tipo y el resultado será también del mismo tipo que los operandos.

Cast explícito
El tipo de una expresión se puede redefinir de forma explícita indicando el nuevo tipo entre paréntesis. Por ejemplo, la expresión `(int)5.5` es de tipo int y tiene el valor 5.

Cast implícito
Si los operandos no tienen el mismo tipo, el compilador llevará a cabo conversiones automáticas de tipos. Las reglas de conversión convierten los operandos al tipo más general (aquel con mayor precisión y/o rango de valores). Por ejemplo, si un operando es int y el otro double, el compilador los tratará a ambos como double.

División entera
La división entre dos números enteros es un número entero. Por ejemplo:

- `9/2` es de tipo `int` y vale 4
- `9/2.0` es de tipo `double` y vale 4.5
- `(float)9/2` es de tipo `float` y vale 4.5

a. Revisad en pareja las siguientes expresiones y apuntad el resultado que da al evaluarlas.

```
/* char   c = 1;
   int    i = 2;
   float  f = 3.5;
   double d = 4.5; */

  Constantes
          0x0F                                  _____________
          010                                   _____________
          'd'                                   _____________
          'd'+i+1                               _____________
          123456789012345678901234.5      (*)   _____________
          123456789012345678901234.5+1    (*)   _____________

  Cast explicito
          (int) 5.5                             _____________
          ((int) d) * 2                         _____________
          (int) d * 2                           _____________
          (int)(d * 2)                          _____________
          (int)c                                _____________
          3+(int)d                              _____________
          9+(float)i                      (*)   _____________

  Division entera
          1 / 2                                 _____________
          1 / 2.0                         (*)   _____________
          (float)(1 / 2)                  (*)   _____________
```

```
        d/i                             (*) ___________
        i/(double)c                     (*) ___________

   Operacion modulo
        18 % 7                              ___________

   Pre-post incremento
        i--                                 ___________
        --i                                 ___________
        4 * ++i                             ___________
        4 * i++                             ___________

   Operadores de comparacion
        c*c >= 1                            ___________
        c*c <= 1                            ___________
        d*3 != 6                            ___________
        (20==20)+(1!=0)                     ___________

   Operador sizeof
        sizeof(c)                           ___________
        sizeof(i)                           ___________
        sizeof(f)                           ___________
        sizeof(d)                           ___________

   Operador ternario
        i > c ? i : c                       ___________

   Operadores logicos
        i > c || i > f                      ___________
        i > c && i > f                      ___________
        1 && 2                              ___________
        1 || 2                              ___________
```

Implementad un programa (`ej2/p1.c`) que muestre por pantalla el listado anterior, con el valor calculado. Comparad vuestros resultados con los del programa.

El programa debe comenzar mostrando la siguiente cabecera:

```
Practica  : 2
Ejercicio : 2a
Grupo     : xxxx
Nombres   : Nombre1, Nombre 2.
```

Tened en cuenta que las expresiones marcadas con (*) con de tipo float o double, y el resto son de tipo int. Para mostrar sus valores con la función printf debes usar `%f` y `%d` respectivamente.

En la memoria:

- Listado generado por el programa

b. Escribe un programa (`ej2/p2.c`) que muestre, para cada tipo de datos, la cantidad de bytes/bits de memoria que usa y los valores máximo/mínimo que se pueden representar (ver listado incluido más abajo).

Notas: para calcular el tipo de los datos signed, ten en cuenta que la mitad de los valores serán negativos. Para los tipos que usan representación en coma flotante (float y double) no debes calcular el rango.

```
Practica : 2
Ejercicio : 2b
Grupo     : xxxx
Nombres   : Nombre1, Nombre 2.

Tipo                Bytes       bits     Rango
====                =====       ====     =====
signed   char       1           8        -128    +127
unsigned char       1           8        0       +255
```

```
short signed   int  ...         ..       ..      ..
short unsigned int  ...         ..       ..      ..
long  signed   int  ...         ..       ..      ..
long  unsigned int  ...         ..       ..      ..

float               ...         ..       --      --
double              ...         ..       --      --
long double         ...         ..       --      --
```

En la memoria:

- Explicación del programa implementado
- Listado generado por el programa

EJERCICIO 3.

A las 00:00 horas del día 1 de enero de 2008 se puso en marcha un reloj que marca los segundos transcurridos desde entonces. Debes escribir un programa para convertir los segundos que muestra la pantalla de este reloj en una fecha/hora en formato legible (por ejemplo 18/01/2008 14:30:00).

a. Antes de nada haz un diagrama de flujo para que te ayude durante la implementación.

b. Crea el programa `ej3/reloj.c`, que leerá del usuario un número entero (segundos transcurridos desde las 00:00 horas del día 1 de enero de 2008) e imprimirá la fecha y la hora en el formato `dd/mm/yyyy hh:mi:ss`. Nota: es suficiente con que el programa funcione para fechas en enero de 2008. Realiza las pruebas necesarias para verificar que tu programa funciona correctamente.. Puede usarse la siguiente plantilla como modelo:

```
/* ----------------------------------------------------------------
   ** Programa : reloj
   ** Convierte segundos en texto en el que se lea la fecha/hora.
   **
   ** Entrada:
   ** El usuario introduce un int que representa los segundos transcurridos desde
   ** 01/01/2008 00:00:00
   **
   ** Salida:
   ** dd/mm/yyyy hh:mi:ss   Si corresponde a una fecha de enero/2008.
   ** ERROR1                Si seg < 0
   ** ERROR2                Si la fecha representada es mayor o igual que
   1/febrero/2008
   **
   ** Ejemplos:
   **
   ** TIEMPO     RESULTADO
   ** ======     =========
   ** -1         ERROR1
   ** 0          01/01/2008 00:00:00
   ** 1          01/01/2008 00:00:01
   ** 60         01/01/2008 00:01:00
   ** 1000       01/01/2008 00:16:40
   ** 86400      02/01/2008 00:00:00
   ** 172801     03/01/2008 00:00:01
   ** 333040     04/01/2008 20:30:40
   ** 2678399    31/01/2008 23:59:59
   ** 2678400    ERROR2
   **
   ** ----------------------------------------------------------------
   */

   int main() {
      int n;          /* Segundos introducidos por el usuario */
```

```
  /* Leemos el numero de segundos del usuario: */
    printf("introduce numero de segundos:\n");
    scanf("%d",&n);

  /* Resto del codigo a programar: */

  return 0;
}
```

c. (opcional). Amplía el programa anterior para que dé formato a fechas de enero, febrero o marzo de 2008. Ejemplos:

```
/*
** TIEMPO     RESULTADO
** ======     =========
** 2678400    01/02/2008 00:00:00
** 3456000    10/02/2008 00:00:00
** 2678400    01/02/2008 00:00:00
** 2678401    01/02/2008 00:00:01
** 5184000    01/03/2008 00:00:00
** 5184001    01/03/2008 00:00:01
** 7862399    31/03/2008 23:59:59
** 7862400    ERROR2
** 7862401    ERROR2
*/
```

d. (opcional). Modifica el programa anterior para que añada el día de la semana. Ejemplos:

```
/*
   ** TIEMPO     RESULTADO
   ** ======     =========
   ** 2678400    01/02/2008 00:00:00 VIERNES
   */
```

En la memoria:

- Diagrama de flujo del apartado a
- Salida del programa con varios ejemplos

ENTREGA

La entrega se debe hacer en formato electrónico **antes del 12/03/2008 23:59:59** a través de la página web de entrega de prácticas de la EPS. Recuerda que el usuario es *practicas* y la contraseña *entrega*.

Se entregará un único archivo con nombre P2_xxxx.tgz (xxxx es el número de la pareja) que incluirá:

- Memoria en formato .pdf o .doc
- ./ej1/* Programas y archivos generados en el ejercicio 1
- ./ej2/* Programas y archivos generados en el ejercicio 2
- ./ej3/* Programas y archivos generados en el ejercicio 3
- No es necesario incluir en la entrega los ejecutables y archivos .o

Se tendrá en cuenta para la evaluación:

- Que los programas estén formateados y comentados apropiadamente
- Que los programas compilen con las opciones `-ansi` y `-Wall` sin errores ni warnings
- Que la memoria esté bien redactada y presentada