**Apellidos:** **Nombre:** **Matrícula:**

# Examen

## 105000016 - Programación para Sistemas

### Grado en Ingeniería Informática (2009)

Lenguajes y Sistemas Informáticos e Ingeniería de Software
Facultad de Informática
Universidad Politécnica de Madrid

Curso 2010/2011 - Enero 2011

## Normas

- El examen puntúa sobre **12 puntos**.
- La duración total del mismo es de **una hora y cuarto**.
- Se deberá tener el DNI o el carnet de la UPM en lugar visible.
- No olvidar rellenar **apellidos, nombre y número de matrícula** en cada hoja.
- La solución al examen se proporcionará antes de la revisión.
- Las calificaciones se darán a conocer el **25 de enero** a través del Moodle de la asignatura.
- La revisión del examen tendrá lugar el **27 de enero** a las 16:00 en la sala 2319.

## Cuestionario

(1 punto) 1. Escribe la sentencias Bash que escriba "`Legible`" (o "`No legible`") en la salida estándar si el fichero `/tmp/foo` es legible (o no).

**Solución:** Las siguientes son soluciones válidas:

```
if test -r /tmp/foo; then echo Legible; else echo No legible; fi
if [ -r /tmp/foo ]; then echo Legible; else echo No legible; fi
if [[ -r /tmp/foo ]]; then echo Legible; else echo No legible; fi
[ -r /tmp/foo ] && echo Legible || echo No legible
[[ -r /tmp/foo ]] && echo Legible || echo No legible
```

(1 punto) 2. Dados los siguientes comandos de Bash

```
A=`echo Hola`
echo $A
```

Se pide señalar la respuesta que contiene la única línea de la salida estándar de dichos comandos.

A. `A`

**B. `Hola`**

C. `echo Hola`

D. `` `echo Hola` ``

(1 punto) 3. Responder si las siguientes líneas (donde *error* indica un error de sintaxis relacionado con `[`)

```
a != b
bash: error
a != b
a = b
a = b
```

son la salida estándar y la salida de error de los siguientes comandos Bash:

```
a=1; b=1;
if  [ a == b ]; then echo "a␣=␣b" ; else echo "a␣!=␣b"; fi
if  [ a -eq b ]; then echo "a␣=␣b" ; else echo "a␣!=␣b"; fi
if  [ $a == $b ]; then echo "a␣=␣b" ; else echo "a␣!=␣b"; fi
if  [ $a -eq $b ]; then echo "a␣=␣b" ; else echo "a␣!=␣b"; fi
```

**Nota aclaratoria:** un error de ejecución en las sentencias de la *condición* del **`if`** conduce a la ejecución de la rama **`else`**.

**A. Verdadero.**

B. Falso.

**Solución:**

1. El comando

   ```
   if  [ a == b ]; then echo "a␣=␣b" ; else echo "a␣!=␣b"; fi
   ```

   produce una salida estándar `a != b` pues el operador == se refiere a los strings `a` y `b` que son evidentemente diferentes.

2. El comando

   ```
   if  [ a -eq b ]; then echo "a␣=␣b" ; else echo "a␣!=␣b"; fi
   ```

   da lugar a un mensaje de error de sintaxis en la salida de error, porque el operador `-eq` solo se emplea con valores enteros y los strings `a` y `b` no lo son. Además, al fallar el programa `[` (**`test`**), se toma la salida alternativa y se envía a la salida estándar el texto `a != b`.

3. El comando

   ```
   if  [ $a == $b ]; then echo "a␣=␣b" ; else echo "a␣!=␣b"; fi
   ```

   produce una salida estándar `a = b` pues tiene lugar una sustitución de las variables por su contenido, con lo que la comparación realizada es `[ 1 == 1 ]`.

4. Lo mismo sucede con el comando

   ```
   if  [ $a -eq $b ]; then echo "a␣=␣b" ; else echo "a␣!=␣b"; fi
   ```

   pues la comparación es `[ 1 -eq 1 ]`, cuya respuesta es verdadera.

En resumen, la respuesta correcta es **A. Verdadero**.

(1 punto) 4. Asumiendo que desde la entrada estándar se introducen los siguientes caracteres:

```
S
Uno Dos Tres
Uno Dos Tres
```

¿Cuáles son las tres líneas de la salida estándar de los siguientes comandos Bash?

```
read -p "Teclee␣su␣respuesta␣(S/N)>"; echo $REPLY
read A B; echo A=$A B=$B
read A B; echo A=$A B=$B $REPLY
```

**Solución:**

```
S
A=Uno B=Dos
A=Uno B=Dos Tres S
```

1. Si se teclea `S` como respuesta al comando

   ```
   read -p "Teclee␣su␣respuesta␣(S/N)>"; echo $REPLY
   ```

   En la salida estándar se muestra `S` pues, en caso de no introducir ninguna variable, la entrada se almacena en la variable `REPLY`.

2. Si se teclea `Uno Dos Tres` como respuesta al comando

   ```
   read A B; echo A=$A B=$B
   ```

   Se almacena en la variable `A` la tira de caracteres `Uno` (ya que el separador estándar es el espacio) y, en la variable `B`, el resto de la línea `Dos Tres`. Siendo así, se muestra en la slida estándar `A=Uno B=Dos Tres`.

3. Lo mismo sucede cuando se teclea `Uno Dos Tres` como respuesta al comando

   ```
   read A B; echo A=$A B=$B $REPLY
   ```

Lo primero es que hay que saber que no se almacena nada en la variable `REPLY` porque se han especificado dos variables de entrada.

En lo que se refiere al mensaje que se muestra en la salida estándar, cabe hacer dos interpretaciones del enunciado:

- Si las tres líneas de comandos corresponden a casos distintos que no guardan relación entre sí, la salida estándar será, como antes, `A=Uno B=Dos Tres` ya que `REPLY` estaría sin definir.
- Si las tres líneas de comandos se introducen en la misma sesión una tras otra, la salida estándar será, `A=Uno B=Dos Tres S` ya que el contenido `REPLY` sigue siendo `S`.

Ambas respuestas han sido consideradas válidas.

(1 punto) 5. ¿Qué es necesario especificar en el prototipo de una función en C?

> **Solución:** En el prototipo de una función en C se especifica cómo se llama, qué argumentos tiene, de qué tipo son, y qué tipo de resultado devuelve a su salida.

(1 punto) 6. ¿Cuánto dura en el tiempo una variable local o parámetro de una función en C?

> **Solución:** Una variable local dura lo que dure la llamada a esa función.

(1 punto) 7. En C, al comenzar la ejecución de una función, ¿cuál es el valor de una variable global?

> **Solución:** El valor inicial es 0 (nulo, todos sus bits a 0) siempre que no se inicialice en la propia declaración.

(1 punto) 8. ¿Existen en C variables de tipo booleano? Responder "sí" o "no". En caso afirmativo, indique cuál es la palabra reservada para ese tipo. En caso negativo, cómo se pueden simular.

> **Solución:** La respuesta es no. Vale cualquier variable de tipo int, float, double, char siendo verdadero equivalente a distinto de 0 (o no nulo) y falso equivalente a 0 (nulo, carácter nulo).

(1 punto) 9. Sea el siguiente fragmento de programa en C que incluye declaraciones de variables y asignaciones de memoria:

```
#include <stdlib.h>
int main (int argc, char *argv[])
{
  int **pint = (int **) malloc (1 * sizeof(pint));

  *pint = (int *) malloc (8 * sizeof(int));

  <<Operaciones sobre pint>>

  <<LIBERAR MEMORIA DINÁMICA>>

  return 0;
}
```

Indique el código a insertar en la parte indicada «LIBERAR MEMORIA DINÁMICA» necesario para liberar, antes de la finalización del programa, la memoria asignada dinámicamente:

**Solución:**

```
free (*pint);
free (pint);
```

(1 punto) 10. Se está realizando en C un programa `prog` que tiene `prog.c` como fichero fuente asociado. El ejecutable ha dado un error de ejecución y se quiere llamar al depurador `gdb` con un `core` para intentar localizar dónde en el código se produce el error.

Indique *todas* las acciones, especificando las llamadas concretas a compilador, sistema operativo, depurador, etc., que debe realizar para ello.

**Solución:**

- Llamada al compilador con flag -g:
  `gcc -g -Wall -ansi -pedantic prog.c -o prog`
- Llamada al comando bash ulimit para permitir la creación de ficheros core:
  `ulimit -c unlimited`
- Llamada al ejecutable prog para crear el fichero core (tras error de ejecución):
  `./prog`
- Llamada al depurador gdb con fichero core generado:
  `gdb prog core`

(1 punto) 11. Escriba el makefile que permita compilar una aplicación en C que consta de 2 archivos fuentes `readjpeg.c` y `dibuja.c` y un archivo cabecera `dibuja.h` donde están los prototipos de todas las funciones usadas por `dibuja.c`. La función `main` está incluida en `dibuja.c`. La aplicación usa una biblioteca del sistema denominada `libjpeg.a`. El nombre del ejecutable será `dibuja`.

**Solución:**

```
CCFLAGS=-g -Wall -ansi -pedantic
#
dibuja: dibuja.o readjpeg.o
        gcc -o dibuja dibuja.o readjpeg.o -ljpeg
dibuja.o: dibuja.c dibuja.h
        gcc $(CCFLAGS) -c dibuja.c
readjpeg.o: readjpeg.c
        gcc $(CCFLAGS) -c readjpeg.c
```

(1 punto) 12. Escriba la salida que produciría la ejecución del siguiente programa en C :

```
#include <stdio.h>
int main (int argc, char *argv[])
{
  int lista[]= {1,2,3,4,5};
  int *p;

  p=&lista[0];
  printf("Salida: %d\n", *(p+1) + *(p+4));
  return 0;
}
```

**Solución:**

```
Salida:7
```