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 2012/2013 - Enero 2013

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.
- La fecha prevista de publicación de calificaciones es el **22 de enero**, y se realizará a través del Aula Virtual de la asignatura.
- La revisión del examen tendrá lugar el **24 de enero** a las 12:00 en la sala 2319.

Cuestionario

(1 punto) 1. ¿Cuál sería la salida estándar del siguiente mandato Bash?

```
a=1; b=1
[[ $a == $b ]]  &&  echo "a␣=␣b"  ||  echo "a␣!=␣b"
```

A. `a != b`

B. `a = b`

(1 punto) 2. Suponiendo que las variables `A` y `B` contienen números enteros válidos. ¿Cuál de los siguientes mandatos comprueba si `$A` es mayor que `$B`?

A. `[ $A -gt $B ]`

B. `[ $A -ge $B ]`

C. `[ $A -lt $B ]`

D. `[ $A -le $B ]`

(1 punto) 3. ¿Cuál es el significado de la variable especial `$2` en una función o en un *script* Bash?

Solución: Segundo argumento con el que se invoca el *script* o función.

(1 punto) 4. En el manual de Bash, se puede leer la siguiente descripción sobre la expansión de variables:

```
${var:despl:long}

  Expansión de subcadena. Expande hasta long caracteres de var,
empezando en el caracter especificado por el índice despl.
Si long evalúa a un entero menor que 0 entonces se interpreta
como un índice desde el final en vez de como el número de
caracteres y expande desde depl hasta ese índice. La indexación
de la subcadena indicada por despl empieza por cero.
```

Escribir las tres líneas de la salida estándar resultado de la ejecución de los siguientes mandatos Bash:

```
unset X
X="123456789"
echo ${X:3:5}
echo ${X:3:1}
echo ${X:3:-5}
```

Solución:

```
45678
4
4
```

(1 punto) 5. Indica el operador que habría que utilizar para obtener el valor almacenado en una dirección de memoria guardada en una variable de tipo puntero.

Solución: `*`

(1 punto) 6. Dado el siguiente programa en C. ¿Cuántas veces se imprime la palabra `Hola`?

```
#include<stdio.h>
int main()
{
    int x;
    for(x=-1; x<=10; x++)
    {
        if(x < 5)
            continue;
        else
            break;
        printf("Hola");
    }
    return 0;
}
```

Solución: Ninguna

(1 punto) 7. El programa `examen` tiene el siguiente código fuente:

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

int main(int argc, char **argv)
{
    printf("%s\n", *++argv);
    return 0;
}
```

Si lo ejecutamos de la siguiente forma:

```
./examen uno dos tres
```

¿Qué imprimiría por pantalla?

A. uno

B. `dos`

C. `tres`

D. `./examen`

(1 punto) 8. ¿Qué valor imprime por pantalla el siguiente programa en C?

```
#include <stdio.h>

int main( )
{
   int a[5] = { 5, 4, 3, 2, 1 };


   printf("%d␣\n", *(a + 1) );

   return 0;
}
```

Solución: 4

(1 punto) 9. En el siguiente programa en C, indique qué código falta para que el programa libere toda la memoria dinámica asignada:

```
#include <stdlib.h>

#define NFILAS 2
#define NCOLUMNAS 4

int main(void) {
  int **ptr;
  int i;

  /* Solicitando memoria: */
  ptr = (int **) malloc(NFILAS * sizeof(int *));
  for (i = 0; i < NFILAS; i++) {
    *(ptr+i) = (int *) malloc(NCOLUMNAS * sizeof(int));
  }

  /* ... */

  /* Liberando la memoria: */
  for (i = 0; i < NFILAS; i++) {
    <<PARTE DE CÓDIGO QUE FALTA>>
  }
  free(ptr);
  return 0;
}
```

Solución: `free(*(ptr+i));`

(1 punto) 10. Escriba un **typedef** con **struct** con nombre `Punto3D` que sea adecuado para representar un punto en un espacio tridimensional, es decir un punto con 3 coordenadas cartesianas enteras.

Solución:

```
typedef struct Punto3D_struct
{
  int x, y, z;
} Punto3D;
```

(1 punto) 11. Escriba un makefile adecuado compilar una aplicación que consta de: (a) 2 archivos fuentes `procesar.c` y `escribir.c`, y (b) un archivo cabecera `procesar.h` donde están las declaraciones de las funciones usadas por `procesar.c` (nota: `escribir.c` no incluye `procesar.h`). La función `main` está incluida en `procesar.c`. La aplicación usa una biblioteca del sistema denominada `libjpeg.a`. El nombre del ejecutable será `procesar`. El fichero makefile no podrá indicar más dependencias entre ficheros de las que realmente existen.

Solución:

```
CCFLAGS=-Wall -ansi -pedantic
procesar: procesar.o escribir.o
  gcc -o procesar procesar.o escribir.o -ljpeg
procesar.o: procesar.c procesar.h
  gcc $(CCFLAGS) -c procesar.c
escribir.o: escribir.c
  gcc $(CCFLAGS) -c escribir.c
```

(1 punto) 12. Se está realizando 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 en el código dónde 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`