UNIVERSIDAD DE VALLADOLID **Departamento de Informática**
Informática Licenciatura en Matemáticas
Parcial Febrero de 2009

Apellidos : Nombre :

AVISOS:

- Es necesario entregar todas las hojas del enunciado del examen al finalizar, se hayan respondido o no a las preguntas.
- Es obligatorio firmar todas las hojas de respuesta del examen.
- En breve estará en la página web de la asignatura una copia del enunciado del examen y la solución del mismo.
- Debes usar el espacio debajo de cada cuestión para contestarla **de forma razonada**.
- **La duración de esta parte del examen es de 1 hora y 30 minutos. La siguiente parte del examen comenzará a las 11:45 h.**.

Cuestiones

1. **(0,5 puntos)** Da una definición de Informática de entre las que se han visto en la asignatura. Enumera además tres campos del saber que estén relacionados con el campo de la Informática.

   ```
   Definición: Conjunto de conocimientos y técnicas que hacen posible el
   tratamiento automático de información por medio de ordenadores.

   Campos relacionados: Lógica /Matemática, Electrónica o Física, por ejemplo.
   ```

2. **(0,5 puntos)** ¿Qué es una proposición en el contexto de la lógica proposicional? ¿Qué son las Fórmulas Bien Formadas y cómo se definen?

   ```
   Una proposición es la unidad básica de representación en la lógica proposicional.
   Es una sentencia declarativa afirmativa, no ambigua, y que tomará los valores de
    verdad cierto o falso.

   Una FBF es una expresión válida del lenguaje de la lógica proposicional y se
   define de forma recursiva:
   ```

   a) Un átomo es una FBF.
   b) Si α es una FBF, $\neg\alpha$ también es una FBF.
   c) Si α y β son FBFs, entonces $\alpha \cup \beta$, $\alpha \cap \beta$, $\alpha \supset \beta$, y $\alpha \equiv \beta$ también son FBFs.
   d) Ninguna otra combinación es una FBF.

3. **(0,75 puntos)** Dado un número entero $N = -1492$, indica cuál sería su representación interna en Signo y magnitud, Complemento a uno, Complemento a dos y Notación sesgada para $n = 16$ bits. Expresa el resultado en binario y en hexadecimal.

```
N= -1492
Su valor en binario con 16 bits es: 0000 0101 1101 0100 y en hexadecimal es 05D4

Notación en signo y magnitud: 1000 0101 1101 0100 y en hexadecimal 85D4.
Notación complemento a 1 es: 1111 1010 0010 1011 y en hexadecimal es FA2B.
Notación complemento a 2 es: 1111 1010 0010 1100 y en hexadecimal es FA2C.
Notación sesgada, con sesgo 32768: en binario 0111 1010 0010 1100 y en hexadecimal 7A2C
```

4. **(0,75 puntos)** ¿Cuál es la representación interna del número $N = -1234,4567$ en notación IEEE-754 en simple precisión?

```
N=-1234,4567 = -0,12344567x10**4 = -0,12344567x2**13,287712 = -1,220703x2**13

En simple precisión: ns=1, ne = 8, nm = 23.
La mantisa ya está normalizada y el signo es 1. Para empaquetarla debo representar:
0,220703 = 0,38800000 en hexadecimal
Sesgo = 128; e = 128 + E = 141 = 8D

N= 1100 0110 1001 1100 0100 1000 0000 0000
```

Cuestiones teórico-prácticas de programación

1. **(0,5 puntos)**¿Qué es el esquema condicional generalizado? ¿Cuándo podemos utilizarlo y cuándo debemos utilizar el esquema condicional tradicional?

```
Es uno de los esquemas que nos permiten hacernos preguntas sobre el contexto de
un programa. Nos permite agrupar varios condicionales anidados, si los condicionales
evalúan un expresión escalar (entera o carácter). Analiza caso por caso y si el
caso coincide con el valor de  la expresión, realiza la acción asociada.
Si la expresión no es escalar o no hay un número significativo de casos distintos,
se puede recurrir al esquema condicional tradicional.
```

2. **(0,5 puntos)** Enumera los distintos tipos de esquemas repetitivos que hay, sin entrar en detalles sobre su sintaxis. Indica cuándo se debe utilizar cada uno.

```
Hay habitualmente tres tipos de esquemas: mientras_hacer_fin_mientras,
repetir_hasta_que y para_desde_hasta_hacer_fin_para. Además, existe un esquema
general, iterar_salir_si_fin_iterar, que puede realizar todos los demás.

El esquema mientras_hacer_fin_mientras debe utilizarse cuando no se conozca a
priori el número de iteraciones y puede que no se entre alguna vez en el bucle.
El esquema repetir_hasta_que debe utilizarse cuando no se conozca a priori
el número de iteraciones y se ejecute al menos una vez el bucle.
El esquema para_desde_hasta_hacer_fin_para debe usarse si se conoce de antemano
el número de iteraciones.
El esquema general puede substituir a cualquiera de los esquemas anteriores.
```

3. **(0,5 puntos)** Enumera y explica los tres tipos de parámetros formales que existen en código algorítmico (según el uso que se hace de los valores de los parámetros). Indica cuál es su equivalente en C.

```
- Parámetros dato: aquellos cuyo valor no es modificado por el algoritmo.
- Parámetro resultado: aquel cuyo valor original no se utiliza y el algoritmo
cambia su valor final.
- Parámetro dato-resultado: aquel cuyo valor original se utiliza y además el
algoritmo cambia su valor.
El equivalente en C de parámetro dato es el paso por valor. El equivalente de
los parámetros dato-resultado son el paso por referencia. Los parámetros
resultado serían equivalentes a los valores que devuelve la función. Y si hay
más de uno, deberían implementarse el resto como paso por referencia.
```

4. **(0,75 puntos)** Realiza una función en C que nos permita comprobar si **un número NO es primo**. Debes indicar cuál es el principio básico de resolución que vas a emplear y qué secuencia de elementos vas a tratar.

   La función debe respetar las siguientes especificaciones en código algorítmico:

```
Algoritmo no_es_primo (N, resp) es
N: numérico; {parámetro dato; entero positivo a comprobar si no es primo}
ok: lógico; {parámetro resultado; cierto si el número no es primo;
falso en caso contrario}

int no_es_primo (int N)
{
  int resp;
  int I;
  resp = 0;
  I = 2;
  while (N % I != 0 && I <= N/2) I++;
  if (I <= N/2) resp = 1;
  return resp;
}
```

5. **(0,75 puntos)** Realiza una versión del algoritmo que permite calcular el cociente y el resto de la división entera entre dos números utilizando **sumas sucesivas**.

   El algoritmo debe respetar las siguientes especificaciones:

```
Algoritmo division_entera (dividendo, divisor, cociente, resto, ok) es
dividendo, divisor: numérico; {parámetros dato; enteros}
cociente, resto: numérico; {parámetros resultado; enteros;  contienen el
  cociente y el resto de la división entera de dividendo entre divisor}
ok: lógico; {parámetro resultado; indica cierto si se pudo hacer la operación;
  falso en caso contrario}
Inicio
  si divisor = 0 entonces
   ok := falso;
  sino
   ok := cierto;
   resto:= 0;
   cociente := 0;
   mientras resto <= dividendo hacer
    resto := resto + divisor;
    cociente := cociente + 1;
   fin mientras;
   {resto > dividendo}
   cociente :=cociente - 1;
   resto := divisor - (resto - dividendo);
  fin si;
Fin
```

   Explica cuál sería **la cabecera de la función en C** equivalente a este algoritmo, justificando la elección del tipo de la función.

```
Hay varios parámetro-resultado, entonces elijo uno como tipo de la función,
por ejemplo ok que en C será un entero, y el resto se pasan como parámetros por
referencia:

int division_entera (dividendo, divisor, cociente, resto)
int dividendo, divisor, *cociente, *resto;
```