# LENGUAJES DE PROGRAMACIÓN

## Solución al examen de Junio 2026, Primera Semana

**PREGUNTA 1** (3 puntos)

Indique la veracidad o falsedad de cada una de las afirmaciones siguientes, **explicando detalladamente** en cada caso el motivo de su respuesta.

**A.** (0.5 puntos) En la memoria de la máquina de Von Neumann únicamente se almacenan los datos del programa a ejecutar.

**B.** (0.5 puntos) Existen lenguajes donde no hay que declarar las variables, ya que se declaran implícitamente al aparecer como destino de sentencias de asignación.

**C.** (0.5 puntos) El resultado de las dos siguientes operaciones de asignación en C++

```
double x = 3.84;
int n = x;
```

produce que el valor de la variable `n` sea 4.

**D.** (0.5 puntos) Cuando el parámetro de una función de C++ es un array, éste se pasa a la función por valor.

**E.** (0.5 puntos) El algoritmo de la STL

```
remove_copy(p1, p2, pResutl, val)
```

copia la secuencia de origen en la secuencia destino sin eliminar elementos.

**F.** (0.5 puntos) En C++ no es posible definir un tipo de estructura en la cual uno de sus miembros sea una estructura de ese mismo tipo.

## Solución a la Pregunta 1

| | | |
|---|---|---|
| **A** | Falso | Véase la página 27 del texto base. |
| **B** | Verdadero | Véase la página 104 del texto base. |
| **C** | Falso | Véase la página 176 del texto base. |
| **D** | Falso | Véase la página 367 del texto base. |
| **E** | Falso | Véase la página 558 del texto base. |
| **F** | Verdadero | Véase la página 236 del texto base. |

**PREGUNTA 2** (1 punto)

Escriba la salida por consola producida al ejecutar el programa en C++ escrito a continuación.

```
#include <iostream>

int main() {
   int *p1, *p2, *p3, L;
   int a[] = { 2, 1 , 4, 3 };
   L = -10;
   p1 = a;
   p2 = &L;
   p1++;
   *p1 = *p2;
   p2 = (p1 + 1);
   std::cout << *p1 << " " << *p2 << " ";
   p2 = new int;
   *p2 = L+1;
   std::cout << a[0] << " " << a[1] << " " << a[2] << " ";
   p3 = p1;
   std::cout << *p1 << " " << *p2 << " " << *p3 << " ";
   return 0;
}
```

**Solución a la Pregunta 2**

La salida por consola producida al ejecutar el programa es:

```
-10 4 2 -10 4 -10 -9 -10
```

**PREGUNTA 3** (1 punto)

Escriba la salida por consola producida al ejecutar el programa en C++ escrito a continuación.

```
#include <iostream>
#include <vector>
#include <algorithm>

int main() {
   std::vector<double> v1, v2;
   for (double x = 5; x >= -1; x -= 1)
      v1.push_back(x);
   v1.push_back(10);
   std::vector<double>::iterator p1 =
       std::find_if(v1.begin(), v1.end(),
                    [](double x) -> bool
                      { return x < 4; });
   while (p1 != v1.end()) {
      std::cout << *p1 << " ";
      p1++;
   }
   std::cout << std::endl;
   for (double x = -1; x <= 5; x += 1)
      v2.push_back(x);
   std::vector<double>::iterator p2 =
        std::find_if(v2.begin(), v2.end(),
                     [](double x) -> bool
                       { return x > 1 && x < 3; });
   while (p2 != v2.end()) {
      std::cout << *p2 << " ";
      p2++;
   }
   std::cout << std::endl;
   return 0;
}
```

**Solución a la Pregunta 3**

La salida por consola producida al ejecutar el programa es:

```
3 2 1 0 -1 10
2 3 4 5
```

**PREGUNTA 4** (3 puntos)

Se desea estudiar el proceso de infección de un hongo que afecta a los robles en una zona boscosa. Para ello, se divide la superficie bidimensional del bosque en una cuadrícula rectangular plana. Cada celda de la cuadrícula se identifica mediante su número de fila y su número de columna. Dichos números toman valores enteros positivos, comenzando por uno. Designaremos $(i, j)$ la posición de la celda situada en la fila $i$ y columna $j$.

Empleamos la variable $t$ para designar el tiempo. La evolución en el tiempo se calcula comenzando en el instante $t_0$ y avanzando a pasos constantes en el tiempo. Los instantes de tiempo en los cuales se evalúa el autómata celular se denominan $t_0$, $t_1$, $t_2$, ..., $t_N$.

Sea $R_0(i, j)$ el número de robles dentro de la celda $(i, j)$. El número de robles que hay en cada celda está almacenado en un fichero de texto llamado *arboles.txt*. En la primera fila del fichero están escritos el número de filas y el número de columnas de la cuadrícula, separados ambos por uno o más espacios en blanco. Seguidamente están escritos, también separados por uno o más espacios en blanco, en tantas filas y columnas como se ha declarado en la primera fila del fichero, el número de robles que hay en cada celda. Por ejemplo, el contenido del fichero podría ser el siguiente:

```
6 7
12 5 18 7 9 14 3
6 20 4 11 8 2 15
9 13 1 17 6 10 5
14 7 19 3 12 8 16
2 11 6 15 4 9 13
10 8 5 20 7 1 18
```

donde se indica que la retícula está compuesta por 6 filas y 7 columnas, y que, por ejemplo, el número de robles en la celda $(1, 1)$ es 12 y en la celda $(6, 7)$ es 18.

Sea $I_n(i, j)$ el número de robles infectados que hay en el instante $t_n$, dentro de la celda $(i, j)$. En el instante inicial todos los robles del bosque están sanos: $I_0(i, j) = 0$.

En el instante $t_1$, uno de los robles de una de las celdas se infecta.

La propagación de la enfermedad se modela a partir de ese instante de la forma descrita a continuación. El número de robles infectados en la celda $(i, j)$, en el instante de tiempo $t_n$, con $n > 1$, se calcula sumando el número de robles infectados que hay en el instante anterior, $t_{n-1}$, al incremento de árboles infectados producido entre los instantes $t_{n-1}$ y $t_n$, que denominaremos $\Delta I_n(i, j)$. Al hacer el cálculo, se tiene en cuenta que el número de robles

infectados en una celda no puede ser mayor que el número total de robles que hay en ella.

$$I_n(i,j) = \text{mín}\left(I_{n-1}(i,j) + \Delta I_n(i,j),\ R_0(i,j)\right) \qquad \text{para } n = 2, \ldots, N$$

Para $n > 1$, el incremento de árboles infectados en una celda se calcula a partir del número de infectados que hay en el instante anterior tanto en esa misma celda, como en las *celdas vecinas*, que con aquellas que están situadas inmediatamente encima, debajo, a la derecha y a la izquierda. Es decir, para $n > 1$, el incremento de árboles infectados se calcula de la forma siguiente:

$$\begin{aligned}\Delta I_n(i,j) =& I_{n-1}(i,j) + \\ & \lceil 0.4 \cdot (I_{n-1}(i,j-1) + I_{n-1}(i,j+1) + I_{n-1}(i+1,j) + I_{n-1}(i-1,j)) \rceil \\ & \text{para } n = 2, \ldots, N\end{aligned}$$

donde $\lceil\ \rceil$ representa la función de redondeo al entero superior (función `ceil` en C++). Obsérvese que las celdas situadas en los bordes de la retícula tienen un número de celdas vecinas menor que las celdas interiores a la retícula. Emplee la expresión anterior también para las celdas de los bordes, asumiendo que el número de infectados es cero en aquellas posiciones más allá de los límites de la retícula.

Escriba el código en C++ indicado a continuación.

**4.1** (0.5 puntos). Una función en C++ con prototipo

```
std::vector < std::vector<int> > LeeFich(
   std::string nombreFich)
      throw (std::invalid_argument);
```

que realice la lectura del fichero de texto llamado *arboles.txt*, devolviendo su contenido en un vector bidimensional. Si se produce error al abrir el fichero para lectura, la función debe lanzar una excepción.

**4.2** (0.5 puntos). Una función en C++ con prototipo

```
bool getPrimerInfectado(unsigned &filaInfectado,
                        unsigned &colInfectado,
                        unsigned numFilas,
                        unsigned numCols);
```

que solicite al usuario que introduzca por consola las coordenadas de la celda donde se inicia la infección, lea dichas coordenadas de consola y compruebe su validez, es decir,

que las coordenadas de la celda corresponden a una posición válida en la retícula. Si las coordenadas son válidas, la función devuelve true, en caso contrario, false. Las coordenadas introducidas por el usuario deben asignarse a los parámetros pasados por referencia. Los dos parámetros pasados por valor son el número total de filas y de columnas de la cuadrícula, respectivamente.

**4.3** (0.5 puntos). Una función en C++ con prototipo

```
void logInfectados(std::vector < std::vector<int> > &I,
                   int n_tiempo);
```

que escriba en la consola el número de robles infectados en cada una de las celdas (primer parámetro), en un determinado instante de tiempo (segundo parámetro). El número de infectados debe escribirse en la consola formando una matriz, en la cual el valor $I_n(i, j)$ ocupe la fila $i$, columna $j$.

**4.4** (0.5 puntos). Una función en C++ con el prototipo

```
std::vector < std::vector<int> > fTranEstado(
   const std::vector < std::vector<int> > &I,
   const std::vector < std::vector<int> > &R0);
```

que calcule los robles infectados en el siguiente instante de tiempo, a partir de los infectados en el instante actual (primer parámetro) y el número total de robles (segundo parámetro).

**4.5** (0.5 puntos). Una función en C++ con el prototipo

```
bool todoInfectado(
   const std::vector < std::vector<int> > &I,
   const std::vector < std::vector<int> > &R0);
```

que devuelva true si todos los árboles del bosque se encuentran infectados y false en caso contrario.

**4.6** (0.5 puntos). Un programa principal en C++ que, invocando las funciones anteriormente definidas, realice las tareas enumeradas a continuación.

1. Declarar una constante global de tipo entero llamada `N` y asignarle el valor 100, la cual almacena el número máximo de pasos de avance en el tiempo a simular. Declarar también una variable global de tipo *string* que almacene el nombre del fichero.

2. Leer el fichero *arboles.txt*, almacenando su contenido en un vector bidimensional llamado `R0`. Si se produce error, mostrar un mensaje en la consola indicándolo y terminar.

3. Escribir mensajes en la consola solicitando al usuario que introduzca por consola las coordenadas de la celda en la cual comienza la infección en el instante $t_1$.

   El programa debe comprobar que los valores introducidos por el usuario son válidos. Si no son válidos, el programa debe mostrar un mensaje en la consola indicándolo y terminar.

4. En cada uno de los instantes de tiempo $t_0$, $t_1$, $t_2$, $\ldots$, escribir en la consola el número de robles infectados en cada una de las celdas de la retícula.

   Esta información debe escribirse formando una matriz, en la cual el valor $I_n(i, j)$ ocupe la fila $i$, columna $j$.

   El programa finaliza cuando se satisface al menos una de las dos condiciones siguientes:

   *a*) El tiempo ha avanzado $N$ pasos, es decir, $t = t_N$.

   *b*) Todos los árboles del bosque están infectados.

## Solución a la Pregunta 4

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <fstream>
#include <stdexcept>
#include <cmath>

const std::string fichero = "arboles.txt";
const int N = 100; // Num. maximo pasos a simular

std::vector < std::vector<int> > LeeFich(std::string nombreFich)
                                          throw (std::invalid_argument) {
   std::ifstream inFich(nombreFich, std::ios::in);
   if (!inFich)
      throw std::invalid_argument("Error al abrir el fichero");
   int numFilas, numCols;
   inFich >> numFilas >> numCols;
   std::vector < std::vector<int> > R0(numFilas, std::vector<int>(numCols));
   for (int i=0; i<numFilas; i++)
      for (int j=0; j<numCols; j++)
         inFich >> R0[i][j];
   inFich.close();
   return R0;
}

bool getPrimerInfectado(unsigned &filaInfectado, unsigned &colInfectado,
                        unsigned numFilas, unsigned numCols) {
   std::cout << "Fila celda primera infeccion: ";
   std::cin >> filaInfectado;
   std::cout << "Columna celda primera infeccion: ";
   std::cin >> colInfectado;
   return  filaInfectado >= 1         &&
           filaInfectado <= numFilas  &&
           colInfectado  >= 1         &&
           colInfectado  <=numCols;
}

void logInfectados(std::vector < std::vector<int> > &I, int n_tiempo) {
   std::cout << "t" << n_tiempo << std::endl;
   for (unsigned i=1; i<I.size()-1; i++) {
      for (unsigned j=1; j<I[i].size()-1; j++)
         std::cout << I[i][j] << " ";
      std::cout << std::endl;
   }
   return;
}

std::vector < std::vector<int> > fTranEstado(
                                  const std::vector < std::vector<int> > &I,
                                  const std::vector < std::vector<int> > &R0) {
   const unsigned numFilas = R0.size();
   const unsigned numCols  = R0[0].size();
   std::vector < std::vector<int> >
```

```
54          Inew(numFilas+2, std::vector<int>(numCols+2, 0));
55    for (unsigned i=1; i<numFilas+1; i++)
56       for (unsigned j=1; j<numCols+1; j++) {
57          unsigned deltaI = I[i][j] +
58                std::ceil(0.4*(I[i][j-1]+I[i][j+1]+I[i+1][j]+I[i-1][j]));
59          Inew[i][j] = I[i][j] + deltaI;
60          if ( Inew[i][j] > R0[i-1][j-1] )
61             Inew[i][j] = R0[i-1][j-1];
62       }
63    return Inew;
64 }
65
66 bool todoInfectado(const std::vector < std::vector<int> > &I,
67                    const std::vector < std::vector<int> > &R0) {
68    const unsigned numFilas = R0.size();
69    const unsigned numCols  = R0[0].size();
70    for (unsigned i=0; i<numFilas; i++)
71       for (unsigned j=0; j<numCols; j++)
72          if (I[i+1][j+1] != R0[i][j])
73             return false;
74    return true;
75 }
76
77 int main() {
78    // Lectura del fichero
79    std::vector < std::vector<int> > R0;
80    try {
81       R0 = LeeFich(fichero);
82    } catch (std::invalid_argument &exc) {
83       std::cerr << exc.what() << std::endl;
84       return 0;
85    }
86    const unsigned numFilas = R0.size();
87    const unsigned numCols  = R0[0].size();
88    // Primer arbol infectado
89    unsigned filaInfectado, colInfectado;
90    if ( !getPrimerInfectado(filaInfectado, colInfectado,
91                             numFilas, numCols) ) {
92       std::cerr << "Posicion no valida\n";
93       return 0;
94    }
95    // Inicializacion infectados
96    std::vector < std::vector<int> >
97       I(numFilas+2, std::vector<int>(numCols+2, 0));
98    logInfectados(I, 0);
99    // Primera infeccion
100   I[filaInfectado][colInfectado] = 1;
101   logInfectados(I, 1);
102   // Evolucion de la infeccion
103   for (int n=2; n<=N && !todoInfectado(I,R0); n++) {
104      I = fTranEstado(I, R0);
105      logInfectados(I, n);
106   }
107   return 0;
108 }
```

**PREGUNTA 5** (2 puntos)

Escriba un programa en C++ que realice las tareas enumeradas a continuación:

1. Declarar e inicializar una lista, llamada `L`, en el cual se almacenan las notas numéricas obtenidas por un grupo de alumnos. La lista debe inicializarse con los valores siguientes:

   ```
   L = (10, 9, 8.5, 8, 7.75, 7, 6, 5, 4, 1, 0)
   ```

2. Empleando el algoritmo `transform`, a partir de la lista `L` que contiene las notas numéricas, generar otra lista, llamada `calificaciones`, que contenga las calificaciones (MH, SB, NT, AP, SS) de los alumnos.

   En la tabla mostrada a continuación se indica la correspondencia entre la nota numérica y la calificación.

| **Nota numérica ($n$)** | **Calificación** |
|---|---|
| $9.8 \leq n \leq 10$ | MH |
| $9 \leq n < 9.8$ | SB |
| $7 \leq n < 9$ | NT |
| $5 \leq n < 7$ | AP |
| $0 \leq n < 5$ | SS |

3. Escribir en la consola el contenido de la lista `calificaciones`.

4. Escribir en la consola cada calificación y el número total de alumnos que la han obtenido. Debe emplear el algoritmo `count` para calcular el número de alumnos que ha obtenido cada calificación (MH, SB, NT, AP y SS).

5. Terminar.

## Solución a la Pregunta 5

```cpp
#include <iostream>
#include <list>
#include <vector>
#include <algorithm>
#include <string>

const std::vector<std::string> notaString { "SS", "AP", "NT", "SB", "MH" };

const std::vector<double> notaCorte { 5.0, 7.0, 9.0, 9.8 };

std::string calificacionesFun(double x) {
   for (unsigned int i = 0; i < notaCorte.size(); i++)
      if (x < notaCorte[i])
         return notaString[i];
   return notaString[notaString.size() - 1];
}

int main() {
   std::list<double> L = {10, 9, 8.5, 8, 7.75, 7, 6, 5, 4, 1, 0};

   std::list<std::string> calificaciones(L.size());
   std::list<std::string>::iterator pEndD = std::transform(
                                              L.begin(), L.end(),
                                              calificaciones.begin(),
                                              calificacionesFun );

   std::cout << "Calificaciones obtenidas por los alumnos:\n";
   for (std::list<std::string>::iterator p = calificaciones.begin();
        p != pEndD; p++)
      std::cout << (*p) << " ";
   std::cout << std::endl;

   for (unsigned int i = 0; i < notaString.size(); i++)
      std::cout << "Numero de " << notaString[i] << ": "
                << std::count(calificaciones.begin(),
                              calificaciones.end(),
                              notaString[i])
                << std::endl;
   return 0;
}
```