# LENGUAJES DE PROGRAMACIÓN

## Solución al examen de Septiembre 2022

**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) La máquina de Von Neumann se programa en lenguaje FORTRAN.

**B.** (0.5 puntos) La principal desventaja de la interpretación pura es el tiempo de ejecución del código fuente, que es mayor que si se compila.

**C.** (0.5 puntos) El lenguaje Pascal permite asignar un valor a la variable al declararla.

**D.** (0.5 puntos) Dentro de un bloque de código únicamente pueden usarse las variables locales al bloque.

**E.** (0.5 puntos) La excepción `std::range_error` de C++ puede ser lanzada por el operador `new` cuando fracasa el intento de reservar el espacio de almacenamiento.

**F.** (0.5 puntos) La variable `v` es declarada de la forma siguiente:

```
std::list<int> v
```

Entonces, la sentencia

```
v.empty()
```

elimina todos los elementos de la lista `v`.

## Solución a la Pregunta 1

| | | |
|---|---|---|
| **A** | Falso | Véase la página 32 del texto base. |
| **B** | Verdadero | Véase la página 62 del texto base. |
| **C** | Falso | Véase la página 104 del texto base. |
| **D** | Falso | Véanse las páginas 122 y 123 del texto base. |
| **E** | Falso | Véase la página 314 del texto base. |
| **F** | Falso | Véase la página 490 del texto base. |

**PREGUNTA 2** (1 punto)

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

```
#include <iostream>

void fun1(int* a, int* b) {
   if (*a <= *b) {
      int temp = *a;
      *a = *b;
      *b = temp;
   }
}

void fun2(int A[]) {
   std::cout << "fun2 " << A[0] << " " << A[1] << std::endl;
   A[1] = -4;
}

int main() {
   int a[] = { 10, 20, 30 };
   int* n1, *n2;
   int ii = 1, jj = 4;
   n1 = &ii;
   n2 = &jj;
   fun1(n1, n2);
   std::cout << ii << " " << jj << std::endl;
   fun1(&jj, &ii);
   std::cout << ii << " " << jj << std::endl;
   fun2(a);
   std::cout <<  a[0] << " " << a[1];
   std::cout << " " << a[2] << std::endl;
   fun2(&a[1]);
   std::cout <<  a[0] << " " << a[1];
   std::cout << "  " << a[2] << std::endl;
   return 0;
}
```

**Solución a la Pregunta 2**

La salida por consola producida al ejecutar el programa es:

```
4 1
1 4
fun2 10 20
10 -4 30
fun2 -4 30
10 -4  -4
```

**PREGUNTA 3** (1 punto)

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

```
#include <iostream>

int main() {
    struct nodo {
        int num1;
        struct nodo* n;
    };
    int a[] = { -3, 1, -4, 7, 80 };
    int* p = a;
    struct nodo* nodo1;
    struct nodo* no1 = new nodo;
    struct nodo* no2 = new nodo;
    struct nodo* no3 = new nodo;
    struct nodo* no4 = new nodo;
    no1->num1 = *(p + 1);
    no2->num1 = *(p + 2);
    no3->num1 = *p;
    no4->num1 = a[4];
    no3->n = no1;
    no1->n = no2;
    no2->n = no4;
    no4->n = NULL;
    nodo1 = no3;
    for (int i = 1; i < 5; i++) {
        std::cout << nodo1->num1 << " ";
        nodo1 = nodo1->n;
    }
    return 0;
}
```

**Solución a la Pregunta 3**

La salida por consola producida al ejecutar el programa es:

```
-3 1 -4 80
```

**PREGUNTA 4** (2 puntos)

Se desea calcular el coste de recoger, en un solo trayecto, en determinado orden, $N$ productos de un almacén. La masa de cada producto es conocida. También lo es la ubicación de cada producto, que está especificada mediante sus coordenadas cartesianas $(x, y)$.

El coste $C$ del trayecto entre dos ubicaciones se calcula de la forma siguiente:

$$C = d \cdot \frac{M}{100}$$

donde $d$ es la distancia en línea recta entre las dos ubicaciones y $M$ es la masa transportada en el trayecto, que es igual a la suma de las masas de todos los productos recogidos anteriormente. Cada vez que se recoge un producto, la masa transportada se incrementa en un valor igual a la masa del producto recogido.

El origen de coordenadas $(0, 0)$ es el punto inicial y final del trayecto completo de recogida de los $N$ productos. El coste del trayecto completo de recogida de los $N$ productos se calcula sumando el coste de los trayectos entre ubicaciones de recogida consecutivas, más el coste del trayecto en línea recta desde la última ubicación de recogida al origen de coordenadas. El coste del trayecto entre el origen de coordenadas y la ubicación del primer producto a recoger es nulo, ya que en dicho trayecto no se transporta ningún producto.

Escriba un programa en C++ que realice las acciones siguientes.

1. Declarar una constante global de tipo entero llamada $N$ y asignarle el valor $10$.

2. Declarar un tipo estructura llamado `Producto` que tenga tres miembros de tipo **double**: las coordenadas X e Y de la posición en que se encuentra el producto, y su masa.

3. Mediante un mensaje escrito en la consola, solicitar al usuario que introduzca por consola la ubicación y masa de cada uno de los $N$ productos a recoger. Leer los datos introducidos por el usuario, almacenándolos en un array de $N$ componentes de tipo `Producto`.

4. Calcular y escribir en la consola, en formato fijo con dos dígitos decimales, el coste de recoger los $N$ productos, en un único trayecto con inicio y finalización en el origen de coordenadas, en el orden en que han sido introducidos por el usuario.

5. Calcular y escribir en la consola, en formato fijo con dos dígitos decimales, la masa total de los $N$ productos.

6. Terminar.

El programa debe funcionar para cualquier $N > 0$.

### Solución a la Pregunta 4

```
#include <iostream>
#include <iomanip>
#include <cmath>

const int N = 10;

struct Producto {
  double x, y, masa;
};

int main() {
  std::cout << "Introduzca la ubicacion y masa de "
      << N << " productos:" << std::endl;
  Producto aP[N];
  for (int i = 0; i < N; i++)
    std::cin >> aP[i].x >> aP[i].y >> aP[i].masa;
  double C = 0, M = aP[0].masa;
  for (int i = 1; i < N; i++) {
    double d = std::sqrt(
        std::pow(aP[i].x - aP[i - 1].x, 2)
            + std::pow(aP[i].y - aP[i - 1].y, 2));
    C += d * M;
    M += aP[i].masa;
  }
  double d = std::sqrt(std::pow(aP[N - 1].x, 2)
            + std::pow(aP[N - 1].y, 2));
  C += d * M;
  std::cout << std::fixed << std::setprecision(2)
      << "Coste: "    << C / 100
      << ", Masa total: " << M  << std::endl;
  return 0;
}
```

**PREGUNTA 5** (3 puntos)

Consideremos un sistema de gestión del reparto de paquetes a domicilio. En un fichero de texto llamado *ordenPend.txt* están almacenadas las órdenes de reparto solicitadas ese mismo día y que aún están pendientes de ejecutar. Cada orden de reparto es descrita mediante una línea del fichero.

- En la primera columna del fichero está escrito el identificador de la orden de entrega. Se trata de una palabra compuesta de letras mayúsculas y/o números, que designa de manera unívoca la orden de entrega.
- En la segunda columna está escrito el instante de tiempo en la cual el cliente ha solicitado dicha entrega. Dicho instante está escrito en el formato HH:MM, donde la hora (HH) varía entre $00$ y $23$, y el minuto (MM) varía entre $00$ y $59$.

A modo de ejemplo, se muestra a continuación lo que podría ser el contenido del fichero.

```
W23A6   11:35
A31D    09:30
HYTT5   13:00
KJYR9   12:16
K8N     13:00
A34D    09:30
```

Obsérvese que no puede asumirse que las órdenes estén escritas en el fichero siguiendo ordenación alguna (ni respecto al identificador, ni al tiempo). Asimismo, obsérvese que puede haber más de una solicitud con el mismo instante de tiempo, es decir, con igual valor en la segunda columna.

En el fichero ejemplo anterior, las órdenes más tempranas son las del instante de tiempo `09:30`, las cuales tienen los identificadores `A31D` y `A34D`.

Mientras no exista ninguna orden de entrega pendiente, el contenido del fichero es el siguiente:

```
NULL     00:00
```

**5.1** (2 puntos) Escriba una función en C++ con el prototipo siguiente:

```
std::vector<Peticion> masAntiguas( )
                          throw (std::invalid_argument);
```

La función devuelve un vector de elementos del tipo estructura `Peticion`, la cual tiene dos campos de tipo `std::string` llamados `id` y `time`.

La función debe abrir el fichero *ordenPend.txt* para lectura. Pueden darse dos situaciones: que el fichero contenga al menos una orden pendiente de atender o que no contenga ninguna (en este caso, el fichero tendrá una única línea y su identificador será `NULL`).

- Si el fichero contiene al menos una orden pendiente de atender, la función debe realizar consecutivamente las tareas siguientes:
  1. Almacenar el contenido completo del fichero en una lista llamada `lPend`, cuyos elementos son del tipo estructura `Peticion`.
  2. Buscar en la lista `lPend` el elemento o elementos con el instante de tiempo más temprano. Almacenar la información de esos elementos en un vector del tipo estructura `Peticion` llamado `vResult`. Eliminar de la lista `lPend` dichos elementos.
  3. Reemplazar todo el contenido del fichero *ordenPend.txt* por el texto indicado a continuación.
     - Si la lista `lPend` no está vacía, escribir en el fichero el contenido de la lista respetando el formato del fichero: primera columna identificadores, segunda columna instantes de tiempo.
     - Si la lista está vacía, escribir en el fichero la línea de texto siguiente:

       ```
       NULL     00:00
       ```
  4. Devolver el vector `vResult`.
- Si en el fichero no hay órdenes pendientes de atender (en este caso, el fichero tendrá una única línea y su identificador será `NULL`), la función devuelve un vector vacío.

Si se produce error al abrir el fichero de texto para lectura o escritura, la función debe lanzar una excepción.

**5.2** (1 punto) Escriba un programa en C++ que realice las acciones siguientes.

1. Escribir en la consola el mensaje siguiente:

   ```
   Desea extraer ordenes de entrega (S/N)?
   ```

2. Leer la respuesta que el usuario introduzca por consola, almacenándola en una variable de tipo **char** llamada `res`.
3. Si el valor de la variable `res` es igual al carácter `S`, invocar la función `masAntiguas` a fin de obtener un vector con las órdenes más tempranas y eliminarlas del fichero.

   Si el vector está vacío, escribir un mensaje en la consola indicando que no hay órdenes de entrega pendientes. En caso contrario, escribir en la consola los identificadores y hora de las órdenes almacenadas en el vector devuelto por la función.

   Si la función lanza una excepción, indicarlo mediante un mensaje en consola.
4. Terminar.

### Solución a la Pregunta 5

```
#include <iostream>
#include <vector>
#include <stdexcept>
#include <string>
#include <fstream>
#include <list>

const std::string nombreFich = "ordenPend.txt";

struct Peticion {
  std::string id, time;
};

std::vector<Peticion> masAntiguas() throw (std::invalid_argument) {
  std::ifstream inFich(nombreFich.c_str(), std::ios::in);
  if (!inFich)
    throw std::invalid_argument("Error al abrir el fichero");
  std::vector<Peticion> vResult;
  Peticion pet;
  inFich >> pet.id >> pet.time;
  if (pet.id == "NULL")
    return vResult;
  std::list<Peticion> lPend;
  lPend.push_back(pet);
  while (inFich >> pet.id >> pet.time) {
    lPend.push_back(pet);
  }
  inFich.close();
  std::list<Peticion>::iterator p = lPend.begin();
  vResult.push_back(*p);
  p++;
  while (p != lPend.end()) {
    if (vResult[0].time > p->time) {
      vResult.clear();
      vResult.push_back(*p);
    } else if (vResult[0].time == p->time) {
      vResult.push_back(*p);
    }
    p++;
  }
  p = lPend.begin();
  while (p != lPend.end()) {
    if (p->time == vResult[0].time)
      p = lPend.erase(p);
    else
      p++;
  }
  std::ofstream outFich(nombreFich.c_str(),
      std::ios::out | std::ios::trunc);
  if (!outFich)
    throw std::invalid_argument("Error al abrir el fichero");
```

```
52    if (lPend.empty()) {
53      outFich << "NULL  00:00" << std::endl;
54    } else {
55      p = lPend.begin();
56      while (p != lPend.end()) {
57        outFich << p->id << "  " << p->time << std::endl;
58        p++;
59      }
60    }
61    outFich.close();
62    return vResult;
63  }
64
65  int main() {
66    std::cout << "Desea extraer ordenes de entrega (S/N)?\n";
67    char res;
68    std::cin >> res;
69    if (res == 'S') {
70      try {
71        std::vector<Peticion> vPets = masAntiguas();
72        if (!vPets.size()) {
73          std::cout << "No hay ordenes pendientes\n";
74        } else {
75          for (unsigned i = 0; i < vPets.size(); i++) {
76            std::cout << vPets[i].id << "  "
77                << vPets[i].time << std::endl;
78          }
79        }
80      } catch (std::invalid_argument &exc) {
81        std::cout << exc.what() << std::endl;
82      }
83    }
84    return 0;
85  }
```