LENGUAJES DE PROGRAMACIÓN

Solución al examen de Septiembre 2025

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 el lenguaje Pascal no es preciso declarar las variables antes de usarlas, ya que se declaran implícitamente al aparecer como destino de sentencias de asignación.

B. (0.5 puntos) En la sentencia de lenguaje C

```
char[] cadena = "Hola";
```

el elemento `cadena[1]` almacena el caracter `'H'`.

C. (0.5 puntos) Las funciones en C++ pueden ser apuntadas por punteros.

D. (0.5 puntos) La función `std::cin.peek()` devuelve el primer carácter del flujo de entrada, pero sin extraerlo del flujo.

E. (0.5 puntos) En el método de la burbuja, el número de intercambios es independiente de la secuencia de entrada.

F. (0.5 puntos) Mediante una expresión lambda, es posible definir en C++11 una función anónima que será creada en tiempo de ejecución y que puede ser pasada como argumento al algoritmo `count_if`.

Solución a la Pregunta 1

A	Falso	Véase la página 46 del texto base.
B	Falso	Véase la página 116 del texto base.
C	Verdadero	Véase la página 405 del texto base.
D	Verdadero	Véase la página 323 del texto base.
E	Falso	Véase la página 538 del texto base.
F	Verdadero	Véase la página 568 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, i;
   int a[] = { 1, 2, 3, 4 };
   i = -1;
   p1 = a;
   p1++;
   p2 = &i;
   *(p1++) = *p2;
   p2 = (p1 + 1);
   std::cout << *p1 << " " << *p2 << " " << std::endl;
   p2 = new int;
   p2 = --p1;
   std::cout << a[0] << " " << a[1] << " "
             << a[2] << " " << std::endl;
   p3 = ++p1;
   std::cout << *p1 << " " << *p2 << " "
             << *p3 << " ";
   return 0;
}
```

Solución a la Pregunta 2

La salida por consola producida al ejecutar el programa es:

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

PREGUNTA 3 (1 punto)

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

```
#include <iostream>
#include <list>

int main() {
   std::list<int> lista1(6, -2);
   std::list<int>::iterator p = lista1.begin();
   *p = 20;
   p++;
   *p = 30;
   p = lista1.end();
   p--;
   *p = 90;
   p = lista1.begin();
   while (p != lista1.end()) {
      std::cout << *p << " ";
      p++;
   }
   std::cout << std::endl;
   return 0;
}
```

Solución a la Pregunta 3

La salida por consola producida al ejecutar el programa es:

```
20 30 -2 -2 -2 90
```

PREGUNTA 4 (2.5 puntos)

Se desea estimar la vida total de una estrella, es decir, la duración de la fase en la que la estrella tiene actividad de fusión de hidrógeno en el núcleo. Dicha vida total (v) se calcula como el producto de cinco parámetros característicos (V_0, m, b, r, L) de la estrella:

$$v = V_0 \cdot m \cdot b \cdot r \cdot L$$

El valor del parámetro V_0 depende del *tipo espectral* de la estrella, el cual se define mediante una letra del conjunto {O, B, A, F, G, K, M}. La relación es la siguiente:

Tipo espectral	V_0 (unidades: millones de años)
O	5
B	50
A	500
F	2000
G	10000
K	30000
M	100000

El parámetro m se calcula a partir de la *metalicidad* de la estrella, es decir, de la proporción del contenido en la estrella de materiales más pesados que el Helio.

$$m = \begin{cases} 0.9 & \text{si metalicidad } < 0.005 \\ 1.0 & \text{si } 0.005 \leq \text{ metalicidad } \leq 0.02 \\ 1.1 & \text{si metalicidad } > 0.02 \end{cases}$$

El valor del parámetro b depende de si la estrella es *binaria*. Si la estrella es binaria, $b = 0.95$. Si no lo es, $b = 1$.

El valor del parámetro r depende de la *velocidad de rotación* de la superficie en el ecuador de la estrella:

$$r = \begin{cases} 1.05 & \text{si velocidad } < 10 \text{ km/s} \\ 1.00 & \text{si } 10 \text{ km/s} \leq \text{ velocidad } \leq 200 \text{ km/s} \\ 0.85 & \text{si velocidad } > 200 \text{ km/s} \end{cases}$$

El valor del parámetro L depende de la *luminosidad* de la estrella:

$$L = \begin{cases} 0.8 & \text{si luminosidad } < 0 \\ 1.0 & \text{si } 0 \leq \text{ luminosidad } \leq 8 \\ 1.2 & \text{si luminosidad } > 8 \end{cases}$$

Un fichero de texto llamado *estrellas.txt* contiene los datos de un conjunto de estrellas. En cada fila se encuentran los datos de una estrella, separados por espacios en blanco:

- En la primera columna se indica el *tipo espectral*, mediante una letra del conjunto {O, B, A, F, G, K, M}.
- En la segunda columna se indica la *metalicidad* (número real).
- En la tercera columna se indica la *velocidad de rotación* (número real).
- En la cuarta columna se indica la *luminosidad* (número real).
- En la quinta columna se indica si la estrella es binaria, mediante un número del conjunto {0, 1}. El número 1 indica que es binaria y el 0 que no lo es.

A modo de ejemplo, se muestran las primeras filas del fichero *estrellas.txt*:

```
G 0.015   2.0  4.8 0
B 0.025 310.0 -1.2 1
M 0.005   7.0 10.5 0
A 0.010 150.0  1.0 0
F 0.030  20.0  2.5 1
K 0.002   5.0  8.2 0
O 0.020 400.0 -5.0 1
```

La vida total estimada de la estrella descrita en la primera línea del fichero es:

$$v = 10000 \cdot 1 \cdot 1.05 \cdot 1 \cdot 1 = 10500 \text{ millones de años}$$

mientras que la vida total estimada de la estrella descrita en la segunda línea es:

$$v = 50 \cdot 1.1 \cdot 0.85 \cdot 0.8 \cdot 0.95 = 35.53 \text{ millones de años}$$

Escriba un programa en C++ que lea el fichero de texto *estrellas.txt* y escriba en la consola la vida total estimada de cada una de las estrellas descritas en el

fichero, en el mismo orden en que aparecen en el fichero. La vida estimada debe expresarse en millones de años, en formato fijo con dos dígitos decimales.

Al escribir el programa, puede suponer que el fichero *estrellas.txt* no contiene errores de formato.

Solución a la Pregunta 4

```cpp
#include <iostream>
#include <fstream>
#include <map>
#include <iomanip>

const std::map<char, double> mV0 = {
      { 'O', 5 },
      { 'B', 50 },
      { 'A', 500 },
      { 'F', 2000 },
      { 'G', 10000 },
      { 'K', 30000 },
      { 'M', 100000 }
};

int main() {
   std::ifstream inFich("estrellas.txt", std::ios::in);
   if (!inFich) {
      std::cerr << "Error al abrir el fichero\n";
      return 0;
   }
   char tipo;
   double met, vel, lum;
   bool bin;
   while (inFich >> tipo >> met >> vel >> lum >> bin) {
      double V0 = mV0.find(tipo)->second;
      double m = met<0.005 ? 0.9 : (met>0.02 ? 1.1 : 1);
      double b = bin ? 0.95 : 1;
      double r = vel<10 ? 1.05 : (vel>200 ? 0.85 : 1);
      double L = lum<0 ? 0.8 : (lum>8 ? 1.2 : 1);
      std::cout << std::fixed << std::setprecision(2)
                << V0 * m * b * r * L << std::endl;
   }
   inFich.close();
   return 0;
}
```

PREGUNTA 5 (2.5 puntos)

A fin de guardar la información relativa a las compras realizadas por los clientes, el sistema informático de un comercio hace uso de las dos estructuras de datos siguientes.

```
struct Items {
   std::string IDproducto;
   unsigned int unidades;
};

struct Compra {
   unsigned int IDcliente;
   std::vector<Items> vItems;
   std::string fecha; // Formato DD:MM:AAAA
};
```

Cada tipo de producto a la venta está identificado unívocamente mediante un identificador (`IDproducto`).

Cada cliente tiene asignado un número (`IDcliente`) que lo identifica de manera unívoca.

La estructura `Compra` registra el paso de un cliente por la caja del comercio: identifica al comprador (`IDcliente`), el número de unidades compradas de cada tipo de producto (`vItems`) y la fecha de la compra (`fecha`).

La fecha de compra es un *string* con el formato DD:MM:AAAA. Por ejemplo, la fecha `12:01:2022` indica el día 12 del mes de enero del año 2022.

La estructura `Items` almacena el número de artículos (`unidades`) del tipo `IDproducto` comprados por el cliente.

Escriba el código C++ descrito a continuación.

5.1 (0.75 puntos) Escriba una función con el prototipo siguiente:

```
int getNumItemsUnProductoPorCliente(
                std::list<Compra> &lCompras,
                std::string IDproducto,
                unsigned int IDcliente );
```

que tenga la funcionalidad descrita a continuación.

El primer argumento de la función referencia una lista con todas las compras realizadas en el comercio por los clientes. La función debe devolver el número total de unidades del producto cuyo `IDproducto` coincide con el segundo argumento de la función, que ha comprado el cliente cuyo IDcliente coincide con el tercer argumento de la función.

5.2 (0.75 puntos) Escriba una función con el prototipo siguiente:

```
std::map<int,int> getClientesPorItem(
                        std::list<Compra> &lCompras,
                        std::string IDproducto );
```

que tenga la funcionalidad descrita a continuación.

El primer argumento de la función referencia una lista con todas las compras realizadas en el comercio por los clientes. La función debe devolver un mapa que contenga como clave el identificador (`IDcliente`) de cada uno de los clientes que ha comprado el producto indicado mediante el segundo argumento de la función, y como valor la suma de las unidades de ese producto que ha comprado el cliente en todos sus pasos por caja.

5.3 (1 punto) Escriba una función con el prototipo siguiente:

```
int EliminaComprasAntiguas( std::list<Compra> &lCompras,
                            std::string fecha );
```

que tenga la funcionalidad descrita a continuación.

El primer argumento de la función referencia una lista con todas las compras realizadas en el comercio por los clientes. La función debe eliminar de la lista pasada como primer argumento todas las compras anteriores a la fecha pasada como segundo argumento de la función. La función debe devolver el número total de compras que ha eliminado de la lista.

Solución a la Pregunta 5

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

struct Items {
   std::string IDproducto;
   unsigned int unidades;
};

struct Compra {
   unsigned int IDcliente;
   std::vector<Items> vItems;
   std::string fecha;   // Formato DD:MM:AAAA
};

int getNumItemsUnProductoPorCliente(
                 std::list<Compra> &lCompras,
                 std::string IDproducto,
                 unsigned int IDcliente ) {
   int numProductos = 0;
   std::list<Compra>::iterator p = lCompras.begin();
   while ( p != lCompras.end() ) {
      if ( p->IDcliente == IDcliente )
         for (unsigned i=0; i< p->vItems.size(); i++)
            if (p->vItems[i].IDproducto == IDproducto)
               numProductos += p->vItems[i].unidades;
      p++;
   }
   return numProductos;
}

std:: map<int,int> getClientesPorItem(
                       std::list<Compra> &lCompras,
                       std::string IDproducto ) {
   std:: map<int,int> mRes;
   std::list<Compra>::iterator p = lCompras.begin();
   while ( p != lCompras.end() ) {
      for (unsigned i=0; i< p->vItems.size(); i++)
         if ( p->vItems[i].IDproducto == IDproducto ) {
            std:: map<int,int>::iterator mp = mRes.find(p->IDcliente);
            if ( mp == mRes.end() )
               mRes[p->IDcliente] = p->vItems[i].unidades;
            else
               mRes[p->IDcliente] += p->vItems[i].unidades;
         }
      p++;
   }
   return mRes;
}
```

```
52
53 int getDia(std::string fecha) {
54    return 10*(fecha[0]-'0') + fecha[1]-'0';
55 }
56
57 int getMes(std::string fecha) {
58    return 10*(fecha[3]-'0') + fecha[4]-'0';
59 }
60
61 int getAgno(std::string fecha) {
62    return 1000*(fecha[6]-'0') + 100*(fecha[7]-'0') +
63              10*(fecha[8]-'0') +      fecha[9]-'0';
64 }
65
66 int EliminaComprasAntiguas( std::list<Compra> &lCompras,
67                             std::string fecha ) {
68    int numElim = 0;
69    std::list<Compra>::iterator p = lCompras.begin();
70    while ( p != lCompras.end() ) {
71       bool borrar = getAgno(fecha) > getAgno(p->fecha)       ||
72                     ( getAgno(fecha) == getAgno(p->fecha) &&
73                       getMes(fecha) > getMes(p->fecha) )     ||
74                     ( getAgno(fecha) == getAgno(p->fecha) &&
75                       getMes(fecha)  == getMes(p->fecha)  &&
76                       getDia(fecha) > getDia(p->fecha) ) ;
77       if ( borrar ) {
78          p = lCompras.erase(p);
79          numElim++;
80       } else
81          p++;
82    }
83    return numElim;
84 }
```