LENGUAJES DE PROGRAMACIÓN

Solución al examen de Junio 2024, 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) FORTRAN IV fue el primer lenguaje de programación que introdujo los punteros.

B. (0.5 puntos) En la máquina de Von Neumann la memoria solo puede contener instrucciones.

C. (0.5 puntos) Dado un vector `v`, declarado en C++ de la siguiente manera:

```
std::vector<double> v(5,10);
```

Esta sentencia declara un vector `v` de 10 componentes del tipo double, teniendo cada componente del vector el valor 5.

D. (0.5 puntos) La clase `std::complex` de C++ permite representar a números complejos cuya parte real e imaginaria ha de ser del mismo tipo de dato. Este tipo de dato tiene que ser especificado al declarar una variable de esta clase.

E. (0.5 puntos) Python tiene un bucle lógico precondición, pero no tiene un bucle lógico postcondición.

F. (0.5 puntos) El algoritmo `transform` aplica una función especificada a cada uno de los elementos de la secuencia origen, almacenando el resultado en la misma secuencia origen.

Solución a la Pregunta 1

A	Falso	Véase la página 45 del texto base.
B	Falso	Véase la página 27 del texto base.
C	Falso	Véase la página 146 del texto base.
D	Verdadero	Véase la página 200 del texto base.
E	Verdadero	Véase la página 264 del texto base.
F	Falso	Véase la página 561 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 *p) {
   *p = 2*a + 1;
}

int main() {
   int *p1, *p2;
   int x = 1, y = 2, i = 0, j = 3;
   p1 = &i;
   p2 = &j;
   *p1 = 3;
   *p2 = *p1 + i;
   std::cout << i << " " << j << std::endl;
   fun1(x, &x);
   std::cout << x << " " << y << std::endl;
   fun1(x, &y);
   std::cout << x << " " << y << std::endl;
   *p2 = *p1;
   std::cout << i << " " << j << std::endl;
   std::cout << *p1 << " " << *p2 << std::endl;
   return 0;
}
```

Solución a la Pregunta 2

La salida por consola producida al ejecutar el programa es:

```
3 6
3 2
3 7
3 3
3 3
```

PREGUNTA 3 (1 punto)

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

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

int main() {
   std::list<int> listaI;
   for ( int i = 1; i < 6; i++ )
      listaI.push_back(i);
   std::list<int> listaD(listaI.size());
   std::list<int>::iterator pEndD =
      std::transform(listaI.begin(), listaI.end(),
         listaD.begin(),
         [](int n) -> int { return 10+n; });
   std::cout << "\nlistaD:";
   for ( std::list<int>::iterator p = listaD.begin();
         p != pEndD; p++ )
      std::cout << " " << (*p);
   std::list<int> listaS(listaD.size());
   std::list<int>::iterator pEndS = std::transform(
      listaD.begin(), listaD.end(), listaS.begin(),
      [](int n) -> int { return n+100; });
   std::cout << "\nlistaS:";
   for ( std::list<int>::iterator p = listaS.begin();
         p != pEndS; p++ )
      std::cout << " " << (*p);
   std::cout << std::endl;
   return 0;
}
```

Solución a la Pregunta 3

La salida por consola producida al ejecutar el programa es:

```
listaD: 11 12 13 14 15
listaS: 111 112 113 114 115
```

PREGUNTA 4 (2.5 puntos)

Escriba un programa en C++ que permita consultar el saldo de una cuenta corriente. El programa debe realizar las acciones siguientes:

1. Abrir para lectura el fichero de texto llamado *cuentas.txt*, en el cual cada línea describe una cuenta y su clave de acceso. Cada línea contiene dos números enteros separados por uno o más espacios en blanco: el número de cuenta (entero de 6 dígitos) y la clave de acceso (entero de 4 dígitos) a esa cuenta. Si se produce error, mostrar un mensaje en la consola indicándolo y terminar.

2. Leer el contenido del fichero, almacenándolo en un vector llamado `vCuentas`, de componentes de la clase struct siguiente:

   ```
   struct CuentaClave {
      int numCuenta, claveAcceso;
   };
   ```

   y a continuación cerrar el fichero. Puede asumir que el fichero no contiene dos líneas con el mismo número de cuenta.

3. Mediante un mensaje en la consola, solicitar al cliente que introduzca su número de cuenta y clave de acceso. Leer los valores introducidos por consola, almacenándolos en una variable de la clase struct `CuentaClave`.

4. Buscar en el vector `vCuentas` un elemento con el mismo número de cuenta y clave que los introducidos por el cliente. Si no existe, mostrar un mensaje en la consola indicándolo y terminar.

5. Abrir para lectura el fichero de texto llamado *saldos.txt*, en el cual cada línea describe un número de cuenta y su saldo. Cada línea contiene dos números separados por uno o más espacios en blanco: número de cuenta (entero de 6 dígitos) y saldo disponible (número de tipo real) expresado en euros. Si se produce error, mostrar un mensaje en la consola indicándolo y terminar.

6. Leer el contenido del fichero, almacenándolo en un mapa llamado `mSaldos`, cuya clave es el número de cuenta y cuyo valor es el saldo. Cerrar el fichero. Puede asumir que el fichero no contiene dos líneas con el mismo número de cuenta.

7. Buscar en el mapa `mSaldos` el elemento cuya clave coincide con el número de cuenta del cliente. Si no existe, indicarlo mediante un mensaje en la consola. Si existe, escribir en la consola el saldo de esa cuenta, en formato fijo, con dos dígitos decimales.

8. Terminar.

Solución a la Pregunta 4

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

struct CuentaClave {
   int numCuenta, claveAcceso;
};

int main() {
   std::ifstream inFich1("cuentas.txt", std::ios::in);
   if (!inFich1) {
      std::cerr << "Error al abrir el fichero\n";
      return 0;
   }
   std::vector<CuentaClave> vCuentas;
   CuentaClave cc;
   while (inFich1 >> cc.numCuenta >> cc.claveAcceso) {
      vCuentas.push_back(cc);
   }
   inFich1.close();
   std::cout << "Introduzca su numero de cuenta y clave: ";
   std::cin >> cc.numCuenta >> cc.claveAcceso;
   bool encontrado = false;
   for (unsigned i=0; !encontrado && i<vCuentas.size(); i++) {
      encontrado = ( cc.numCuenta   == vCuentas[i].numCuenta ) &&
                   ( cc.claveAcceso == vCuentas[i].claveAcceso );
   }
   if (!encontrado) {
      std::cerr << "Numero de cuenta o clave no encontrada\n";
      return 0;
   }
   std::ifstream inFich2("saldos.txt", std::ios::in);
   if (!inFich2) {
      std::cerr << "Error al abrir el fichero\n";
      return 0;
   }
   std::map<int,double> mSaldos;
   int numCuenta;
   double saldo;
   while (inFich2 >> numCuenta >> saldo) {
      mSaldos[numCuenta] = saldo;
   }
   inFich2.close();
   std::map<int,double>::iterator p = mSaldos.find(cc.numCuenta);
   if ( p != mSaldos.end() )
      std::cout << std::fixed << std::setprecision(2)
                << "Saldo: " << p->second << std::endl;
   else
      std::cerr << "Cuenta no encontrada en fichero saldo";
   return 0;
}
```

PREGUNTA 5 (2.5 puntos)

El Programa de Fidelización de clientes de una tienda sigue las reglas siguientes:

- Cada cliente del Programa de Fidelización es identificado de manera unívoca mediante un número entero mayor que cero, que es su *número de cliente*.
- Cuando un nuevo cliente se da de alta en el Programa de Fidelización, se le asigna su *número de cliente* y recibe 50 puntos de regalo.
- Cada cliente recibe puntos de regalo por cada compra que realiza en la tienda: si el cliente realiza una compra por importe de X euros, recibe un número de puntos P igual al número entero obtenido al dividir X entre 10 y despreciar todos los decimales. Por ejemplo, si el importe de la compra son $X = 39.99$ euros, entonces recibe $P = 3$ puntos. Si el importe de la compra son $X = 9$ euros, entonces recibe $P = 0$ puntos.
- En el momento en que el cliente lo desee, puede canjear parte o la totalidad de sus puntos por un regalo. El número de puntos del cliente tras el canje nunca puede ser negativo.
- Cuando el cliente se da de baja del Programa pierde todos sus puntos (pasa a tener cero puntos), pero no es eliminado de la lista de clientes, de manera que si en el futuro vuelve a darse de alta en el Programa no recibe los 50 puntos de regalo para nuevos clientes.

La información sobre los clientes del Programa de Fidelización se encuentra almacenada en una lista de elementos del tipo estructura `Cliente`, definido de la forma siguiente:

```
struct Cliente {
   int idCliente, puntos;
};
```

En el miembro `idCliente` se almacena el *número de cliente* y en el miembro `puntos` los puntos que tiene el cliente.

Se pueden realizar las cuatro operaciones siguientes de gestión del Programa de Puntos: dar de alta un nuevo cliente, incrementar los puntos de un cliente debido a que éste ha realizado una compra, decrementar los puntos de un cliente debido a que éste los ha canjeado por un regalo y dar de baja a un cliente. A tal fin, programe el código C++ descrito a continuación.

5.1 (0.5 puntos) Una función con el prototipo

```
int getPuntos(int id, std::list<Cliente> &lClientes);
```

que, inspeccionando la lista pasada como segundo argumento, devuelva el número de puntos del cliente cuyo *número de cliente* es pasado como primer argumento. Si el cliente no existe en la lista, la función devuelve el valor -1.

5.2 (0.5 puntos) Una función con el prototipo

```
int incrementaPuntos(int id, int incrementoPuntos,
                     std::list<Cliente> &lClientes);
```

que modifica la lista pasada como argumento de la forma siguiente. Si en la lista existe un cliente con el *número de cliente* pasado como primer argumento (`id`), la función incrementa en la lista los puntos de ese cliente en la cantidad pasada como segundo argumento (`incrementoPuntos`), la cual puede ser positiva o negativa. Si en la lista no existe un cliente con el *número de cliente* pasado como primer argumento, la función añade al final de la lista un nuevo cliente cuyo *número de cliente* es el pasado como primer argumento y que tiene los puntos pasados como segundo argumento. En cualquier caso, la función devuelve el número de puntos del cliente una vez realizada la operación.

5.3 (0.5 puntos) Una función con el prototipo

```
int menorIDlibre(std::list<Cliente> &lClientes);
```

que inspecciona la lista pasada como argumento y devuelve el menor número entero mayor que cero que no esta asignado como un *número de cliente*.

5.4 (1 punto) Una función con el prototipo siguiente:

```
std::string fideliza(std::string gestion,
                     int id,
                     double importeCompra,
                     int puntosCanje,
                     std::list<Cliente> &lClientes)
                 throw (std::invalid_argument);
```

la cual, dependiendo del valor del string pasado como primer argumento (`gestion`), realiza las acciones indicadas a continuación, invocando para ello, cuando sea preciso, las funciones definidas en los apartados anteriores.

1. Lanzar una excepción, finalizando con ello la función, si el string pasado como primer argumento no es igual a una de las cuatro palabras siguientes: *alta*, *compra*, *canje*, *baja*.
2. Si el string pasado como primer argumento es igual a la palabra *alta*:
   a) Obtener, inspeccionando la lista pasada como argumento, el menor número entero mayor que cero que no está asignado como un *número de cliente*.
   b) Añadir un elemento al final de la lista que describa un cliente cuyo *número de cliente* es el número calculado y que tiene 50 puntos.
   c) Devolver un string en el que se indique el *número de cliente* y su número de puntos (por ejemplo, *Id cliente: 12, puntos: 50*), finalizando con ello la función.
3. Inspeccionar la lista pasada como argumento para comprobar si existe un cliente cuyo *número de cliente* coincide con el valor pasado como segundo argumento a la función (`id`). Si no lo hay, lanzar una excepción finalizando con ello la función.
4. Dependiendo del valor del string pasado como primer argumento, realizar una de las acciones siguientes referidas al cliente cuyo *número de cliente* coincide con el valor pasado como segundo argumento a la función (`id`):
   – Si el string es igual a la palabra *compra*, incrementar en la lista el número de puntos del cliente conforme al importe de su compra. Dicho importe, expresado en euros, es el tercer argumento de la función (`importeCompra`).
   – Si el string es igual a la palabra *canje*, primeramente comprobar si el cliente tiene un número de puntos menor al número que solicita canjear (argumento `puntosCanje`), en cuyo caso debe lanzarse una excepción, finalizando con ello la función. En caso contrario, decrementar el número de puntos del cliente en la lista, conforme al canje solicitado.
   – Si el string es igual a la palabra *baja*, poner a cero el número de puntos del cliente en la lista.
5. Devolver un string en el que se indique el *número de cliente* y su nuevo número de puntos, finalizando con ello la función.

Solución a la Pregunta 5

```
#include <iostream>
#include <list>
#include <stdexcept>
#include <string>
#include <sstream>

struct Cliente {
   int idCliente, puntos;
};


int getPuntos(int id, std::list<Cliente> &lClientes) {
   std::list<Cliente>::iterator p = lClientes.begin();
   while (p != lClientes.end()) {
      if (p->idCliente == id)
         return p->puntos;
      p++;
   }
   return -1;
}

int incrementaPuntos(int id, int incrementoPuntos,
                     std::list<Cliente> &lClientes) {
   std::list<Cliente>::iterator p = lClientes.begin();
   while (p != lClientes.end()) {
      if (p->idCliente == id) {
         p->puntos += incrementoPuntos;
         return p->puntos;
      }
      p++;
   }
   Cliente nuevoCliente = {id, incrementoPuntos};
   lClientes.push_back(nuevoCliente);
   return incrementoPuntos;
}

int menorIDlibre(std::list<Cliente> &lClientes) {
   if (lClientes.size())
      for (int idLibre=1; ; idLibre++) {
         std::list<Cliente>::iterator p = lClientes.begin();
         bool existeID = false;
         while (!existeID && p != lClientes.end()) {
            existeID = ( p->idCliente == idLibre );
            p++;
         }
         if (!existeID)
            return idLibre;
      }
   return 1;
}

```

```
52
53 std::string fideliza(std::string gestion, int id,
54                       double importeCompra, int puntosCanje,
55                       std::list<Cliente> &lClientes)
56                throw (std::invalid_argument) {
57    if (gestion != "alta"  && gestion != "compra" &&
58        gestion != "canje" && gestion != "baja")
59       throw std::invalid_argument("Gestion no valida");
60    std::stringstream ss;
61    int puntos;
62    if (gestion == "alta") {
63       id = menorIDlibre(lClientes);
64       puntos = 50;
65    } else if (getPuntos(id, lClientes) == -1) {
66       throw std::invalid_argument("Id desconocido");
67    } else if (gestion == "compra") {
68       puntos = (int)importeCompra/10;
69    } else if (gestion == "canje") {
70       if (getPuntos(id, lClientes) < puntosCanje)
71          throw std::invalid_argument("Canje no valido");
72       puntos = -puntosCanje;
73    } else {  // gestion == "baja"
74       puntos = -getPuntos(id, lClientes);
75    }
76    puntos = incrementaPuntos(id, puntos, lClientes);
77    ss << "Id cliente: " << id << ", puntos: " << puntos << "\n";
78    return ss.str();
79 }
```