LENGUAJES DE PROGRAMACIÓN

Solución al examen de Junio 2019, Segunda 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) El lenguaje Pascal facilita la programación estructurada.

B. (0.5 puntos) Las sentencias **switch** de C++ han de tener siempre una etiqueta **default**.

C. (0.5 puntos) La expresión en notación prefija $* + XYZ$ es equivalente a la expresión $Z * (X + Y)$ en notación infija.

D. (0.5 puntos) En un bucle **while** de C++, la sentencia **continue** hace que el control pase al comienzo del bucle.

E. (0.5 puntos) En el cuerpo de una función en C++ pueden escribirse una o varias sentencias **return**.

F. (0.5 puntos) La función `lst.end()` devuelve un iterador a la posición siguiente al último elemento de la lista `lst`.

Solución a la Pregunta 1

A	Verdadero	Véase la página 52 del texto base.
B	Falso	Véase la página 268 del texto base.
C	Falso	Véase la página 172 del texto base.
D	Verdadero	Véase la página 278 del texto base.
E	Verdadera	Véase la página 379 del texto base.
F	Verdadero	Véase la página 456 del texto base.

PREGUNTA 2 (1 punto)

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

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

double x = 1, y = 2, z = 3;

int main () {
   double *p;
   double a[2] = {3.2382, 2.4394};
   p = a;
   x = p[1];
   std::cout << std::scientific << std::setprecision(2) <<
                "a[1] "  << a[1]  << std::endl;
   std::cout << "*p " << *p << std::endl;
   std::cout << "x "  << x  << std::endl;
   {
       double x = 40;
       double y = 25;
       a[1] = x;
       z += y;
   }
   std::cout << "*(p+1) " << *(p+1)  << std::endl;
   std::cout << std::setprecision(3) <<
                "a[1] " << a[1] << std::endl;
   std::cout << "x " << x << std::endl;
   std::cout << "y " << y << std::endl;
   std::cout << "z " << z << std::endl;
   p = p + 1;
   std::cout<< "p[0] " << p[0] << std::endl;
   std::cout<< "a[0] " << a[0] << std::endl;
   return 0;
}
```

Solución a la Pregunta 2

```
a[1] 2.44e+000
*p 3.24e+000
x 2.44e+000
*(p+1) 4.00e+001
a[1] 4.000e+001
x 2.439e+000
y 2.000e+000
z 2.800e+001
p[0] 4.000e+001
a[0] 3.238e+000
```

PREGUNTA 3 (1 punto)

Escriba la salida por consola producida al ejecutar el programa en C++ escrito a continuación. Tenga en cuenta que la función `at` del tipo de datos `std::vector` lanza una excepción del tipo `std_out_range` si se intenta acceder a una posición fuera del rango del contenedor.

```
#include <iostream>
#include <vector>
#include <stdexcept>

int main () {
   bool b1, b2, b3, b4, b5;
   int x = 3, y = 8;
   std::vector<int> a(3);
   a.push_back(1);
   a.push_back(2);
   a.push_back(3);
   b1 = !( (x<3) || (y>7) );
   b2 = (x+1) > 2  || (x+1) < -3;
   b3 =  x+1  > 2  ||  x+1  < -3;
   b4 = !(y == 0);
   b5 = (x<0) && (x/y > 0);
   try{
     std::cout << "b1 = " << b1 << std::endl;
     std::cout << "b2 = " << b2 << std::endl;
     std::cout << "b3 = " << std::boolalpha <<
                   b3     << std::endl;
     std::cout << "b4 = " << b4 << std::endl;
     std::cout << "b5 = " << b5 << std::endl;
     std::cout << x%y      << std::endl;
     std::cout << a.at(10) << std::endl;
   } catch( const std::out_of_range &oor ) {
      std::cout << "ERROR";
      return 1;
   }
   std::cout << "Programa finalizado." << std::endl;
   return 0;
 }
```

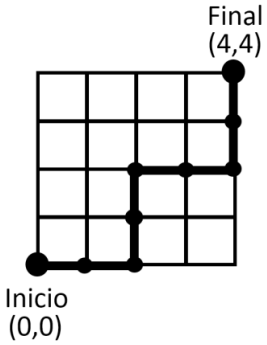
Solución a la Pregunta 3

```
b1 = 0
b2 = 1
b3 = true
b4 = true
b5 = false
3
ERROR
```

PREGUNTA 4 (2 puntos)

La distancia Manhattan, también llamada distancia L_1, es una medida de la distancia entre dos puntos aplicando la *geometría del taxista*. La distancia Manhattan es la distancia mínima que debe recorrer un taxista cuando va de un punto a otro de la ciudad, cuyas calles forman una cuadrícula.

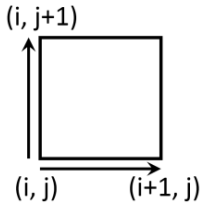
La distancia Manhattan entre dos puntos es siempre la misma y corresponde a la suma de la diferencia de coordenadas entre el punto inicial y final. En el caso del ejemplo mostrado bajo esta línea, la distancia Manhattan es 8.



Se desea calcular el número de posibles trayectorias de un taxista que recorre la distancia Manhattan entre $(0, 0)$ y otro punto cuyas coordenadas son números naturales.

El criterio para que el desplazamiento unitario sea correcto es:

– Que siempre nos acerquemos al destino en la geometría euclidiana. Por lo tanto, sólo podrá aumentar una unidad en un eje o en el otro.



– Que las coordenadas del desplazamiento no excedan el valor de las coordenadas del destino final (esto se debe cumplir al aplicar la primera condición). En el ejemplo anterior, ninguna coordenada de los puntos de las trayectorias posibles puede superar el valor 4.

Escriba un programa en C++ que solicite al usuario las coordenadas del punto destino, que calcule el número de posibles trayectorias de un taxista entre $(0, 0)$ y el punto destino, y que escriba un mensaje en la consola indicando dicho número de posibles trayectorias.

Solución a la Pregunta 4

```
#include <iostream>

int xd=0, yd=0; // Coordenadas destino

int calcTray(int x, int y) {
    if (x==xd  && y==yd) return 1;
    int nTray  = 0;
    if (x<xd) nTray  = calcTray(x+1,y);
    if (y<yd) nTray  += calcTray(x,y+1);
    return nTray;
}

int main() {
    // Entrada por consola
    std::cout << "Introduzca coordenadas destino:"
        << std::endl;
    std::cin >> xd >> yd;
    if (xd<0 || yd<0) {
        std::cout << "Coordenadas no validas" << std::endl;
        return 0;
    }
    // Escritura de resultado en la consola
    std::cout << "Numero de posibles trayectorias: " <<
          calcTray(0,0) << std::endl;
    return 0;
}
```

PREGUNTA 5 (3 puntos)

Se desea generar observaciones de una variable aleatoria discreta X, cuyo conjunto de valores posibles es $\{1, 2, \ldots, 10\}$, y cuya función de probabilidad es:

$$f_X(x) = \frac{x}{55} \qquad \text{con } x = 1, \ldots, 10 \tag{1.1}$$

Es decir, la probabilidad de que la variable valga 1 es $\frac{1}{55}$, la de que valga 2 es $\frac{2}{55}$, la de que valga 3 es $\frac{3}{55}$ y así sucesivamente.

A tal fin, primeramente se define un vector, que llamaremos $\mathbf{a} = (a_1, \ldots, a_n)$, con los $n = 55$ componentes siguientes:

$$1, 2, 2, 3, 3, 3, 4, 4, 4, 4, \ldots, 10, \ldots, 10 \tag{1.2}$$

Obsérvese que el primer componente del vector tiene valor 1, los dos siguientes valor 2, los tres siguientes valor 3, y así sucesivamente. Los 10 últimos componentes del vector tienen valor 10.

Para generar una observación de la variable aleatoria discreta X, se aplica el procedimiento siguiente:

1. Obtener una observación independiente u de una distribución $U(0, 1)$.
2. Calcular:

   $$i = \lceil u \cdot n \rceil \tag{1.3}$$

   La función $\lceil x \rceil$, denominada `ceil`, devuelve el menor número entero que es mayor o igual que x. Por ejemplo, $\lceil 3.1 \rceil$ y $\lceil 3.8 \rceil$ valen 4.

   El valor entero n es el número de componentes del vector $(a_1, \ldots, a_n)$.
3. Devolver a_i. Esto es, el componente i-ésimo del vector $(a_1, \ldots, a_n)$.

Escriba el código C++ descrito a continuación.

5.1 (0.5 puntos) Escriba una función que devuelva el vector de $n = 55$ componentes **int** que se muestra en (1.2). El prototipo de la función es el siguiente:

```
std::vector<int> vectorFuncProb();
```

5.2 (0.5 puntos) Escriba una función que lea un fichero de texto que contiene números pseudoaleatorios (observaciones independientes de una distribución $U(0,1)$) escritos en una columna y devuelva una lista de elementos **double** que contenga dichos números en el mismo orden en que aparecen escritos en el fichero. El nombre del fichero es pasado como argumento. Si se produce error al leer el fichero, la función lanza una excepción. El prototipo de la función es el siguiente:

```
std::list<double> leeFich(std::string &nombreFich)
      throw (std::invalid_argument);
```

5.3 (2 puntos) Escriba un programa en C++ que realice las acciones siguientes:

1. Declarar un vector de componentes **int** llamado `vFuncProb` y asignar valor a sus componentes tal como se muestra en (1.2), invocando para ello la función definida en el Apartado 5.1.
2. En un fichero de texto llamado `obsU.txt` hay escritos números pseudoaleatorios. El fichero contiene una única columna de números. Leer el contenido del fichero y almacenar los números en una lista llamada `numerosPseudo`, invocando para ello la función definida en el Apartado 5.2.

   Si la función lanza una excepción, el programa debe capturarla y realizar las acciones siguientes. Primeramente, mostrar un mensaje en la consola que indique que se ha producido error al abrir el fichero. A continuación, el programa debe terminar.
3. Generar tantas observaciones de la variable aleatoria discreta X como números pseudoaleatorios se han leído del fichero, empleando para ello el algoritmo descrito anteriormente. La probabilidad $f_X(x)$ de la variable aleatoria discreta X es (1.1). Las observaciones deben almacenarse en una lista llamada `distDiscreta`.
4. En las observaciones de X generadas anteriormente, calcular la proporción con la que aparece cada uno de los diez posibles valores de X: $\{1, 2, \ldots, 10\}$. Emplee para ello el algoritmo **count** de la STL de C++.

   Escribir en la consola dicha proporción, para cada uno de los 10 posibles valores de X, en formato fijo, con 4 dígitos detrás del punto decimal.
5. Terminar.

Solución a la Pregunta 5

```
#include <iostream>
#include <iomanip>
#include <fstream>
#include <string>
#include <cmath>                    // ceil()
#include <vector>
#include <list>
#include <algorithm>
#include <stdexcept>

std::string nombreFich = "obsU.txt";

std::vector<int> vectorFuncProb() {
    std::vector<int> vFuncProb;
    for (unsigned int k = 1; k <= 10; k++)
        for (unsigned int j = 1; j <= k; j++)
            vFuncProb.push_back(k);
    return vFuncProb;
}


std::list<double> leeFich(std::string &nombreFich)
                                    throw (std::invalid_argument) {
    std::ifstream inFich(nombreFich.c_str(), std::ios::in);
    if (!inFich)
        throw std::invalid_argument ("ERROR al abrir fichero " + nombreFich);
    // Lectura del fichero
    std::list<double> listaNumPseudoaleatorios;
    while ( !inFich.eof() ) {
        double num;
        inFich >> num;
        if (inFich.eof()) break;
        listaNumPseudoaleatorios.push_back(num);
    }
    inFich.close();
    return listaNumPseudoaleatorios;
}
```

```
int main() {
    // Declaración y asignación de valores al array a
    std::vector<int> vFuncProb = vectorFuncProb();
    // Lee los números de fichero y los devuelve en una lista
    std::list<double> numerosPseudo;
    try {
        numerosPseudo = leeFich(nombreFich);
    } catch (std::invalid_argument &exc) {
        std::cout << exc.what() << std::endl;
        return 0;
    }
    // Genera observaciones de la variable aleatoria discreta
    std::list<int> distDiscreta;
    std::list<double>::iterator p = numerosPseudo.begin();
    while (p != numerosPseudo.end()) {
        distDiscreta.push_back(vFuncProb[std::ceil((*p)*vFuncProb.size())-1]);
        p++;
    }
    // Escritura de resultados en la consola
    for (unsigned int k=1; k<=10; k++) {
        double p_calc =
                (double) std::count(distDiscreta.begin(), distDiscreta.end(), k) /
                 distDiscreta.size();
        std::cout << "x = " << k << "\t"
            << std::fixed << std::setprecision(4)
            << "p(calculada) = " << p_calc << std::endl;
    }
    return 0;
}
```