LENGUAJES DE PROGRAMACIÓN

Solución al examen de Junio 2014, 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) La unidad central de proceso de la máquina de Von Neumann está formada únicamente por la unidad de control.

B. (0.5 puntos) Los lenguajes funcionales permiten realizar una programación a más alto nivel que los lenguajes imperativos, pero su ejecución es menos eficiente.

C. (0.5 puntos) En Fortran los arrays de dimensión mayor que uno se almacenan en memoria por columnas.

D. (0.5 puntos) En C y C++ la expresión de control del bucle **for** es opcional.

E. (0.5 puntos) En C++ los arrays se pasan por valor.

F. (0.5 puntos) El número de comparaciones realizadas en el método de la burbuja depende de la secuencia de entrada.

Solución a la Pregunta 1

A	Falsa	Véase la página 36 del texto base.
B	Verdadera	Véase la página 63 del texto base.
C	Verdadera	Véase la página 122 del texto base.
D	Verdadera	Véase la página 274 del texto base.
E	Falsa	Véase la página 349 del texto base.
F	Falsa	Véase la página 510 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>
int main () {
   double* p;
   double a[2];
   a[0] = 0.033358333;
   a[1] = 3.12345678;
   p = a;
   std::cout << std::scientific << std::setprecision(4) <<
                p[0] <<std::endl;
   std::cout << std::setprecision(2)<< p[1] << std::endl;
   p[0] = p[0]+1.0;
   std::cout<<a[0]<<std::endl;
   p = p + 1;
   std::cout<<p[0]<<std::endl;
   std::cout<<a[0]<<std::endl;
   return 0;
 }
```

Solución a la Pregunta 2

La salida por consola producida al ejecutar el programa es:

```
3.3358e-002
3.12e+000
1.03e+000
3.12e+000
1.03e+000
```

PREGUNTA 3 (1 punto)

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

```
#include <iostream>
#include <vector>
#include <string>
int main() {
   std::vector<std::string> SS;
   SS.assign(4, "vacio ");
   SS.push_back("A");
   SS.push_back("B");
   SS.push_back("C");
   for(int ii=0; ii < (int)SS.size(); ii++)
      std::cout << SS[ii] << " ";
   std::cout << std::endl;
   SS.pop_back();
   std::vector<std::string>::const_iterator cii;
   for(cii=SS.begin(); cii!=SS.end(); cii++)
      std::cout << *cii << "  ";
   std::cout << std::endl;
   SS.erase(SS.begin()+2, SS.begin()+4);
   for(int i = 0; i<(int)(SS.size()); i++)
      std::cout << SS.at(i) << " ";
   std::cout << std::endl << "SS size: " << SS.size() << std::endl;
   std::cout << "SS[2]: " << SS[2] << std::endl;
   return 0;
}
```

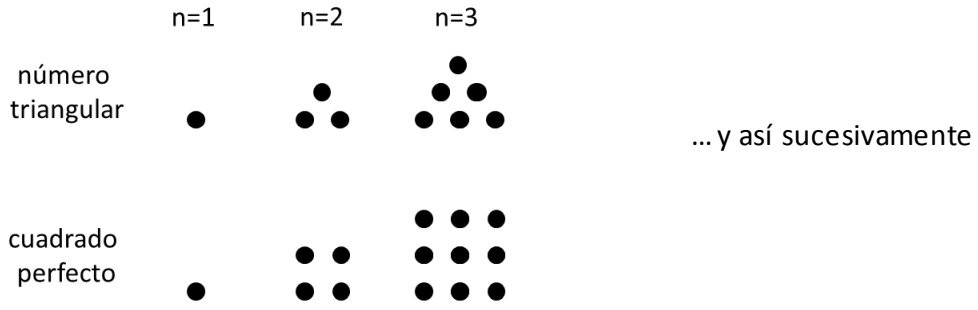
Solución a la Pregunta 3

La salida por consola producida al ejecutar el programa es:

```
vacio  vacio  vacio  vacio  A B C
vacio  vacio  vacio  vacio  A B
vacio  vacio  A B
SS size: 4
SS[2]: A
```

PREGUNTA 4 (2 puntos)

En la figura se muestra la forma que tienen los tres primeros números enteros *triangulares* y los tres primeros números enteros *cuadrados perfectos*.



Un *número entero triangular* N_t se define mediante un índice entero positivo n_t, tal que se satisface la relación siguiente:

$$N_t = \frac{n_t \cdot (n_t + 1)}{2}$$

En la parte superior de la figura se muestran los números triangulares 1, 3 y 6, cuyos índices son 1, 2 y 3 respectivamente. Obsérvese que el número N_t es el número total de puntos que componen el triángulo equilátero y el índice n_t es el número de puntos de la base del triángulo.

Un *número entero cuadrado perfecto* N_c se define mediante un índice entero positivo n_c tal que se satisface:

$$N_c = (n_c)^2$$

En la parte inferior de la figura se muestran los números cuadrados perfectos 1, 4 y 9, cuyos índices son 1, 2 y 3 respectivamente. El número N_c es igual al número total de puntos que componen el cuadrado y n_c es igual al número de puntos del lado del cuadrado.

A partir de las dos anteriores, se define una tercera categoría de números enteros: los *números cuadrados triangulares*. Un número cuadrado triangular N_{ct} es un número que es a la vez un número triangular y un número cuadrado perfecto. Es decir, existen dos números enteros positivos n_t y n_c tales que se satisface:

$$N_{ct} = \frac{n_t \cdot (n_t + 1)}{2} = (n_c)^2$$

Escriba un programa en C++ que calcule los *números cuadrados triangulares* cuyo índice n_t sea menor o igual a 10000. El programa debe realizar las acciones siguientes:

1. Declarar un array bidimensional 10×4, de tipo **double**, llamado *resultado*, en el cual se irán almacenando los resultados. Puede suponerse que los datos que van a almacenarse en el array no superarán su capacidad. Es decir, que no van a encontrarse más de 10 números cuadrados triangulares.

2. Calcular todos los números triangulares N_t tales que n_t satisface: $n_t \in [1, 10000]$. Para cada número triangular, comprobar si además es un cuadrado perfecto, en cuyo caso se ha encontrado un número cuadrado triangular.

3. Para cada número cuadrado triangular encontrado, almacenar en las sucesivas columnas de la primera fila libre del array la información siguiente:

   a) El número cuadrado triangular N_{ct}.

   b) El índice triangular n_t.

   c) El índice cuadrado n_c.

   d) El cociente n_t/n_c.
   (Únicamente a título informativo: la finalidad de almacenar este cociente es poder observar que cuando el número cuadrado triangular es grande, la relación n_t/n_c vale aproximadamente $\sqrt{2}$).

4. Mostrar en la consola el contenido almacenado en el array.

5. Terminar.

Solución a la Pregunta 4

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

const int nt_MAX = 10000;

int main () {
// Array donde se almacenarán los resultados
double resultado[10][4];

std::cout << "N_t -- n_t -- n_c -- n_t/n_c"<< std::endl;
int j=0;
for (int n_t=1; n_t <= nt_MAX; n_t++) {
     long int N_t = n_t * (n_t + 1)/2;
     if (floor(sqrt((double)N_t)) == sqrt((double)N_t)) {
          resultado[j][0] = N_t;
          resultado[j][1] = n_t;
          resultado[j][2] = sqrt((double)N_t);
          resultado[j][3] = resultado[j][1]/resultado[j][2];
          std::cout << std::fixed << std::setprecision(0) <<
                       resultado[j][0] << "--"<<
                       resultado[j][1] << "--"<<
                       resultado[j][2] << "--"<<
                       std::setprecision(6)    <<
                       resultado[j][3] << std::endl;
          j++;
     }
}
return 0;
}
```

PREGUNTA 5 (3 puntos)

Escriba en C++ una función que lea un fichero de texto que contiene números reales y los almacene en una lista, y un programa que invoque dicha función y muestre en la consola el contenido de la lista. A continuación se explica todo ello con detalle.

5.a (1.5 puntos) Defina una función en C++ que acepte dos parámetros: una referencia a una lista de elementos **double** y un string. Se muestra a continuación el prototipo de la función, la cual puede lanzar fuera de sí una excepción:

```
void leeDatos( std::list<double>&, std::string )
     throw ( std::invalid_argument );
```

La función debe realizar las acciones siguientes:

1. *Apertura para lectura de un fichero de texto*. El segundo argumento de la función contiene el nombre del fichero. El programa debe abrir el fichero de texto para lectura. Si se produce error, la función debe lanzar una excepción.
2. *Lectura del fichero de texto y almacenamiento en la lista*. El fichero de texto contiene números reales. El programa debe ir leyendo los datos del fichero y almacenándolos en la lista referenciada en el primer argumento de la función. Los elementos deben ir añadiéndose al final de la lista en el mismo orden en que son leídos.
3. *Devolver el control*, ejecutando la sentencia **return**.

5.b (1.5 puntos) Escriba un programa en C++ que realice las acciones siguientes.

1. Declarar un lista de **double** cuyo nombre debe ser *listaDatos*.
2. Invocar la función anteriormente definida, pasando como argumento una referencia a la lista anteriormente declarada y un string. Dicho string debe ser una constante global llamada *nombreFich*, cuyo valor debe ser `"datos.txt"`.
3. El programa debe estar preparado para capturar y tratar la excepción que puede ser lanzada por la función. Si se captura una excepción, el programa debe mostrar un mensaje en la consola indicándolo y a continuación debe terminar.
4. Escribir en la consola el contenido de la lista.
5. Terminar.

Solución a la Pregunta 5

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

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

void leeDatos(std::list<double>&, std::string )
                throw (std::invalid_argument);

int main() {
     std::list<double> listaDatos;
     // Invocar la función leeDatos
     try {
          leeDatos(listaDatos, nombreFich);
     } catch (std::invalid_argument exc) {
          std::cout << exc.what() << std::endl;
          return 0;
     }
     // Muestra el contenido de la lista en la consola
     std::list<double>::iterator p = listaDatos.begin();
     while (p != listaDatos.end()) {
          std::cout << *p << std::endl;
          p++;
     }
     return 0;
}

void leeDatos(std::list<double> &listaDatos,
                std::string nombreFichero)
                throw (std::invalid_argument) {
     // Apertura del fichero para lectura
     std::ifstream file_in(nombreFichero.c_str(),std::ios::in);
     if (!file_in)
          throw std::invalid_argument("No se puede abrir el fichero");
     // Leer fichero y añadir a la lista
     while (!file_in.eof()) {
          double d;
          file_in >> d;
          // Si fin de fichero, entonces salir del bucle while
          if (file_in.eof()) break;
          listaDatos.push_back(d);
     }
     return;
}
```