

LENGUAJES DE PROGRAMACIÓN

Solución al examen de Septiembre 2016

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) Las instrucciones de la máquina de Von Neumann soportan sólo los dos siguientes tipos de operaciones: operaciones aritméticas y operaciones de movimiento de datos.
- B. (0.5 puntos) En Pascal los bloques de código se delimitan por las palabras reservadas **begin** y **end**.
- C. (0.5 puntos) En Java y Fortran 90 el programador puede especificar completamente el rango de índices del array.
- D. (0.5 puntos) La variable `paco` declarada en la sentencia

```
std::list<int>::iterator paco;
```

es una lista doblemente enlazada.
- E. (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.
- F. (0.5 puntos) La función f , definida a continuación, tiene recursividad de cola.

```
int f (int n) {  
    if (n > 1) {  
        return n*f(n-1);  
    } else {  
        return 1;  
    }  
}
```

Solución a la Pregunta 1

- A Falso Véanse las páginas 37 y 38 del texto base.
- B Verdadero Véase la página 114 del texto base.
- C Falso Véase la página 121 del texto base.
- D Falso Véase la página 458 del texto base.
- E Falso Véase la página 534 del texto base.
- F Falso Véase la página 355 del texto base.

PREGUNTA 2 (1 punto)

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

```
#include <iostream>

enum color {blanco, azul, rojo, amarillo=8};

void f(int x) {
    if (x == 1)
        throw x;
    else
        std::cout << amarillo << std::endl;
}

int main() {
    try {
        f(2);
        f(1);
        std::cout << blanco << "B" << rojo << std::endl;
    } catch (int ex) {
        std::cout << azul << "A" << rojo << std::endl;
    }
    std::cout << "C" << blanco << std::endl;
    return 0;
}
```

Solución a la Pregunta 2

8
1A2
C0

PREGUNTA 3 (1 punto)

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

```
#include <iostream>

void suma10 (int a, int *p) {
    *p = a + 10;
}

int main () {
    int i, j, *p, *q;
    int x = 2, y = 6;
    p = &i;
    q = &j;
    *p = 5;
    *q = *p + i;
    std::cout << i << " " << j << std::endl;
    std::cout << *p << " " << *q << std::endl;
    suma10 (x, &x);
    std::cout << x << " " << y << std::endl;
    suma10 (x, &y);
    std::cout << x << " " << y << std::endl;
    i = 23;
    j = 72;
    *p = *q;
    std::cout << i << " " << j << std::endl;
    return 0;
}
```

Solución a la Pregunta 3

```
5 10
5 10
12 6
12 22
72 72
```

PREGUNTA 4 (5 puntos)

La constante de Kaprekar es el número 6174. Este número es el resultado de la aplicación del Algoritmo de Kaprekar, que consiste en:

- Paso 1. Escoger 4 dígitos que no sean todos iguales.
- Paso 2. Obtener el número entero mayor y el número entero menor que se puedan construir con los cuatro dígitos.
- Paso 3. Restar estos dos números enteros (el mayor menos el menor).
 - a) Si el resultado de la resta es distinto de 6174, ir al Paso2. Los cuatro nuevos dígitos a emplear son los del número entero resultado de la resta, añadiendo a la izquierda ceros si fuera necesario.
 - b) Si el resultado de la resta es 6174, terminar.

Este algoritmo converge a la constante de Kaprekar después de menos de 8 iteraciones. A continuación se muestra un ejemplo:

```
(Paso 1) - Elección inicial: 7467
(Paso 2) - Valor superior 7764, Valor inferior 4677
(Paso 3) - Resta 3087
(Paso 2) - Valor superior 8730, Valor inferior 0378
(Paso 3) - Resta 8352
(Paso 2) - Valor superior 8532, Valor inferior 2358
(Paso 3) - Resta 6174
FIN
```

Escriba el código C++ indicado a continuación.

4.1 (1.5 puntos) Escriba una función cuya declaración sea:

```
void ordenaCreciente (std::list<int> &nums);
```

que, aplicando el *método de la burbuja*, ordene crecientemente los componentes de la lista pasada por referencia.

4.2 (1 punto) Escriba una función cuya declaración sea

```
std::list<int> int2list(unsigned int num);
```

que devuelva una lista cuyos componentes sean los dígitos del número entero pasado como argumento.

4.3 (1.5 puntos) Escriba una función cuya declaración sea:

```
std::vector<int> fKaprekar (int valorInicial)
                        throw (std::invalid_argument);
```

y que realice las acciones siguientes:

1. Comprobar que el argumento es un valor inicial válido para el Algoritmo de Kaprekar. Si no lo es, lanzar una excepción.
2. Declarar un vector de **int** llamado *restas* y añadir al mismo el valor inicial.
3. Aplicar el Algoritmo de Kaprekar, empleando las funciones definidas al contestar a las Preguntas 4.1 y 4.2. En cada iteración del algoritmo, debe añadirse al final del vector el número obtenido como resultado de la resta en el Paso 3 del algoritmo.
4. Devolver el vector.

4.4 (1 punto) Escriba un programa principal que invoque repetidamente la función *fKaprekar* de la Pregunta 4.3, pasando sucesivamente como valor inicial para el algoritmo los valores 1, ..., 9998.

Si la llamada a la función genera una excepción, no se realiza ninguna acción más para ese valor inicial y se continúa con el siguiente.

Si la llamada a la función no genera una excepción, el programa debe escribir en un fichero de texto llamado *Kaprekar.txt* los componentes del vector, separados por un espacio en blanco, finalizando con un salto de línea.

Solución a la Pregunta 4

```

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

void ordenaCreciente (std::list<int> &nums){
    if (nums.size() < 2)
        return;
    std::list<int>::iterator pi = nums.begin();
    while (pi != nums.end()){
        std::list<int>::iterator pj      = nums.end();
        pj--;
        std::list<int>::iterator pj_m1 = pj;
        while (pj != pi){
            pj_m1--;
            if ( *pj < *pj_m1 ){
                int aux = *pj_m1;
                *pj_m1 = *pj;
                *pj = aux;
            }
            pj--;
        }
        pi++;
    }
    return;
}

std::list<int> int2list(unsigned int num){
    std::stringstream ss;
    ss << num;
    std::string sNum = ss.str();
    std::list<int> lnum;
    for (unsigned int i=0; i<sNum.size(); i++)
        lnum.push_back(sNum[i] - '0');
    return lnum;
}

```

```

std::vector<int> fKaprekar (int valorInicial )
    throw (std::invalid_argument){

    const int NUM_KAPREKAR = 6174;
    // Comprueba que el argumento sea un valor inicial válido
    if ( valorInicial < 1 || valorInicial > 9998 || valorInicial %
1111 == 0 ){
        std::stringstream ss;
        ss << "Valor inicial no valido: " << valorInicial;
        throw std::invalid_argument (ss.str());
    }
    std::vector<int> restas;
    restas.push_back(valorInicial);
    // Algoritmo
    unsigned int num = valorInicial;
    while ( num != NUM_KAPREKAR ){
        std::list<int> digitos = int2list(num);
        while ( digitos.size() < 4 ){
            digitos.push_back(0);
        }
        ordenaCreciente (digitos);
        std::list<int>::iterator p = digitos.begin();
        int menor = (*p)*1000 + *(++p)*100 + *(++p)*10 + *(++p);
        p = digitos.begin();
        int mayor = (*p) + *(++p)*10 + *(++p)*100 + *(++p)*1000;
        num = mayor - menor;
        restas.push_back(num);
    }
    return restas;
}

int main() {
    const char nombreFich[] = "Kaprekar.txt";
    std::ofstream fileOut(nombreFich, std::ios::out | std::ios::trunc);
    if (!fileOut ) {
        std::cout << "Error al abrir fichero" << std::endl;
        return 1;
    }
    for (unsigned int valInicial = 1; valInicial < 9999; valInicial++){
        try {
            std::vector<int> li = fKaprekar (valInicial);
            for (unsigned int i=0; i < li.size(); i++)
                fileOut << li[i] << " ";
            fileOut << std::endl;
        } catch ( std::exception exc) {
        }
    }
    fileOut.close();
    return 0;
}

```