LENGUAJES DE PROGRAMACIÓN

Solución al Ejercicio de Autocomprobación 1

PREGUNTA 1 (1 punto)

Señale razonadamente la veracidad o falsedad de las afirmaciones siguientes:

A. La diferencia fundamental entre los lenguajes ensamblador y los lenguajes máquina es que los lenguajes ensamblador son portables. Es decir, un programa escrito en ensamblador puede ser ejecutado en diferentes CPU sin necesidad de realizar cambios en el programa.

B. El lenguaje ensamblador permite asignar etiquetas a las posiciones de almacenamiento.

C. Las sentencias de control del flujo del programa de FORTRAN I se basaron en las instrucciones máquina del IBM 704, de modo que la traducción de la sentencia FORTRAN a la instrucción máquina fuera óptima.

D. La interpretación pura no permite relacionar de manera sencilla los errores de ejecución con las sentencias del código fuente que los han producido.

Solución a la Pregunta 1

La opción A es falsa, ya que ni los lenguajes máquina ni los lenguajes ensamblador son portables.

Las opciones B y C son verdaderas.

La opción D es falsa, ya que una de las ventajas de la interpretación pura es que permite relacionar de manera sencilla los errores de ejecución con las sentencias de código fuente que los han producido.

PREGUNTA 2 (1 punto)

Señale razonadamente la veracidad o falsedad de las afirmaciones siguientes:

A. Al declarar una variable hay que especificar siempre su nombre, su tipo y su valor inicial.

B. En Pascal los bloques de código vienen delimitados por llaves.

C. El ámbito de las variables globales se extiende típicamente desde el punto de la declaración hasta el final del fichero fuente.

D. Al contrario que los arrays unidimensionales, los arrays de dimensión mayor que uno no se almacenan en posiciones consecutivas de memoria.

Solución a la Pregunta 2

La opción A es falsa, ya que la variable puede no tener nombre y además no todos los lenguajes permiten inicializar variables. Por ejemplo, Pascal y Modula-2 no permiten inicializar variables.

La opción B es falsa, ya que en Pascal los bloques están delimitados por las palabras reservadas **begin** y **end**.

La opción C es verdadera.

La opción D es falsa, ya que todos los arrays, con independencia de si son unidimensionales o multidimensionales, se almacenan en posiciones consecutivas de memoria.

PREGUNTA 3 (1 punto)

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

```
#include <iostream>

int main()
{
int i1, i2, *p1, *p2;
int blocks[5] = {1, 2, 3, 4, 5};
int *ptr = &blocks[0];
int temp;

i1 = 1;
i2 = 2;
p1 = &i1;
p2 = &i2;
p1 = p2;
std::cout << "i1: Valor " << i1 <<  "\n";
std::cout << "i2: Valor " << i2 << "\n";
std::cout << "p1: Apunta al dato " << *p1 << "\n";
std::cout << "p2: Apunta al dato " << *p2 << std::endl;

i1 = 1;
i2 = 2;
p1 = &i1;
p2 = &i2;
*p1 = *p2;
std::cout << "i1: Valor " << i1 <<  "\n";
std::cout << "i2: Valor " << i2 <<  "\n";
std::cout << "p1: Apunta al dato " << *p1 << "\n";
std::cout << "p2: Apunta al dato " << *p2 << std::endl;

temp = blocks[0];
std::cout << temp << std::endl;
temp = *(blocks + 3);
std::cout << temp << std::endl;
temp = *(ptr + 2);
std::cout << temp << std::endl;
temp = *ptr;
std::cout << temp << std::endl;

ptr = blocks + 1;
temp = *ptr;
std::cout << temp << std::endl;
temp = *(ptr + 1);
std::cout << temp << std::endl;

return 0;
}
```

Solución a la Pregunta 3

A continuación se muestra la salida por consola producida al ejecutar el programa.

```
i1: Valor 1
i2: Valor 2
p1: Apunta al dato 2
p2: Apunta al dato 2
i1: Valor 2
i2: Valor 2
p1: Apunta al dato 2
p2: Apunta al dato 2
1
4
3
1
2
3
```

PREGUNTA 4 (1 punto)

Señale razonadamente la veracidad o falsedad de las afirmaciones siguientes:

A. Para evitar el problema del **else** ambiguo, Java y Ada emplean una palabra enmarcadora para la sentencia **if**.

B. Las sentencias **case** y **switch** son conceptualmente iguales.

C. En Ada y Pascal no se puede modificar el valor de la variable del bucle mediante asignaciones dentro del cuerpo del bucle.

D. En el bucle **for** de C, C++ y Java son opcionales los códigos de inicialización, de actualización y la expresión de control.

Solución a la Pregunta 4

La opción A es falsa, ya que Ada emplea una palabra enmarcadora, pero en Java el **else** se asocia con el **if** anterior más cercano, que no tenga ya una parte **else** y que se encuentre en el mismo bloque de código.

La opción B es falsa, ya que la sentencia **case** decide qué fragmento de código debe ejecutarse, mientras que la sentencia **switch** decide a qué punto de la propia sentencia debe saltar la ejecución del programa.

Las opciones C y D son verdaderas.

PREGUNTA 5 (1 punto)

Señale razonadamente la veracidad o falsedad de las afirmaciones siguientes:

A. Al implementar una lista empleando un array, las operaciones de inserción y eliminación de elementos se realizan de manera muy eficiente.

B. Una ventaja del uso de estructuras autorreferenciadas para la implementación de listas es que no es preciso reservar inicialmente una determinada cantidad de memoria, como sucede al emplear un array.

C. La pila es un tipo especial de lista, en la cual los elementos son insertados por un extremo y extraídos por el otro.

D. En la programación dinámica, al contrario que en el método divide y vencerás, los problemas son más o menos independientes.

Solución a la Pregunta 5

La opción A es falsa. La operación de inserción de elementos en la lista se realiza de manera poco eficiente, ya que es necesario desplazar todos los elementos situados en posiciones posteriores a la posición en la cual se inserta el nuevo elemento. Similarmente, cuando se elimina un elemento de la lista es necesario avanzar una posición en el array todos los elementos que se encontraban situados a continuación del elemento que ha sido eliminado.

La opción B es verdadera, ya que las estructuras autorreferenciadas emplean memoria dinámica.

La opción C es falsa, ya que la pila es un tipo especial de lista donde los elementos se insertan y eliminan sólo por uno de los extremos de la lista, que se denomina parte superior de la pila.

La opción D es falsa, ya que en la programación dinámica los problemas se solapan mientras que en la técnica divide y vencerás los problemas son más o menos independientes.

PREGUNTA 6 (2 puntos)

Escriba una función llamada `primo` que calcule los n primeros números primos, en orden creciente, comenzando desde el valor 1. La función debe tener un argumento de tipo **int**: el número de primos n a calcular. La función devuelve un vector de enteros, que contiene los n números primos calculados. A continuación se muestra la declaración de la función.

```
std::vector<int> primo(int n);
```

Solución a la Pregunta 6

La solución del ejercicio se muestra en el Código 1.1–1.2.

```
#ifndef PRIMOS_H_
#define PRIMOS_H_

#include <vector>

std::vector<int> primo(int);

#endif /* PRIMOS_H_*/
```

Código 1.1: Fichero cabecera de la función que calcula los números primos.

```
#include<iostream>
#include "primos.h"

std::vector<int> primo(int n) {
     std::vector<int> numPrimos;
     if (n>0) numPrimos.push_back(1);
     if (n>1) numPrimos.push_back(2);
     int nPrimos = 2; // Número de primos calculados hasta el momento
     // Bucle para buscar los números primos
     int numExaminar = numPrimos[1];
     while (nPrimos<n) {
          numExaminar++;
          bool noDivisible = true;
          for (int iPrimo=1; iPrimo<nPrimos && noDivisible; iPrimo++)
               noDivisible = numExaminar % numPrimos[iPrimo];
          if (noDivisible) {
               numPrimos.push_back(numExaminar);
               nPrimos++;
          }
     }
     return numPrimos;
}
```

Código 1.2: Función que calcula los n primeros números primos.

PREGUNTA 7 (3 puntos)

Escriba un programa que, dependiendo de la opción escogida por el usuario, calcule la suma de una de las tres series siguientes:

1. Serie geométrica de razón $1/3$ ($1, \frac{1}{3}, \frac{1}{9}, \frac{1}{27}, \ldots$).
2. Serie donde el término i es el inverso del i-ésimo número primo ($1, \frac{1}{2}, \frac{1}{3}, \frac{1}{5}, \frac{1}{7}, \frac{1}{11}, \ldots$).
3. Versión alterna de la serie armónica ($1, -\frac{1}{2}, \frac{1}{3}, -\frac{1}{4}, \frac{1}{5}, -\frac{1}{6}, \ldots$).

Para el cálculo de la segunda serie, se ha de emplear la función `primo`, que ha programado al resolver el ejercicio anterior. Las acciones a realizar por el programa son las siguientes:

- Solicitar al usuario que seleccione 1, 2 ó 3, según quiera sumar la serie geométrica de razón $1/3$, la serie de los inversos de los números primos o la versión alterna de la serie armónica, respectivamente.
- Solicitar al usuario que introduzca el número de términos que desea sumar.
- Realizar la suma de los términos de la serie seleccionada.
- Mostrar por consola el resultado de la suma de la serie.

Emplee para realizar este programa una sentencia **switch** y bucles **for**.

Solución a la Pregunta 7

La solución del ejercicio se muestra en el Código 1.3.

```
#include <iostream>
#include <cmath>
#include "primos.h"

int main()
{
char select;
do {
     std::cout <<
          "1. Serie geometrica de razon 1/3\n"<<
          "2. Serie de inversos de numeros primos\n"<<
          "3. Version alterna de la serie armonica\n"<<
          "Escoja (1, 2, 3): "<< std::endl;
     std::cin >> select;
} while (select!='1'&& select!='2'&& select!='3');

std::cout << "Numero de terminos a sumar: "<< std::endl;
int n;
std::cin >> n;

double suma = 0;
std::vector<int> numPrimos;
switch(select) {
     case '1':
          for (int i=0; i<n; i++)
               suma += 1.0/std::pow(3.0,i);
          break;
     case '2':
          numPrimos = primo(n);
          for (int i=0; i<n; i++)
               suma += 1.0/numPrimos[i];
          break;
     case '3':
          for (int i = 1; i<=n; i++)
               suma += std::pow(-1.0,i+1)/i;
          break;
}
std::cout << suma << std::endl;
return 0;
}
```

Código 1.3: Programa solución a la Pregunta 7