INFORMÁTICA APLICADA curso 2006-2007

PRÁCTICA 1: Memoria dinámica.

Los objetivos de esta práctica son:

- Comprender el concepto de puntero y la sintaxis de su uso en el lenguaje C.
- Utilizar el paso de parámetros a funciones por dirección por medio de punteros.
- Gestionar memoria dinámica para el manejo de vectores unidimensionales y bidimensionales.

Memoria dinámica y paso de parámetros por dirección

Esta práctica consiste en realizar un programa que realice las siguientes tareas:

a) Intercambiar en una función los valores de dos variables enteras. A continuación se muestra un ejemplo de ejecución (en negrita aparecen los datos introducidos por el usuario):

```
Introduzca a: 2
Introduzca b: 8
Inicialmente: a = 2, b = 8
Tras intercambiar: a = 8, b = 2
```

b) Invertir un vector de números reales. Para ello en primer lugar se leerán números reales positivos y se almacenarán en un vector. Se pedirá por teclado el número de elementos a introducir.

```
Introduzca la dimension del vector:
Introduzca el vector:
3.45
5.34
7.89
Vector introducido: 3.45 5.34 7.89
Vector invertido: 7.89 5.34 3.45
```

c) Sumar dos matrices cuadradas. Se deberá solicitar la dimensión de las matrices de forma que se reservará sólo la memoria necesaria para almacenar los elementos de las matrices (memoria dinámica).

d) Crear un menú para elegir cada una de las opciones que se pueden ejecutar en el programa.

Tareas:

1) Crear y configurar un proyecto en Dev C++ que se llamará **practica1**.

2) Definir la función **void Intercambiar(int *p1, int *p2);** que debe intercambiar el valor de dos valores enteros pasados como parámetros (paso por dirección).

3) Para invertir un vector de números reales deberán realizarse los siguientes pasos:

   a) Reservar memoria estática para el vector de reales en el que se guardarán los números introducidos por el usuario. Se considerará que el número máximo de elementos del vector es 50.

   b) Definir la función **void LeerVector(float *v, int n)** que leerá números reales y los almacenará en el vector *v* pasado como parámetro hasta la dimension n.

   c) Reservar la memoria necesaria para el vector de reales en el que se guardarán los elementos de forma invertida.

   d) Definir la función **void InvertirVector(float *v, float *inv, int n)** que copia en el vector *inv* los *n* elementos de *v* de forma inversa.

   e) Definir la función **void MostrarVector(float *v, int n)** para mostrar los elementos del vector *v* pasado como parámetro. El parámetro *n* indica el número de elementos del vector.

   f) Una vez efectuadas todas las operaciones es necesario liberar la memoria previamente reservada.

4) Para realizar la suma de dos matrices se recomienda seguir los siguientes pasos:

   a) Reservar memoria de forma dinámica para tres matrices: las dos matrices a sumar más la matriz donde se almacenará el resultado. Para ello previamente será necesario conocer la dimensión de las matrices, que será introducida por teclado.

   b) Crear la función **void LeerMatriz(float **m, int n)** para leer una matriz por teclado de dimensión *n*.

   c) Definir la función **void SumarMatrices(float **ma, float **mb, float **sum, int n)** que realizará la suma de las matrices *ma* y *mb* y almacenará el resultado en la matriz *sum*.

   d) Definir la función **void MostrarMatriz(float **m, int n)** que muestra en pantalla la matriz *m* de dimensión *n*.

   e) Eliminar la memoria de las matrices previamente reservada.

5) Definir la función principal del programa (**main**).

6) El programa deberá mostrar el siguiente menú, ejecutándose cada una de las funciones antes definidas según indique el usuario.

```
Practica 1

1.- Intercambiar dos numeros
2.- Invertir un vector de números reales
3.- Sumar dos matrices cuadradas
4.- Salir

Elija una opcion:
```

Notas:

- Para la asignación dinámica de memoria debe utilizarse la función **void *malloc (tamaño)** (cabecera *stdlib.h*). *tamaño* es el número de bytes de memoria que se quiere reservar. La función devuelve un puntero vacío (*void* *) al espacio de memoria reservado. Este puntero se convertirá al puntero del tipo que se desee. En caso de que no haya memoria disponible la función devuelve NULL.

- Para liberar memoria reservada dinámicamente debe utilizarse la función **void free(void *puntero)** (cabecera *stdlib.h*). Esta función libera la memoria a la que apunta *puntero* previamente reservada.

Se da a continuación una plantilla para realizar la práctica:

```
//*********************************************************************
//    NOMBRE DEL ARCHIVO:     practica1.c
//    MÓDULO:
//    FECHA:
//    AUTOR:                   Arturo Gil
//    DESCRIPCIÓN:
//*********************************************************************
#define  __PRACTICA1_C__

// Librerías Estándar
#include <stdio.h>

// Otras librerías
// Encabezado Módulo-Proyecto
// Encabezado del archivo
#include "..\inc\practica1.h"

// Variables Globales externas
// Variables Globales internas

void main(void)
{
       int opcion; //Opcion del programa
       int n, i; //Tamaño del vector
    int a, b; //Numeros para intercambiar
     float v[MAXVECTOR];
       float inv[MAXVECTOR];

 float **mat1, **mat2, **mat3; //Reserva de memoria para las matrices
```

```
	printf("Practica 1\n\n");

	do
	{
		printf("1.- Intercambiar dos números\n");
		printf("2.- Invertir vector de numeros reales\n");
		printf("3.- Sumar dos matrices cuadradas\n");
		printf("4.- Salir\n");
		printf("\nElegir opcion: ");
		scanf("%d", &opcion);
		switch(opcion){

		case 1:
			printf("\nIntroduzca a: ");
			scanf("%d", &a);
			printf("\nIntroduzca b: ");
			scanf("%d", &b);
		   printf("\nAntes de intercambiar. a=%d, b=%d", a, b);
		   Intercambiar(.....);
		   printf("\nDespues de intercambiar. a=%d, b=%d", a, b);
				break;

		case 2:
		printf("\nIntroduzca el numero de elementos del vector: ");
		scanf("%d", &n);
		LeerVector(....);
		MostrarVector(....); //Mostrar vector v
		InvertirVector(....);
		MostrarVector(....); //Mostrar vector invertido inv
				break;

		case 3:
		printf("\nIntroduzca la dimension de las matrices (cuadradas) ");
				scanf("%d", &n);
			mat1 = (float **)malloc(n*sizeof(float*));
			for(i=0; i < n; i++)
				mat1[i] = (float *)malloc(n*sizeof(float));
			LeerMatriz(...);
			mat2 = (float **)malloc(n*sizeof(float*));
			for(i=0; i < n; i++)
				mat2[i] = (float *)malloc(n*sizeof(float));

			break;
	      case 4:
				salir=1;
				break;

			default:
				printf("Opcion incorrecta\n\n");
				break;

		}

	}while(salir != 1);
}

void Intercambiar(int *p1, int *p2)
{

...

}

//Recoge por teclado los elementos del vector v
void LeerVector(float *v, int n)
{
	int i;

	for (i=0; i<n; i++)
	{
		printf("Introduce el elemento %d: ",i);
		scanf("%f", &v[i]);
	}
}

//Invierte el orden de los elementos de v y los almacena en inv
```

```
void InvertirVector(float *v, float *inv, int n)
{

...

}

//Muestra por pantalla los elementos del vector hasta la dimension n
void MostrarVector(float *vector, int n)
{

}
```

```
//*********************************************************************
//
//    NOMBRE DEL ARCHIVO:     practica1.h
//    MÓDULO:
//    FECHA:
//    AUTOR:
//    DESCRIPCIÓN:
//
//*********************************************************************
#ifndef  __PRACTICA1_H__
#define  __PRACTICA1_H__
//---------------------------------------------------------------------
// Bloque de definiciones  comunes
//---------------------------------------------------------------------
// Definición de constantes
#define MAXVECTOR 50
// Definición de macros

// Definición de tipos de datos y clases

// Prototipos de funciones globales
void Intercambiar(int *p1, int *p2);
void LeerVector(float *v, int n);
void InvertirVector(float *v, float *inv, int n);
void MostrarVector(float *v, int n);
//---------------------------------------------------------------------
#ifndef     __PRACTICA1_C__
//---------------------------------------------------------------------
// Bloque de definiciones externas

// Variables Globales externas

//---------------------------------------------------------------------
#else
//---------------------------------------------------------------------
// Bloque de definiciones internas

// Prototipos de funciones internas

void Intercambiar(int *p1, int *p2);
void LeerVector(float *v, int n);
void InvertirVector(float *v, float *inv, int n);
void MostrarVector(float *v, int n);

//---------------------------------------------------------------------
#endif            // Fin bloque de definiciones internas
#endif            // Fin bloque de la cabecera
```