Ejercicios del Tema 5

Punteros

1. Ejercicios resueltos

1.1. Cambio de dos variables

Ejemplo de una función que, mediante el paso por referencia, intercambia el valor de dos variables.

```c
#include <stdio.h>

void cambio (int *p, int *q);

int main()
{
  int a = 1, b = 7;

  printf("a vale %d y b vale %d\n", a, b);
  // Función que intercambia el valor de dos variables, accediendo a
  // su dirección de memoria dado que usa el paso por referencia (no
  // entrega ningún resultado)
  cambio(&a, &b);

  printf("a vale %d y b vale %d\n", a, b);

  return 0;
}

void cambio (int *p, int *q)
{
  int aux;
  aux = *p;
  *p = *q;
  *q = aux;
}
```

1.2. Ecuación de segundo grado

```c
#include <stdio.h>
#include <math.h> //para la funcion raiz cuadrada sqrt

// Funcion que combina paso por valor (a, b , c) y referencia (r1 y r2)
int raices(float a, float b, float c, float *r1, float *r2);

int main()
{
  float a, b, c;//coeficientes
  float r1, r2;//raices, soluciones
  int ret;

  printf("Introduzca los coeficientes a, b y c.\n");
```

```
 printf("de ax2 + bx + c = 0, separados por espacios.\n");

 scanf("%f %f %f", &a, &b, &c);

 printf("Ecuacion %f x2 + %f x + %f = 0\n",a,b,c);

 ret = raices(a, b, c, &r1, &r2);

 switch(ret)
  {
  case 0: printf("Soluciones reales %f %f\n", r1, r2);
   break;
  case 1: printf("Raiz doble %f\n", r1);
   break;
  case 2: printf("Raices complejas conjugadas %f+-i%f\n", r1, r2);
   break;
  case 3: printf("Ecuacion lineal, raiz: %f\n", r1);
   break;
  case 4: printf("Error parametros\n");
   break;
  }
}
// Los datos iniciales se pasan por valor (lectura), y los resultados
// por referencia (escritura)
int raices(float a, float b, float c, float *r1, float *r2)
{
 if(a != 0.0) //ecuacion cuadrada
  {
   float disc;//discriminante
   disc = b * b - 4 * a * c;
   if(disc > 0)//dos soluciones reales
    {
     *r1 = (-b + sqrt(disc)) / 2 * a;
     *r2 = (-b - sqrt(disc)) / 2 * a;
     return 0;
    }
   else if(disc == 0)//raiz doble
    {
     *r1 = *r2 = -b / 2 * a;
     return 1;
    }
   else//soluciones complejas
    {
     *r1 = -b / 2 * a;
     *r2 = sqrt(-disc) / 2 * a;
     return 2;
    }
  }
 else //ecuacion no cuadrada
  {
   if(b != 0.0)
    {
     *r1 = -c / b;
     return 3;
    }
   else
    return 4;
  }
}
```

2. Ejercicios propuestos

2.1. Longitud de una cadena de caracteres

Diseñe un programa que calcule la longitud de una cadena de caracteres usando una función basada en punteros.

```
int stringLength(char *string);
```

```
#include <stdio.h>

int stringLength(char *string);

int main()
{
  int len;
  char mensaje[] = "Hello World!";

  len = stringLength(mensaje);

  printf("Longitud %i\n", len);

  return 0;
}


int stringLength(char *string)
{
  // p es un puntero movil, que apunta inicialmente al primer caracter
  // de la cadena string
  char *p = string;
  // Movemos el puntero a lo largo de la cadena hasta llegar al final
  while (*p != '\0')
    ++p;
  // La distancia entre el puntero p y string (primer caracter) es el
  // numero de caracteres de la cadena
  return p - string;
}
```

2.2. Copia de cadenas de caracteres

Diseñe un programa que copie dos cadenas de caracteres usando una función basada en punteros. Esta función imita el comportamiento de la función `strcpy` de la librería `string.h` (véase el capítulo 4, diapositiva 40).

```
void copyString(char *to, char *from);
```

```
#include <stdio.h>

int copyString(char *to, char *from);

int main()
{
  char mensaje1[] = "Hello World!";
  char mensaje2[20];

  copyString(mensaje2, mensaje1);

  printf("%s\n", mensaje2);
```

```
  return 0;
}

int copyString(char *to, char *from)
{
  while (*from != '\0')
    {
      *to = *from;
      to++;
      from++;
    }
  // Añadimos el caracter 0 como cierre de la cadena
  *to = '\0';
}
```

2.3. Suma de un vector

Diseñe un programa que calcule la suma de un vector usando una función basada en punteros.

```
int sumaVector(int *vector, int n);
```

```
#include <stdio.h>

int sumaVector(int *vector, int n);

int main()
{
  int vals[] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
  int suma;

  suma = sumaVector(vals, 9);

  printf("La suma es %i.\n", suma);

  return 0;
}


int sumaVector(int *vector, int n)
{
  int i, suma = 0;
  for (i = 0; i < n; i++)
    suma += *(vector + i); // equivalente a vector[i]

  return suma;
}
```

2.4. Media, máximo y mínimo de un vector de datos

Escribe un programa que calcule el valor medio, el valor máximo y el valor mínimo de los datos contenidos en un vector de números reales empleando punteros. La función de cálculo debe acomodarse al siguiente prototipo:

```
void stats(float *vector, int n, float *media, float *max, float *min);
```

Debe escribir dos versiones diferentes. La primera versión trabajará con un vector de dimensión fija y conocida (p.ej. 5 elementos). La segunda versión debe emplear asignación dinámica de memoria, siguiendo el mismo itinerario que el ejercicio anterior.

Nota: Este ejercicio es equivalente al ejercicio 6 del capítulo 4 (vectores). En aquel caso se usa notación de vectores, y ahora se debe emplear notación de punteros. Además, allí se usan tres funciones independientes, y ahora se debe usar una única función que entrega los tres resultados.

```
#include <stdio.h>
#define N 5

void stats(float *vector, int n, float *media, float *max, float *min);

int main()
{
  float vector[N] = {1.0, 2.0, 3.0, 4.0, 5.0};
  float media, max, min;

  stats(vector, N, &media, &max, &min);

  printf("Media %f, Máximo %f, Mínimo %f",
        media, max, min);

  return 0;
}


void stats(float *vector, int n, float *media, float *max, float *min)
{
  float suma = 0;
  float *p;
  float *fin = vector + n;

  // Media
  for (p = vector; p < fin; ++p)
    suma += *p;

  *media = suma / n;

  // Maximo
  *max = vector[0]; //El maximo inicial es el primer elemento
  // Busca en el vector algún valor mayor
  for (p = vector; p < fin; ++p)
    if (*p > *max) *max = *p; // y lo sustituye si lo encuentra

  //Minimo
  *min = vector[0];
  for (p = vector; p < fin; ++p)
    if (*p < *min) *min = *p;
}
```

2.5. Movimiento de un punto (estructura)

Construye un programa que sirva para mover un punto contenido en el plano (usando un tipo de datos `punto`) usando funciones basada en punteros a la estructura `punto`. El programa funciona con la interacción del usuario a partir de las teclas 'w' (arriba), 'a' (izquierda), 'x' (abajo), 'd' (derecha), y sus correspondientes mayúsculas. Por ejemplo, si el usuario aprieta la tecla 'w' el punto se desplaza hacia arriba una unidad, y si aprieta 'W' se desplaza cinco unidades hacia arriba. El programa muestra en pantalla las coordenadas del punto tras cada interacción del usuario. El programa termina cuando el usuario pulsa la tecla 'q'.

```
// Función para controlar el desplazamiento horizontal, siendo xy el
// punto, y n el número de unidades a desplazar
void horizontal(punto *xy, int n);
// Función para controlar el desplazamiento vertical
```

```
void vertical(punto *xy, int n);
```

```
#include <stdio.h>

typedef struct
{
  int x, y;
} punto;


// Función para controlar el desplazamiento horizontal, siendo xy el
// punto, y n el número de unidades a desplazar
void horizontal(punto *xy, int n);
// Función para controlar el desplazamiento vertical
void vertical(punto *xy, int n);

int main()
{
  punto p1 = {0, 0};
  char tecla;
  printf("(%i, %i)\n", p1.x, p1.y);
  while (tecla != 'q')
    {
      fflush(stdin);
      scanf("%c", &tecla);
      switch(tecla)
        {
        case 'a':
          horizontal(&p1, -1);
          break;
        case 'd':
          horizontal(&p1, 1);
          break;
        case 'A':
          horizontal(&p1, -5);
          break;
        case 'D':
          horizontal(&p1, 5);
          break;
        case 'w':
          vertical(&p1, 1);
          break;
        case 'x':
          vertical(&p1, -1);
          break;
        case 'W':
          vertical(&p1, 5);
          break;
        case 'X':
          vertical(&p1, -5);
          break;
        }

      printf("(%i, %i)\n", p1.x, p1.y);
    }
  return 0;
}


void horizontal(punto *xy, int n)
{
  xy->x += n;
}
```

```
void vertical(punto *xy, int n)
{
   xy->y += n;
}
```

2.6. Asignación dinámica de memoria

Escribe un programa que permita al usuario rellenar un vector de dimensión variable. En primer lugar, el programa pregunta al usuario el número de elementos que tendrá el vector. A continuación, empleando asignación dinámica de memoria, el programa reservará memoria para este vector, y solicitará al usuario los valores del mismo. Finalmente, el programa mostrará el contenido de este vector.

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
  int i, n;
  int *p;

  printf("Nº datos? ");
  scanf("%d", &n);
  // Reservamos n direcciones de memoria del tamaño de un int. El
  // resultado de malloc es la dirección inicial, que almacenamos en
  // un puntero. Este puntero equivale al elemento 0 de un vector de n
  // elementos.
  p = malloc(sizeof(int) * n);
  // Comprobamos que hay memoria disponible, y comunicamos error en su
  // caso
  if (p == NULL)
    {
      printf("Error de memoria.\n");
      exit(-1);
    }
  // Si todo ha ido bien recorremos el puntero-vector para rellenarlo
  // de contenido
  for(i = 0; i < n; i++)
    {
      scanf("%d", p + i);
    }
  // Y ahora lo mostramos
  for(i = 0; i < n; i++)
    {
      printf("%d", *(p + i));
    }
  // Una vez terminado, liberamos la memoria
  free(p);
}
```