Programación Procedural

Trabajo Práctico 10

Programación Procedural

Trabajo Práctico 10: Punteros y Estructuras de datos.

1. Indicar los valores de las variables en función de las declaraciones:

```
int x = 14, y = 7;
int *k = &x;
int *r = &y;
```

x = [_____] &x = [_____]

y = [_____] &y = [_____]

k = [_____] &k = [_____]

r = [_____] &r = [_____]

Considerar que las variables están almacenadas en las siguientes direcciones de memoria:

x -> 0x13ff6c

y -> 0x13ff68

k -> 0x13ff60

r -> 0x13ff5c

2. Indicar al lado de cada sentencia el significado de la misma:

```
Int x = 1, y = 2, z[10];

Int *ip;          //se declara ip como puntero a un tipo de dato entero

ip = &x; . . . . . .. . . . . .
```

y = *ip; . . . . . .

*ip = 0; . . . . . .

ip = &z[0]; . . . . . .

3. Indicar cuál es la salida del siguiente programa:

```
#include <stdio.h>
void main(void)
{
        char c='\0', *pc=NULL;
        pc=&c;
        for(c='A'; c<='Z';c++)
                printf("%c",*pc);
        return;
}
```

4 Realizar la traza del siguiente programa (incluir la salida en pantalla), sabiendo que la primera posición de la matriz x está en la dirección 0x12fedc y que k está en la dirección 0x12feec:

```
#include <stdio.h>
#include <stdlib.h>
void iniciar(int m[2][2],int n)
{
        printf("n = %d (0x%x) \n",n,&n);
        for(int i=0;i < 2;i++)
        {
                for(int j=0;j < 2;j++)
                {
                        m[i][j] = n;
                        printf("m[%d,%d] = 0x%x \t",i,j,&m[i][j]);
                }
                printf("\n");
        }
}
void main()
{
        int x[2][2];
        int k = -1;
        iniciar(x,k);
        printf("\n");
        printf("k = %d (0x%x) \n",k,&k);
        for(int i=0;i < 2;i++)
```

```
        {
                for(int j=0;j < 2;j++)
                        printf("x[%d,%d] = %d (0x%x) \t",i,j,x[i][j],&x[i][j]);
                printf("\n");
        }
        return;
}
```

5. Indicar cuál será la salida en pantalla del siguiente código, sabiendo que la primera posición del vector está en 0x12FF58:

```
#include  <stdio.h>
#include <stdlib.h>
int main()
{
        int x[5] = {2,4,6,8,10};
        int i;
        for(i=0;i < 5;i++)
        {
                printf("x[%d] = %d \n",i,x[i]);
                printf("&x[%d] = 0x%x \n",i,&x[i]);
        }
        return 0;
}
```

6. Suponer que se han definido dos registros de la siguiente manera:

```
Struct tipo_a
{
        char nombre[30];
        int precio;
        int fecha[3];
        char cod;
};

struct tipo_a A ;

struct tipo_b
{
        char nom_ap[30];
        float precio;
        int dia;
        int mes;
        int annio;
```

```
};

struct tipo_b *B ;
```

Indicar con una cruz cuáles de las siguientes asignaciones son sintácticamente incorrectas. Explicar por qué.

A.nombre = nom_ap;	[ ]
B->precio = B->precio + 0.5 ;	[ ]
B.nom_ap ="Juan Perez"	[ ]
A.cod = 'm'	[ ]
A.fecha[2] = B->mes;	[ ]
B.mes = A.fecha[2];	[ ]
B->precio= A.cod;	[ ]
A.fecha[] = {15, 9, 2005};	[ ]
B->precio = A.precio;	[ ]
B.precio = A.precio;	[ ]

7. Dada la siguiente definición de estructura en lenguaje C:

```
struct trabajadores
{
        char nombre [LONGITUD];
        int legajo, edad;
};
```

Realice un programa en lenguaje C que permita la carga de 10 trabajadores. La función de carga

debe tener el siguiente prototipo:

```
int function carga_datos(struct trabajadores datos[]) // devuelve la cantidad de trabajadores cargados
```

A su vez esta función llamará a otra función específica para la carga de datos de un trabajador, con el siguiente prototipo:

void function carga_uno(**struct** trabajadores *dato)

El programa deberá permitir al usuario realizar las siguientes tareas:

a) Carga de los datos de los trabajadores.
b) Listar los datos de los trabajadores cargados previamente.
c) Mostrar los datos de un determinado trabajador.

8. Reescribir la función carga_uno del ejercicio anterior con el siguiente prototipo:

struct trabajadores function carga_uno()

Cuál es la diferencia entre ambas funciones

9. Escribir en lenguaje C una función con el siguiente prototipo.

double *menorVector(**double** *v, **int** n)

El primer argumento se debe corresponder con un vector y el segundo con el número de elementos del vector. La función debe devolver la dirección del elemento menor. El vector se debe recorrer con aritmética de punteros.