Práctica 4: Punteros, Funciones y Memoria dinámica

Cátedra Programación II Octubre 2023

Para los siguientes ejercicios se considerarán los siguientes tamaños de los tipos de datos básicos:

Tipo	Tamaño (bytes)
char, unsigned char	1
short int, unsigned short int	2
int, unsigned int, long int, unsigned long int	4
float	4
double, long double	8
Puntero de cualquier tipo	4

1. Estructuras

Ejercicio 1. ¿Cuál es la salida de los siguientes programas con punteros? Explique su respuesta.

a)

```c
#include<stdio.h>

int main() {
    int* punt;
    int x = 7, y = 5;
    punt = &x;
    *punt = 4;
    printf("%d %d", x, y);
    return 0;
}
```

b)

```c
#include<stdio.h>

int main() {
    int* punt;
    int x = 7, y = 5;
    punt = &x;
    x = 4;
    punt = &y;
    printf("%d %d", *punt, x);
    return 0;
}
```

c)

```c
#include<stdio.h>

int main() {
    int* punt, i;
    int x[] = {1, 2, 3, 4, 5};
    punt = x;
    *punt = 9;
    for (i = 0; i < 5; i++)
        printf("%d, ", x[i]);
    return 0;
}
```

d)

```c
#include<stdio.h>

int main() {
    int* punt, i;
    int x[] = {1, 2, 3, 4, 5};
    punt = x;
    *(punt + 2) = 9;
    *(x + 3) = 7;
    punt[1] = 11;
    for (i = 0; i < 5; i++)
        printf("%d, ", *(punt+i));
    return 0;
}
```

e)

```c
#include<stdio.h>

int main() {
    int *punt, i;
    int x[5] = {1, 2, 3, 4, 5};
    punt = &x[0] + 3;
    *(punt - 2) = 9;
    punt--;
    *(punt) = 7;
    punt[1] = 11;
    punt = x;
    for(i = 0; i < 5; i++)
        printf("%d, ", punt[i]);
    return 0;
}
```

f)

```c
#include<stdio.h>

int main() {
    int v[4] = {2, 4, 5, 7}, a, *p;
    p = v + 2;
    p--;
    a = *p + *(p + 1) + *(v + 1) + p[2];
    printf("%d", a);
    return 0;
}
```

g)

```c
#include<stdio.h>

void aumentar(int* x, int* y, int z){
    *x = *x + 2;
    *y = *y + 5;
    z = z + 4;
}
int main() {
    int x, y, z;
    x = 3;
    y = 10;
    z = 15;
    aumentar(&x, &y, z);
    printf("%d %d %d", x, y, z);
    return 0;
}
```

h)

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

void ingreseCadena(char* c){
    fgets(c, 10, stdin);
}

int main() {
    char* cadena = malloc(sizeof(char) * 10);
    fgets(cadena, 10, stdin);
    printf("%s\n", cadena);
    ingreseCadena(cadena);
    printf("%s", cadena);
    free(cadena);
    return 0;
}
```

i)

```c
#include <stdio.h>

int *direccion_local(int x) {
    return &x;
}

int main() {
    int *ptr = NULL;
    ptr = direccion_local(2);
    printf("%d\n", *ptr);
    return 0;
}
```

Ejercicio 2. Analice los programas dados a continuación. ¿Cuál es la salida por pantalla del programa? ¿Hay errores en el manejo de memoria? ¿Cómo solucionaría estos errores?

a)

```c
#include<stdio.h>

int main() {
    char textoA[30] = "Agarrate Catalina";
    char textoB[30] = "El Cuarteto de Nos";
    char* p = textoA;
    char* q = textoB;
    char a;
    printf("textoA: %s\ntextoB: %s\n", textoA, textoB);
    while(*p++ = *q++) a = a;
    printf("while(*p++ = *q++);\n");
    printf("textoA: %s\ntextoB: %s\n", textoA, textoB);
    return 0;
}
```

b)

```c
#include<stdio.h>

int main() {
    int array[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int* ptr;
    ptr = array;
    printf("array[0]>%d; *ptr: %d\n", array[0], *ptr);
    printf("array[1]>%d; *(ptr+1): %d\n", array[1], *(ptr+1));
    ptr++;
    printf("ptr++; *ptr: %d\n", *ptr);
    return 0;
}
```

c)

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

char* copiar_cadena(char* cad, int longitud){
  char* a = malloc(sizeof(char) * longitud);
  a = cad;
  return a;
}

int main(){
  char a[10] = "hola";
  char* b = copiar_cadena(a, 10);
  printf("%s %s\n", a, b);
  b[0] = 's';
  printf("%s %s\n", a, b);
  return 0;
}
```

2. Memoria dinámica

Ejercicio 3. Considere el siguiente código y luego responda las preguntas a continuación.

```c
int vector[5] = {10, 20, 30, 40, 50};

#define SIZE 100

/* Informacion sobre la celda */
struct informacionCelda {
   char nombre[SIZE];          //nombre de la celda
   int identificador;         //número identificador
   float calidad;          //calidad de la señal (entre 0 y 100)
   struct informacionOp* op; //puntero a una segunda estructura
};
struct informacionOp {
   char nombre[SIZE];         //nombre del operador
   int prioridad;         //prioridad de conexión
   int ultimaComprobacion;  //fecha de la ultima comprobación
};
```

a) ¿Cuántos bytes ocupa una variable de tipo `struct informacionCelda` en memoria?

b) Las siguientes dos líneas declaran dos variables. ¿Cuánto ocupa cada una de ellas en memoria?

```
struct informacionCelda a;
struct informacionCelda* b;
```

c) Si una variable de tipo `struct informacionCelda` está almacenada en la posición de memoria 100, ¿qué dirección tienen cada uno de sus campos?

d) ¿Se pueden hacer las siguientes asignaciones? Explique lo que se realiza en cada una de ellas.

```
struct informacionCelda c;
struct informacionCelda* cptr = &c;
struct informacionCelda d;
struct informacionCelda* dptr = cptr;
```

Ejercicio 4. Considere el siguiente código.

```
struct pack3 {
   int a;
};

struct pack2 {
   int b;
   struct pack3 *next;
};

struct pack1 {
   int c;
   struct pack2 *next;
};

int main(){
   struct pack1 data1, *dataPtr;
   struct pack2 data2;
   struct pack3 data3;
   data1.c = 30;
   data2.b = 20;
   data3.a = 10;
   dataPtr = &data1;
   data1.next = &data2;
   data2.next = &data3;
   return 1;
}
```

Analice si las siguientes expresiones son correctas y, en caso de serlo, dé el valor de las mismas.

Expresion	**Correcta**	**Valor**
`data1.c`		
`dataPtr->c`		
`dataPtr.c`		
`data1.next->b`		
`dataPtr->next->b`		
`dataPtr.next.b`		
`dataPtr->next.b`		
`(*(dataPtr->next)).b`		
`data1.next->next->a`		
`dataPtr->next->next.a`		
`dataPtr->next->next->a`		
`dataPtr->next->a`		
`dataPtr->next->next->b`		