Práctica 0 - Repaso C

Gestión de la memoria y punteros

1. Escriba un programa que declare algunas variables locales, e imprima las direcciones de memoria de las mismas. Pruebe declarar un arreglo de caracteres, y verifique que las direcciones de sus elementos son contiguas.

2. Implemente una función **void set_first(int [])** que ponga en cero el primer elemento del arreglo recibido. Verifique desde la función llamante que efectivamente modifica este valor. ¿Por qué pasa esto? ¿No llama a la función por valor?

3. Implemente una función **set_in(int *)** que toma un puntero a un entero, y reemplaza el entero apuntado por un `1` si el entero apuntado era diferente a `0`, y `0` en caso contrario.

4. Implemente una función **void swap(int *, int *)** que dados dos punteros a variables, intercambie el contenido de las variables apuntadas.

5. Explique el tipo de la función **malloc** de stdlib.h, ¿qué valor retorna la función en caso de que no pueda reservar el espacio solicitado?

6. Implemente una función **char *get_new_line(void)** que ingrese una linea por teclado (hasta `\n`), y devuelva un puntero a la cadena ingresada.

7. Escriba un programa que reserve un espacio de memoria de 100 bytes, y luego libérelo dos veces. ¿Se produce algún error?

8. Implemente las siguientes funciones que reciben como argumento punteros a función:

a) **int apply(int (*)(int), int)** que toma un puntero a función, y un entero, y aplica la función al entero y retorna el valor dado.

b) **void apply_in(int (*)(int), int*)** que toma un puntero a función, un puntero a un entero, y reemplaza el entero apuntado por el valor de ejecutar la función apuntada sobre el valor apuntado.

c) **void recorre(VisitorFunc, int[], int)** que toma un puntero a una función, un arreglo de enteros, y su longitud, y aplica la función a cada elemento del arreglo. `VisitorFunc` está definido por `typedef void (*VisitorFunc)(int)`.

d) Pruebe las funciones anteriores pasándoles como parámetro las siguientes funciones:

i. Para **a** y **b**:

```
int sucesor (int n) {
    return n+1;
}
```

ii. Para **c**:

```
void imprimir (int n) {
    printf("%d \n", n);
}
```

9. ¿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[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;
}
```

e)
```c
#include<stdio.h>
```

```c
void aumentar_punteros(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_punteros(&x, &y, z);
   printf("%d %d %d", x, y, z);
   return 0;
}
```

f)

```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;
}
```

g)

```c
#include<stdio.h>

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

10. Analice el programa dado a continuación ¿Cuál es el resultado? ¿Hay errores en el manejo de memoria? ¿Cómo solucionaría estos errores?

```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;
}
```

11. Para cada uno de los programas que se listan a continuación, elija la salida que supone que ocurrirá al ejecutarlo, luego compruebe si estaba en lo cierto. ¿Por qué se obtuvo cada resultado?

a)

```
#include<stdio.h>

void nullify(int* a){
  a = NULL;
}

int main(){
  int a[67];
  a[0] = 123;
  printf("%d\n", a[0]);
  nullify(a);
  printf("%d\n", a[0]);
  return 0;
}
```

Resultado:

i. 123
   Segmentation fault

ii. 123
    123

iii. 123
     basura

iv. Segmentation fault

v. Ninguna de las anteriores

b)

```
#include<stdio.h>

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

Resultado:

i. hola mundo

ii. sola mundo

iii. Segmentation fault

iv. Ninguna de las anteriores

Estructuras y punteros

12. Defina una estructura para representar puntos en el plano, y una función **medio** que dados dos de estos puntos, calcule el punto medio.

13. Suponga que tiene que implementar un juego que requiere utilizar un mazo de cartas. Implemente los siguientes puntos para crear una representación del mismo que luego pueda utilizarse para el juego:

a) Defina un tipo de datos **Carta** para representar una carta de la baraja española. Represente el palo con una enumeración.

b) Defina un tipo de datos **Mazo** como un arreglo de 48 cartas.

c) Implemente una función **void llenar(Mazo)** que tome un mazo y lo llene con las cartas correspondientes.

d) Implemente una función **Carta azar(Mazo)** que reciba un mazo y devuelva una carta al azar del mismo.

14. Cree una representación para una agenda de contactos:

a) Defina un tipo de datos **Contacto** que permita almacenar: una cadena para el nombre de una persona, una cadena para el número de teléfono, y un entero sin signo para llevar la edad de la persona.

b) Implemente una función **Contacto crear_contacto()** que pida por teclado los datos pertinentes, rellene un nuevo contacto con esos datos, y lo devuelva. Puede usar la función **get_new_line** del ejercicio 6.

c) Implemente una función **void actualizar_edad(Contacto *)** que dado un puntero a un contacto, pida una nueva edad por teclado, y la actualice. ¿Es necesario que su argumento sea un puntero?

d) Defina un tipo de datos **Agenda** que almacene un arreglo de contactos y un entero para llevar la cantidad de contactos ingresados en la misma.

e) Implemente una función **void alta_contacto(Agenda *)** que pida por teclado los datos de un contacto a agregar (función **crear_contacto**) e inserte el resultado en la agenda.

f) Implemente una función **void modificar_edad(Agenda *)** que pida por teclado el nombre del contacto a modificar, lo busque en la agenda, y luego actualice su edad (función **actualizar_edad**).

g) Implemente una función **void imprimir_contactos(Agenda *)** que muestre por pantalla los datos de los contactos cargados.

h) Implemente una función **double prom(Agenda *)** que devuelva el promedio de la edad de los contactos dentro de la agenda.

15. Considere el código y luego explique si las expresiones, que se encuentran a continuación, son correctas; en caso de serlo, dé el valor de las mismas.

```
#include<stdio.h>

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 0;
}
```

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		