Programación y Algoritmos I
Tarea 3

Problema 1 [2.5 punto]

Consideramos la estructura siguiente:

```
struct myStruct {
        char a;
        char *b;
        char c;
};
```

1. Escribir un programa que permita desplegar el tamaño de esta estructura.
2. Dar una explicación del resultado obtenido.
3. Moviendo el orden de los elementos de esta estructura, qué sería el tamaño mínimo al cual se podría llegar? Explicar.

Problema 2 [1.5 punto]

```
float a, b, *ptr_a, *ptr_b, **dptr_a, **dptr_b;
a = -1.0;
b =  5.0;
ptr_a  = &a; // Explicar a partir de aqui
dptr_a = &ptr_b;
dptr_b = dptr_a;
*dptr_b= ptr_a;
(**dptr_a) = -3.0;
```

Explicar paso a paso qué está pasando en el programa arriba a partir de la cuarta línea. Representar después de cada línea la situación de cada apuntador: describir a qué variable apunta, y qué valor tiene la variable apuntada.

Problema 3 [1 punto]

En la clase (sesión 1), se ha usado el siguiente código para visualizar la representación binaria de un número flotante z:

```
unsigned int iz = *(unsigned int*)&z;
fprintf(stderr,"Diff inv : %\n", iz);
```

1. Explicar qué hace exactamente el código.
2. Funcionaría el mismo código para desplegar un double? Si no, proponer una manera de visualizar la representación binaria de un double.

Problema 4 [2 punto]

Escribir una función que tome de entrada un entero positivo n y que use alocación dinámica para crear y rellenar un triángulo de Pascal (es decir, una estructura con los coeficientes binomiales $C[i][j]$ para $0 \leq i \leq j \leq n$). Se usará la menor cantidad de memoria posible.

Problema 5 [3 punto]

Una estructura de datos muy útil es la lista ligada

```
typedef struct structList {
        int value;
        struct structList *next;
} list;
```

Esta estructura está compuesta de elementos encadenados. Cada elemento es una de las estructuras descritas arriba, con un valor guardado en ella. Además del valor, el elemento de la lista contiene un apuntador al siguiente elemento. Se usa el apuntador NULL para marcar la terminación de la lista.

1. Escribir una función que agrega un nuevo elemento a la izquierda de una lista existente. Se usará alocación dinámica.

   ```
   list* addElement(int element, list* prev);
   ```

2. Escribir una función para desplegar todos los elementos de una lista ligada.

   ```
   void print(list* l);
   ```

3. Escribir una función para liberar la memoria de todos los elementos contenidos en una lista ligada.

   ```
   void liberate(list* l);
   ```

4. Escribir una función para recuperar el valor del k-ésimo elemento de una lista ligada.

   ```
   int getKthValue(int k, list* l);
   ```

5. Escribir una función que determine si los contenidos de dos listas ligadas son idénticos o no.

   ```
   int isEqual(list* la, list* lb);
   ```

6. Escribir una función main que use **todas** las funcionalidades descritas arribas para generar dos listas ligadas de al menos 100 elementos. Liberar la memoria alocada.