BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA
FACULTAD DE CIENCIAS DE LA COMPUTACIÓN
EJERCICIOS CON MANEJO DE MEMORIA DINÁMICA

1. Implementa una función en C, denominada concatenación, que, dados dos vectores de enteros y dos enteros que indican sus dimensiones, devuelva un vector dinámico formado por la concatenación de los dos vectores.
Implementa una pequeña función main que utilice la función anterior y escriba el vector resultante por pantalla. Recuerda que la función main, antes de terminar, tendría que liberar la memoria dinámica que se haya reservado dentro de la función concatenación.
Nota: Ten en cuenta que la función concatenación necesitarla un parámetro adicional para poder devolver la longitud del nuevo vector, por lo que tendría un total de 5 parámetros.

2. Modifica tu solución del ejercicio anterior de forma que consigas evitar el uso de dos variables independientes para almacenar en una el vector dinámico y en otra su talla. Para ello debes definir y utilizar un registro que agrupe ambos datos.

3. Un polígono puede representarse mediante la secuencia ordenada de sus vértices, que a su vez pueden indicarse mediante puntos del plano, es decir, valores (x, y). Diseña la estructura de datos tipoPunto que permita almacenar las componentes (x, y) de un punto en el plano.
Como cada polígono puede tener un número diferente de vértices, diseña la estructura de datos tipoPolígono de forma que permita almacenar un vector dinámico de puntos y su talla (el numero de vértices del polígono).
Escribe las siguientes funciones:
leePoligono Esta función debe crear y leer de teclado un polígono. Se pedirá al usuario que introduzca el numero de vértices del polígono y luego las componentes x e y de cada uno de ellos. La función debe devolver el polígono leído.
imprimePoligono Esta función debe admitir como argumento un polígono e imprimir en la pantalla sus vértices.
perimetroPoligono Esta función debe admitir como argumento un polígono, calcular su perímetro y devolver dicho valor.
Para comprobar el funcionamiento de tus funciones, utiliza la siguiente función main:

```
  int main(void)
{
  tipoPoligono miPoligono;
  miPoligono = leePoligono();
  printf("V´ertices del pol´ıgono:\n");
  imprimePoligono( miPoligono );
  printf("Per´ımetro del pol´ıgono: %f\n",
  perimetroPoligono(miPoligono));
  return 0;


}
```

Ejemplo: el polígono de 3 vértices formado por los puntos (1, 1), (3, 1) y (2, 3) tiene un perímetro de 6,472136
4. Implementa una función en C que, dado un numero entero n, devuelva una matriz dinámica de n × n elementos, donde en cada posición [i][j] de la matriz se almacene lo siguiente:

Si i _ j se almacena el valor i + j + 1.
Si i < j se almacena el valor cero.

Por ejemplo, para n = 4 tendríamos la siguiente matriz:

1 0 0 0
2 3 0 0
3 4 5 0
4 5 6 7

Para ahorrar espacio de almacenamiento, no queremos almacenar los elementos cuyo valor es cero, es decir, sólo almacenaremos los elementos del triangulo inferior de la matriz.
Implementa una pequeña función main que utilice la función anterior y escriba la matriz obtenida por pantalla. Recuerda que la función main, antes de terminar, tendría que liberar la memoria dinámica reservada al construir la matriz.

5. Un programa en C necesita manipular un conjunto de palabras. Se nos encarga la tarea de implementar esta funcionalidad y para ello se nos ofrece la siguiente descripción de requisitos:

- La aplicación puede manipular varios conjuntos separados de palabras.
- El orden y la forma en que se almacenan las palabras en la estructura de datos no es relevante para el resto de la aplicación.

- Las operaciones que se deben soportar son:

    1. Obtener un conjunto vacío.

    2. Dada una palabra, obtener un conjunto con esa única palabra.

    3. Dado un conjunto y una palabra, obtener el nuevo conjunto que contenga la palabra dada.

    4. Dado un conjunto y una palabra, devolver un conjunto que no contenga la palabra dada. Si la palabra no existe, el conjunto no se modifica

    5. Destruir un conjunto de palabras dado.

- Todas las operaciones que reciben una palabra y la almacenan, deben crear un duplicado con la llamada al sistema strdup. Esta función recibe una cadena de tipo char * terminada por un byte igual a cero y devuelve una cadena (de tipo

char *) que es un duplicado del parámetro en una porción de memoria reservada con malloc.

- La solución propuesta debe ser compacta, es decir, tener un uso de memoria lo más reducido posible.

6. En base a la siguiente información responde las siguientes preguntas:

| 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 |

| Línea | Código |
|---|---|

```
 1  #include <stdlib.h>
 2  #include <string.h>
 3
 4  struct bluetoothinfo {
 5      char *name;
 6      unsigned int strength;
 7      char mac[6];
 8  };
 9
10  struct bluetoothinfo binfo, *newInfo;
11
12  struct bluetoothinfo *duplicate(struct bluetoothinfo *);
13
14  int main(int argc, char **argv) {
15      struct bluetoothinfo *bPtr;
16      int i;
17
18      bPtr = &binfo;
19      bPtr->name = "My phone";
20      bPtr->strength = 100;
21      for (i = 0; i < 6; i++) {
22          bPtr->mac[i] = 10;
23      }
24      newInfo = duplicate(bPtr);
25      free(newInfo);
26
27      return 0;
28  }
29
30  struct bluetoothinfo *duplicate(struct bluetoothinfo *srcPtr) {
31      struct bluetoothinfo *result;
```

| Línea | Código |
|---|---|
| 32<br>33<br>34<br>35<br>36<br>37<br>38<br>39<br>40<br>41<br>42 | `    int i;`<br><br>`    result = (struct bluetoothinfo *)malloc(sizeof(struct bluetoothinfo));`<br>`    result->name = (char *)malloc(strlen(srcPtr->name) + 1);`<br>`    strcpy(result->name, srcPtr->name);`<br>`    result->strength = srcPtr->strength;`<br>`    for (i = 0; i < 6; i++) {`<br>`        result->mac[i] = srcPtr->mac[i];`<br>`    }`<br>`    return result;`<br>`}` |

Se recomienda que descargues este programa en tu área de trabajo para poder hacer cambios puntuales, compilarlo y ejecutarlo. Algunos de los siguientes problemas se pueden resolver de esta forma.

1. ¿Qué tamaño tiene la estructura que se define en las líneas 4 a 8?
2. ¿Por qué crees que se pone la línea 12? Prueba a comentarla y compila el programa.

3. En la línea 19 se realiza una asignación de una cadena de texto, ¿en qué tipo de memoria (global, heap, pila) crees que está esa cadena?

4. Reescribe las líneas 19 y 20 pero en lugar de utilizar la variable `bPtr` utiliza directamente `binfo`. ¿Puedes escribir la función `main` para que haga exactamente lo mismo pero suprimiendo la variable `bPtr`?

5. Busca qué hace exactamente la función `strlen` que se utiliza en la línea 35. ¿Qué resultado devuelve para el caso del programa? ¿Cuánta memoria se está reservando en esa invocación a `malloc`?

6. Busca qué hace la función `strcpy` de la línea 36 y explica qué memoria se está modificando y dónde está para caso del programa de ejemplo.

7. ¿Cómo accederías a la tercera letra de la cadena del campo `name` de la estructura a la que apunta `result` en la línea 37? (Puedes responder a esta pregunta insertando una línea que imprima la respuesta y ejecutando el programa)

8. ¿Dónde se copia el valor del puntero `result` al ejecutar la línea 41?

9. ¿Cuántos bytes de memoria dinámica se utilizan en el programa de ejemplo?

10. La función `duplicate` definida en las líneas 30 a 42, como su nombre indica, crea una estructura que es un duplicado de la que apunta el parámetro dado. En la línea 24 del `main` se obtiene ese duplicado y se almacena en `newInfo`. ¿Cuál sería el efecto de, en lugar de llamar a `duplicate` simplemente hacer `newInfo = bPtr`?

11. La duplicación del campo `name` de la estructura a la que apunta el parámetro se realiza en las líneas 35 y 36. ¿Por qué no basta con poner simplemente `result->name = srcPtr->name`?

12. ¿Cómo reservarías espacio para una tabla de 200 elementos de la estructura `struct bluetoothinfo` y a la vez inicializar su contenido todo al valor cero? Asigna la dirección de ese espacio al puntero `ptr`. Escribe la porción de código equivalente pero utilizando `malloc`.

13. En un programa en C hay que reservar espacio en memoria dinámica para una tabla de 100 estructuras de datos previamente definidas, pero no es preciso inicializarlas a ningún valor en particular. ¿Cuál de las dos funciones `malloc` y `calloc` utilizarías? ¿Por qué?

14. Un programa define la siguiente estructura de datos en la que almacena una tabla de enteros y una tabla de letras pero de tamaño desconocido:

```
struct twotables {
    int *inttable;
    char *chartable;
}
```

Escribe una función que recibe dos enteros, reserva una estructura de este tipo y con tablas de tamaños igual a los valores de los enteros y devuelve el puntero a esta estructura. No es preciso inicializar ningún dato. Pista: se requiere más de una llamada a `malloc`.

Escribe la función contraria, es decir, dado un puntero a una estructura de este tipo, libera la memoria que ocupa la estructura y las tablas.

¿Cómo reservarías espacio en memoria para una tabla de 100 estructuras de tipo `struct twotables`? ¿Cuánto espacio ocupa en memoria?

15. Una aplicación gráfica mantiene una tabla con un conjunto de puntos que representa como números enteros. El número de puntos fluctua en un rango entre cero y un millón de puntos. Para no tener reservada memoria para un millón de puntos, la aplicación comienza por reservar espacio sólo para 1000 puntos con la línea:

```
points = (int *)malloc(1000 * sizeof(int));
```

La aplicación lleva la cuenta de los puntos que tiene almacenados, y al cabo de un rato ejecutando necesita almacenar más de 1000 puntos. ¿Qué operación sobre memoria dinámica necesitas ejecutar? ¿Qué parámetros utilizarías? Justifica tu respuesta.

16. Una aplicación almacena la información sobre contactos en una tabla con 100 estructuras como la siguiente:

```
struct contact {
    char *name;
    char *lastname;
};
```

La variable `tabla` almacena estos elementos y su espacio ha sido reservado de forma dinámica (con `malloc` o `calloc`). De forma análoga, para cada elemento, el espacio al que apuntan los campos `name` y `lastname` también ha sido reservado de forma dinámica. Escribe la función que recibe como parámetro un puntero a una tabla de este tipo y un entero con su número de elementos, y que libera toda la memoria ocupada por la tabla.

```
void nuke(struct contact *table, int size) {
...
...
}
```

17. Supóngase la siguiente definición de estructura de datos, variables y código:

```
struct unit {
    struct unit *next;
} *unit1, *unit2, *unit3;

unit1 = (struct unit *)malloc(sizeof(struct unit));
unit2 = (struct unit *)malloc(sizeof(struct unit));
unit3 = (struct unit *)malloc(sizeof(struct unit));
unit1->next = unit2;
```

```
unit2->next = unit3;
unit3->next = NULL;
```

Escribe la función `void borra(struct unit *ptr)` tal que si se invoca como `borra(unit1)` libera la memoria reservada de las tres estructuras. ¿Tienes que cambiar algo en tu función para que haga la misma operación para una longitud arbitraria de la cadena de estructuras?

18. Supongamos que a lo largo de la ejecución de un programa en C, cada vez que se invoca a `malloc` o `calloc` se incrementase una variable global entera a modo de contador inicializada a cero, y cada vez que se invocase `free` se decrementase.
    1. ¿qué valor debería tener justo antes de terminar la ejecución?

    2. Y si un programa termina su ejecución y esta variable vale cero, ¿es esto suficiente para concluir que hace una gestión de memoria correcta?

19. Un programa en C manipula un número muy elevado de elementos de las dos siguientes estructuras de datos:

```
struct picture_info {
    char *name;
    char *location;
    char *note;
    int size;
    int latitude;
    int longitude;
};
struct video_info {
    char *location;
    int *encoding
    int length;
};
```

Todos los elementos se crean utilizando memoria dinámica y el código de la aplicación está perfectamente dividido en dos partes, cada una para manipular los elementos de una de las estructuras.

Al terminar la ejecución, el programa tiene porciones de memoria que se han reservado pero no liberado. Se utiliza una herramienta que supervisa esta ejecución y al final emite un informe en el que consta que 2345 porciones de memoria, todas ellas de 12 bytes, han sido reservadas pero no liberadas. ¿En qué parte del código empezarías a buscar la anomalía en la gestión de memoria y por qué?