Práctica No. 6

Apuntadores y memoria dinámica.

Objetivo: Desarrollar aplicaciones de software utilizando estructuras de datos para modelar y manipular la información requerida para la solución de problemas, de forma analítica, propositiva y organizada..

Material:
- Computadora Personal (PC)
- Programa Editor de texto (ASCII), compilador GCC

Equipo:
- Computadora Personal

Introducción

Apuntadores

Un apuntador es una variable que contiene la dirección en memoria de otra variable. Se pueden tener apuntadores a cualquier tipo de variable.

El operador unario o monádico `&` devuelve la dirección de memoria de una variable.

El operador de indirección o dereferencia `*` devuelve el "contenido de un objeto apuntado por un apuntador".

Para declarar un apuntador para una variable entera hacer:

```
int *apuntador;
```

Se debe asociar a cada apuntador un tipo particular. Por ejemplo, no se puede asignar la dirección de un short int a un long int.

Un apuntador a cualquier tipo de variables es una dirección en memoria -- la cual es una dirección entera, pero un apuntador no es un entero.

La razón por la cual se asocia un apuntador a un tipo de dato, es porque se debe conocer en cuántos bytes está guardado el dato. De tal forma, que cuando se incrementa un apuntador, se incrementa el apuntador por un "bloque" de memoria, en donde el bloque está en función del tamaño del dato.

Por lo tanto para un apuntador a un char, se agrega un byte a la dirección y para un apuntador a entero o a flotante se agregan 4 bytes. De esta forma si a un apuntador a flotante se le suman 2, el apuntador entonces se mueve dos posiciones float que equivalen a 8 bytes.

Memoria dinámica

malloc

Reservación de memoria en tiempo de ejecución con `malloc`. La función malloc se utiliza para reservar un bloque de memoria de tamaño dado por un parámetro (size), si la operación fue exitosa regresa un apuntador al bloque de dirección reservado, por el contrario, si la operación no fue exitosa regresa un apuntador con la dirección NULL (\x0).

```
void *malloc(size_t size)
```

Se debe considerar el tamaño del tipo de dato que se quiere reservar, debido a que los tipos de dato tienen diferente tamaño en bytes, por ejemplo, el bloque de memoria requerido para almacenar 5 valores del tipo char es menor al requerido para almacenar 5 valores del tipo int. Por lo tanto, es recomendable hacer uso de la función `sizeof` para conocer el tamaño en bytes de un determinado tipo de dato.

Por ejemplo si se requiere reservar memoria para 3 elementos del tipo entero de forma dinámica se tendría que realizar lo siguiente:

```
int *ptrArray  = (int*) malloc(sizeof(int)*3);
```

Es importante destacar que dado que malloc regresa un apuntador del tipo void, por lo tanto, se podría realizar una conversión explícita de tipo, en este caso `(int *)`.

realloc

En ocasiones es necesario cambiar el tamaño del bloque de memoria reservado con malloc, ya sea para liberar cierto espacio que no se va a requerir, o en caso contrario, expandir el bloque de memoria para almacenar más información. Para este cambio de tamaño en la memoria reservada, se encuentra la función realloc el cual recibe un apuntador al bloque de memoria reservado y el tamaño que se desea adquiera ese bloque de memoria.
Es sumamente importante recalcar que el tamaño indicado (size) es el tamaño en bytes que tendrá el bloque, si se envía un tamaño menor del actual, se libera el espacio restante de memoria. Si es enviado el valor 0 en el parámetro de tamaño en la función realloc, se libera el espacio de memoria reservado y se regresa un apuntador apuntando a la dirección NULL (`\x0`).

```
void *realloc(void *ptr, size_t size)
```

Por ejemplo si se requiere extender el bloque del ejemplo anterior, donde se reservó memoria para almacenar 3 números enteros, ahora para que almacene 3 bloques más de memoria se requeriría realizar lo siguiente:

```
ptrArray  = (int*) realloc(ptrArray, sizeof(int)*3*2);
```

Nótese que la parte subrayado en azul es el tamaño actual y se multiplica por dos para que pueda almacenar otros 3 elementos del tipo entero, en este caso el bloque de memoria reservado se expande.

free

Siempre que se hace uso de memoria dinámica en C, se debe indicar en algún punto del programa que se dejará de usar un bloque de memoria que fue reservado ya sea con malloc, calloc o realloc. La operación de liberar memoria que fue reservada se realiza a través de la función `free`, la cual recibe como parámetro un apuntador que previamente le fue asignado un bloque de memoria reservada.

```
void free(void *ptr)
```

Siguiendo los ejemplos anteriores, una vez que ya no requiramos hacer uso de ese bloque de memoria, se debe realizar la liberación de memoria de la siguiente forma:

```
free(ptrArray);
```

Desarrollo de la práctica

Implementar los siguientes programas en archivos independientes .c.

1. Implementar una función `swap` que intercambie los valores de dos variables enteras. Asignar valores por defecto a ambas variables e imprimir sus valores, seguido de ejecutar la función swap, volver a imprimir sus valores, los cuales deberían estar invertidos.

2. Implementar una función swap para intercambiar direcciones de memoria correspondientes a cadenas de caracteres, las cuales deben guardar respectivamente lo siguiente: “Esta es la primer cadena” y “Esta es la segunda cadena”. Al iniciar el programa se deberá imprimir el contenido de ambas cadenas, seguido de ejecutar la función `swap_strings`, se deberán imprimir los valores de ambas cadenas, cuyos valores deberán estar intercambiados.

3. Crear una función que pida al usuario capturar valores y guardarlos en un apuntador, al inicio del programa NO se le pide al usuario indicar el número de

elementos que contendrá el arreglo, por lo tanto se requerirá hacer uso de memoria dinámica. El programa en cada iteración preguntará al usuario si desea agregar otro elemento al conjunto de enteros, de ser positiva la respuesta se debe incrementar la capacidad del bloque de memoria, caso contrario si el usuario indica que no, la función debe regresar el apuntador al bloque reservado.
Una vez terminada la función se debe imprimir el número de elementos contenidos en el arreglo e imprimir el valor mayor, menor y la suma de todos sus elementos sin usar corchetes "[]" para acceder a sus elementos.

4. Crear una función que reciba dos parámetros indicando las dimensiones de una matriz de números flotantes y debe regresar un apuntador al bloque de memoria reservado.
   Hacer un programa para probar la función, generando una matriz con valores aleatorios de -10 a 10 del tamaño indicado por el usuario, ya sea por parámetros de consola o por entrada de datos a través de un formulario.

Restricciones

1. El código fuente deberá contener comentarios que indique el objetivo del programa y descripción de instrucciones más relevantes.
2. Evitar nombres de variables y funciones que no reflejen su propósito claramente.
3. Cuando se haga uso de memoria dinámica es importante no olvidar liberar la memoria.

Conclusiones y Comentarios.