Ejercitación – Serie K

Memoria dinámica

Conceptos que se agregan en esta serie: Manejo de memoria en forma dinámica, uso del Heap y consideraciones generales respecto al manejo de recursos (liberarlos cuando ya no se requieren).

Como ya se sabe, está prohibido el uso de variables globales.

Ejercicio 1:

Realizar una aplicación para gestionar el nombre de personas junto a su número de legajo.

Cada persona tendrá su nombre y apellido, junto a un legajo.

Tanto el nombre como el apellido pueden contener varias palabras separada por espacios.

La aplicación debe permitir ingresar datos, consultar y eliminar bajo las siguientes condiciones. Estas acciones se pueden repetir varias veces durante la ejecución de la aplicación.

Detalles a considerar:

- Se desconoce la cantidad máxima de datos a ingresar.
- Cada vez que se inicia un proceso de ingreso (o reingreso), la aplicación debe consultar al usuario, cuantos datos quiere ingresar (***uso de realloc – ver el man***).
- Más allá de contar con una cantidad máxima definida, se debe poder interrumpir el ingreso antes de alcanzar dicha cantidad.
- No debe haber legajos repetidos. En caso de querer ingresar un legajo que ya se encuentre ingresado, se debe negar su ingreso e informar al usuario la situación.
- En caso de ingresar un usuario con el mismo nombre y apellido, alertar al usuario y consultarle si quiere continuar con el ingreso (independiente de si se ingresó con minúsculas o mayúsculas).
- El largo máximo de un nombre y de un apellido nunca serán mayor a 80 caracteres (cada uno de ellos). Solo debe aceptar caracteres alfabéticos, espacios y apóstrofes entre otros. De existir más de un espacio de separación entre las palabras que conforman el nombre o el apellido, el excedente se debe eliminar.
- La memoria utilizada para almacenar estos datos debe ajustarse a la cantidad necesaria.
- Para eliminar un usuario, se debe ingresar su legajo.
- En caso que luego de un proceso de ingreso o eliminación, la capacidad de los bloques de memoria posea espacio para más de 20 datos disponibles, el sistema debe reajustar los tamaños de los bloques (***uso de realloc – ver el man***).
- En caso de error al solicitar memoria, debe informarle al usuario esta condición y continuar con la aplicación normalmente.

Ejercicio 2:

Modificar el punto anterior, de manera que cuando se tenga que hacer un ingreso (o reingreso), el usuario no tenga que ingresar la cantidad de datos que desea entrar.

Ejercicio 3:

Modificar el punto anterior de manera que cada vez que se tenga que solicitar memoria, esta petición se realice optimizando los tiempos de procesamiento del S.O., esto es, no pedir un solo bloque, sino que en su lugar pedir memoria para poder ingresar varios datos. Este valor debe estar declarado en la etiqueta QTTY_MEMO_ALLOC.

Ejercicio 4:

Modificar a la aplicación realizada de manera tal que tanto el nombre como el apellido se ingresen en forma directa en la memoria dinámica, considerando utilizar solo la capacidad necesaria de memoria.

Ejercicio 5:

Modificar el ejercicio anterior, para permitir eliminar un registro por nombre, por apellido o por nombre y apellido. En caso de existir más de uno, se debe dar la opción de eliminar uno en particular (por ejemplo, a través del legajo) o todos.

Ejercicio 6: **Liberar recursos**

Agregar al anterior la opción de borrar todos los elementos.

Ejercicio 7: **Prueba operativa**

7.1.-

Modificar la aplicación obtenida en el punto anterior de manera de contar con un set de datos que permita evaluar el comportamiento de la aplicación sin ingresar datos por teclado.

El set de datos, debería probar/testear todas las opciones de la aplicación.

Se debe considerar que, en el paso previo a la finalización de la aplicación, no queden recursos tomados.

7.2.-

Meter el código de la aplicación generada dentro de un while (1), de manera que su ejecución sea perpetua.

Una forma simple de hacer esto, es modificar el nombre de la función main por main_2, y generar un nuevo main que llame al anterior.

Por ejemplo:

```
int main ()
{
     while (1)
        main_2();
return 0;
}
```

o

```
int main (int argc, char **argv, char ** env)
{
     while (1)
        main_2(argc, argv, env);
return 0;
}
```

Puede verificar el estado de la memoria desde otro terminal, por ejemplo, con los comados top o free entre otros. Si bien la ocupación del procesador podría subir drásticamente, la ocupación de memoria debería permanecer dentro de un rango estable.

7.3.-

Introducir en el código a través de compilación condicional, condiciones controladas de fallo para verificar el comportamiento de la aplicación ante estas condiciones.

Como mínimo considerar:

- Error al pedir memoria para nuevos registros
- Error al pedir memoria para el nombre y/o apellido.