Universidad de Costa Rica
CI-0113: Programación 2
Laboratorio #3

Objetivo: Familiarizar al estudiante con el concepto y uso de punteros en C++.

Enunciado

Resuelva los siguientes problemas. Escriba un código en C/C++ que resuelva los siguientes problemas:

1. Implemente una función *swap* en C/C++ que reciba dos parámetros "enteros" y realice el cambio de los datos contenidos en ellas.
    a. Los datos almacenados en las variables cambian.
    b. E.g: con a = 3, b = 7 y swap(a,b) se obtendría el valor 7 en la variable a y 3 en la variable b.

2. Las hileras o strings no existen como tipo de dato básico en C/C++, por lo que para utilizarlas lo que se utiliza realmente es un arreglo de caracteres (char*) de tamaño fijo. Es deber del programador asegurarse que al utilizar una "string" los datos no se excedan del tamaño de su arreglo. Al mismo tiempo, dado que el texto almacenado puede ser de tamaño variable (aunque el arreglo sea de tamaño fijo), para identificar el final de una hilera se utiliza el carácter especial '\0' cuya representación numérica es el valor 0. De esta manera la hilera "hola mundo" realmente está compuesta por 11 caracteres, 10 normales y un '\0' adjunto al final.
    a. Implemente una función *length* que reciba por parámetro una "hilera" constante de caracteres (const char*) y calcule y retorne el tamaño de la hilera. Al calcular el tamaño no debe contar el caracter nulo de finalización.
        i. length("hola mundo") ---> retorna 10

    b. Implemente una función *copy* que reciba por parámetros una hilera constante "origen", una hilera "destino" y el tamaño "n" máximo del arreglo. Su función debe copiar los datos de la hilera origen a la hilera destino hasta un máximo de n caracteres y retornar el tamaño de la hilera resultante. Recuerde copiar el caracter '\0' al final de la hilera.
        i. copy("progra2",arreglo,256) ---> retorna 7 y copia el texto a arreglo

    c. Implemente una función *compare* que reciba por parámetros dos hileras constantes de caracteres. Su función debe retornar 0 si ambas hileras son iguales. Un número negativo si la primera hilera es "menor" a la segunda (usando ordenamiento alfabético estándar como el visto para Java) y un número positivo si ocurre lo contrario.
        i. compare("amo","amor") ---> retorna <0
        ii. compare("hola","alo") ---> retorna >0
        iii. compare("curioso","curioso") ---> retorna 0

d. (Opcional: puntos de logro +5) Implemente una función *startsWith* que reciba por parámetros dos hileras constantes de caracteres. La función debe retornar true si la primera hilera empieza con la segunda hilera.
   i. startsWith(“habia una vez”,”ha”) ---> retorna true
   ii. startsWith(“hola mundo”,”ola”) ---> retorna false

e. (Opcional: puntos de logro +5) Implemente una función *count* que reciba por parámetros dos hileras constantes de caracteres. La función debe retornar la cantidad de veces que aparece la segunda hilera en la primera hilera.
   i. count(“tres tristes tigres”,”es”) ---> retorna 3
   ii. count(“treees tristees tigrees”,”ee”) ---> retorna 4

f. (Opcionales/práctica) Si quieren más ejercicios opcionales relacionados a hileras aquí hay una lista de funciones adicionales a considerar: *append/concat, endsWith, findFirst, findLast, removeChar, removeString, replaceChar, replaceString, substring, reverse, skipLeading, toInteger, toDouble, split.*

3. (Opcional: puntos de logro +10) En los tiempos de C no existían aún los comandos new/new[]/delete/delete[], sino que se utilizaban las funciones malloc (memory-allocation) y free pertenecientes a la biblioteca “stdlib.h”. La diferencia esencial en comparación al new es que el comando malloc no recibe ningún tipo de dato ni índice, sino que es una función cuyo único parámetro es la cantidad de bytes de memoria a alocar. Además, la función retorna la dirección de memoria alocada como un tipo de dato (void*/dirección de memoria pura) por lo que era deber del programador convertirla al tipo de dato que se deseaba utilizar:

```
int* arreglo = (int*)(malloc(256))
```

Toda alocación de memoria generada con el comando malloc luego debía ser liberada utilizando el comando free:

```
free(arreglo)
```

Usualmente se mezclaba su uso con el comando sizeof(type) que permite averiguar el tamaño en bytes de un tipo específico de la siguiente manera:

```
int* arreglo = (int*)(malloc(64*sizeof(int)))
```

creando así arreglos de “n” elementos de “t” tipo y haciendo el código más legible.

   a. Cree una función que reciba por parámetros un alto y un largo, su función debe generar el arreglo de bytes de tres dimensiones (unsigned char***) necesario para almacenar una imagen RGB de alto x largo x 3.
   b. Para ello **solo** puede utilizar **UN** llamado al comando malloc. No puede utilizar ningún comando new, ni ningún otro tipo de alocación de memoria.
   c. El resto de la organización de la memoria alocada y su contenido recae en sus manos.
   d. Debe retornar una variable de tipo unsigned char*** que sea un arreglo de 3 dimensiones funcional.
   e. En el main recuerde liberar la memoria después de utilizarla
   f. Puede utilizar su código del laboratorio 2 para probar el funcionamiento adecuado del arreglo creado.