Guía Práctica 8: Cadenas de caracteres

Observación:
En aquellos ejercicios en los que no se indica el tamaño máximo de la cadena de caracteres, el alumno deberá definirlo en forma coherente de acuerdo a lo que se pretende representar con la cadena en cuestión.

Ejercicios propuestos:

Problema 1: Diseñar un programa que permita generar direcciones de correo electrónico. El programa recibe el apellido y nombre de un usuario de la facultad (apellido y nombre se asignan a una sola variable) y debe retornar la dirección de correo electrónico (e-mail) generada. El dominio asignado a la Facultad para el e-mail es: frsf.utn.edu.ar, y el nombre de usuario se forma con la inicial del nombre y a continuación el apellido.

```
Ejemplo:
Ingrese apellido y luego el nombre del profesor: Perez Juan
Email: jperez@frsf.utn.edu.ar
```

Problema 2: Diseñar una función que permita registrar un nuevo usuario al sistema.
Los identificadores deben tener 8 caracteres, comenzar con una letra y los restantes pueden ser letras o números. La función debe recibir una cadena y comprobar que la misma sea un identificador de usuario válido.

```
Ejemplos:
registrar( "jperez12") → true
registrar( "1jperez123") → false
```

Problema 3: Realizar un programa que pida una frase y nos indique si es o no palíndroma (se lee igual de izquierda a derecha que de derecha a izquierda).

```
Ejemplo:
Entrada  "dabale arroz a la zorra el abad" → True
```

Problema 4: Dadas 2 cadenas, a y b, retornar otra cadena con la forma *corto+largo+corto*, de manera que la cadena más corta esté en los extremos y la más larga en el medio. Las cadenas no serán de la misma longitud pero sí pueden estar vacías (longitud 0).

```
Ejemplos:
comboString("Hello", "hi") → "hiHellohi"
comboString("hi", "Hello") → "hiHellohi"
comboString("aaa", "b") → "baaab"
```

Problema 5: Escribir un programa que dada una cadena de caracteres indique el número total de caracteres y de palabras que la conforman. Tener en cuenta que los espacios también son caracteres. Se asume que la oración está bien conformada, teniendo un único espacio entre palabras, los signos de puntuación van inmediatamente después de la última letra de cada palabra, etc.

```
Ejemplo:
"El perro de San Roque no tiene rabo, Porque Ramón Ramírez se lo ha cortado."
Palabras: 15
Caracteres: 75
```

Problema 6: Realizar un programa que busque todas las ocurrencias de la palabra prohibida, que introduzca el usuario, en una frase leída y las reemplace por la cadena "XXX" (donde el número de "X" que contiene la cadena deberá ser igual al número de caracteres que tiene la palabra prohibida).

Ejemplo:
Introduce una frase: "Esto es una ese escondida y la he escrito yo"
Introduce la palabra prohibida: "es"
La frase resultante es: "Esto XX una XXe XXcondida y la he XXcrito yo

Problema 7: Se dice que la palabra α es un anagrama de la palabra β si es posible obtener α cambiando el orden de las letras de β. Por ejemplo, "MORA" y " ROMA" son anagramas de RAMO.
Realizar una función que compruebe si dos palabras son anagramas entre sí. Las palabras pueden contener letras mayúsculas y minúsculas.

Problema 8: Escribir un programa que limpie de ruidos una señal de entrada. La señal de entrada será una cadena con letras y números y la salida será la misma cadena eliminando los números.
Ejemplo:
Entrada: *"Es2to0 3es u9na se88ñal c0on ru1id2os"*
Salida: *"Esto es una señal con ruidos".*

Problema 9: Escribir un programa que convierta una fecha en formato "MMDDYYYY" al formato "DD de mes del YYYY".
Por ejemplo, para "12072006" debería devolver "7 de diciembre del 2006".

Problema 10: Dado un texto que contiene un telegrama que termina en punto, desarrollar una función que:

- informe la cantidad de veces que aparece cada vocal.
- informe el porcentaje de espacios en blanco.
- cuente y muestre la cantidad de palabras que posean más de 7 letras.
- informe el costo del telegrama sabiendo que cada palabra cuesta $5,2.

Ejemplo:
Leyendo el telegrama ...
En breve evaluación de Algoritmos a estudiar.
El telegrama ingresado consta de:

- ➔ 3 palabras con más de 7 letras
- ➔ Ocurrencias de vocales: 19
  - a: 5
  - e: 6
  - i: 3
  - o: 3
  - u: 2
- ➔ 13.33% de espacios en blanco sobre el total de caracteres.
- ➔ Costo del telegrama: $36,40.

Nota: Considerar que las palabras están separadas por un único espacio blanco. Los números se consideran como palabra.

Problema 11: Un texto se ha guardado en una cadena de caracteres, donde cada componente tiene un carácter. Hay frases que terminan con punto y párrafos que terminan con ´#´.

```
Ejemplo:
El cielo está oscuro.Llega la noche.#Serenidad absoluta.Silencio profundo.#estoy aquí.
```

a) Se desea formatear el texto, modificando la lista de manera que luego de cada salto de línea se agreguen 3 "blancos".
b) También se debe controlar que después de un punto el primer carácter sea mayúscula.
c) Se debe informar cuantas frases hay en todo el texto.

Problema 12: El cifrado por desplazamiento, es una de las técnicas de codificación de textos más simples y usadas. Es un tipo de cifrado por sustitución en el que una letra en el texto original es reemplazada por otra letra que se encuentra un número fijo de posiciones más adelante en el alfabeto. Por ejemplo, con un desplazamiento de 3 posiciones la A sería sustituida por la D (situada 3 lugares a la derecha de la A), la B sería reemplazada por la E, etc. Se supone que el alfabeto es circular de modo que a continuación de la Z comienza de nuevo la letra A.

Se propone que programe una función que reciba como parámetros una cadena de caracteres y el desplazamiento a realizar (cantidad de posiciones) y devuelva el texto codificado. Para simplificar la solución, previo a realizar la sustitución la función deberá pasar a mayúsculas la cadena recibida como argumento.

```
Ejemplo:
Si se recibe la cadena "UN TEXTO" y el desplazamiento 1, se retorna "VO UFYUP"
```

Nota: Tener en cuenta que sólo se codifican los caracteres correspondientes a letras mayúsculas del alfabeto. Los espacios en blanco que aparecen en la cadena deben mantenerse.

Problema 13: Se recibe una matriz **SopaLetras** (orden 10 x 10) de caracteres y una palabra **P** almacenada en una cadena de caracteres de longitud máxima 10.
Se debe informar:

a. Si esa palabra se encuentra en alguna fila, de derecha a izquierda. (Si se la encuentra indicarlo con un mensaje del tipo: la palabra XXXX está en la fila NN).
b. Cuántas columnas tienen únicamente letras minúsculas)

Nota: Si la palabra tiene menos de 10 caracteres (por ejemplo 6), se considera que está en una fila si la palabra aparece ocupando las últimas seis posiciones y todas las restantes son blancos.

Anexo: Funciones de Manejo de Caracteres

Prototipo	Descripción
`bool isalnum(char c);`	`(isalpha(c) || isdigit(c))`
`bool isalpha(char c);`	`(issupper(c) || islower(c))`
`bool iscntrl(char c);`	Caracteres de control.
`bool isdigit(char c);`	Dígito decimal.
`bool isgraph(char c);`	Caracteres imprimibles excepto espacio.
`bool islower(char c);`	Letra minúscula.
`bool isprint(char c);`	Caracteres imprimibles incluyendo espacio.
`bool ispunct(char c);`	Caracteres imprimibles excepto espacio, letra o dígito.
`bool isspace(char c);`	Espacio, `'\r'`, `'\n'`, `'\t'`, `'\v'`, `'\f'`.
`bool isupper(char c);`	Letra mayúscula.
`bool isxdigit(char c);`	Dígito hexadecimal.
`char tolower(char c);`	Retorna la letra minúscula correspondiente a *c*.
`char toupper(char c);`	Retorna la letra mayúscula correspondiente a *c*.
`int getchar(void);`	Lee el próximo caracter desde la entrada estándar y lo retorna como un entero.
`int putchar(int c);`	Imprime el carácter almacenado en *c*.

Anexo: Funciones de Manejo de Cadenas tipo String

Prototipo	Descripción
`string(const char *s);`	Devuelve la cadena tipo C *s* como una cadena tipo string.
`const char *c_str();`	Devuelve la cadena tipo string como una cadena tipo C terminada en `'\0'`.
`size_t length()` `size_t size()`	Devuelve la longitud de la cadena tipo string.
`istream &getline(istream &is, string &str);`	Extrae caracteres desde *is* y los guarda en *str* hasta que se encuentra el caracter `'\n'`.
`size_t find(const string &str)`	Devuelve el índice de la primer ocurrencia de *str* en la cadena. Si no se encuentra, devuelve `-1`.
`string &insert(size_t pos, const string &str);`	Inserta la cadena *str* dentro de la cadena original a partir de la posición *pos*.
`char &at(size_t pos);`	Devuelve el carácter situado en la posición *pos*. Alternativa: operador `[]`.
`string substr(size_t pos, size_t len);`	Devuelve la cadena que resulta de la composición de *len* caracteres a partir de la posición *pos*.
`void swap(string &str);`	Realiza un intercambio entre los caracteres de la cadena origen y los caracteres de *str*.
`int compare(const string& str);`	Funcionamiento equivalente a `strcmp()` de cadenas tipo C. Alternativa: operadores `==`, `!=`, `<`, `<=`, `>`, `>=`.
`string &append(const string& str);`	Concatena *str* al final de la cadena. Alternativa: operador `+`.

Anexo: Funciones de Manejo de Cadenas tipo C

Prototipo	Descripción
`char *gets(char *s);`	Lee caracteres desde la entrada estándar en el arreglo *s* hasta un caracter **newline** o **EOF**. Se agrega al array un caracter de terminación **NULL**.
`int puts(const char *s);`	Imprime el string *s* seguido por un caracter **newline**.
`char *strcpy(char *destino, const char *origen)`	Copia el valor de *destino* hacia *origen*. Precauciones: No verifica que *destino* sea lo bastante grande como para almacenar el valor de *origen*.
`char *strncpy(char *destino, const char *origen, size_t limite);`	Igual que la función **strcpy()** de dos argumentos, sólo que se copian cuando mucho *limite* caracteres.
`char *strcat(char *destino, const char *origen);`	Concatena el valor de *origen* con el final de la cadena que se encuentra en la variable *destino*. Precauciones: No verifica que *destino* sea lo bastante grande como para almacenar el resultado de la concatenación.
`char *strncat(char *destino, const char *origen, size_t limite);`	Igual que la función **strncat()** de dos argumentos, sólo que se anexan cuando mucho *limite* caracteres.
`size_t strlen(const char *s);`	Devuelve un entero igual a la longitud de *s*. (El caracter nulo `'\0'` no se cuenta en la longitud).
`int strcmp(const char *s1, const char *s2);`	Devuelve **0** si *s1* y *s2* son iguales. Devuelve **<0** si *s1* es menor que s2. Devuelve **>0** si *s1* es mayor que *s2* (es decir, devuelve un valor distinto de cero si *s1* y *s2* son distintas). El orden es lexicográfico. Precauciones: Si *s1* es igual a *s2* esta función devuelve **0**, lo que la convierte en **falsa**. Esto es lo inverso de lo que podríamos esperar que devolviera si las cadenas son iguales.
`int strncmp(const char *s1, const char *s2, size_t limite);`	Igual que la función **strcmp()** de dos argumentos, sólo que se comparan cuando mucho *limite* caracteres. Precauciones: Si *limite* se elige con cuidado, esta función es más segura que la versión de dos argumentos de **strcmp()**. No se implementa en todas las versiones de C++.
`char *strchr(const char *s, int c);`	Determina la primer ocurrencia del caracter *c* en el string *s*. Si se encuentra, se retorna un puntero a *c* en *s*. En otro caso se retorna un puntero **NULL**.
`char *strrchr(const char *s, int c);`	Determina la última ocurrencia de *c* en el string *s*. Si se encuentra, se retorna un puntero a *c* en el string *s*. En otro caso se retorna un puntero **NULL**.
`char *strpbrk(const char *s1, const char *s2);`	Determina la primer ocurrencia en el string *s1* de cualquier caracter en el string *s2*. Si se encuentra un carácter del string *s2*, se retorna un puntero al caracter en el string *s1*. En otro caso se retorna un puntero NULL.
`char *strstr(const char *s1, const char *s2);`	Determina la primer ocurrencia en el string *s1* del string *s2*. Si se encuentra, se retorna un puntero al string en *s1*. En otro caso se retorna un puntero **NULL**.
`size_t strspn(const char *s1, const char *s2);`	Determina y retorna la longitud del segmento inicial del string *s1* consistente sólo de caracteres contenidos en el string *s2*.
`size_t strcspn(const char *s1, const char *s2);`	Determina y retorna la longitud del segmento inicial del string *s1* consistente de caracteres no contenidos en el string *s2*.
`char *strtok(char *s1, const char *s2);`	Una secuencia de llamadas a **strtok()** divide al string *s1* en "tokens" - piezas lógicas como palabras en una línea de texto - separadas por caracteres contenidos en el string *s2*. La primer llamada contiene *s1* como el primer argumento, y las llamadas siguientes para continuar tokenizando el mismo string contienen **NULL** como el primer argumento. Se retorna un puntero al token actual en cada llamada. Si no hay más tokens cuando se llama a la función, se retorna **NULL**.