GUÍA DE APRENDIZAJE PARA CONCURSANTES ICPC Y IOI: CADENAS DE CARACTERES

1. Introducción

Dentro de los problemas que podemos encontrar en los ejercicios de concursos están los relacionados con el trabajo con cadenas carácteres que no son mas que el trabajo de manera eficiente con variables de tipo **string**.

Están amplio el grupo de problemas que podemos enmarcar dentro de está area que está definida dentro de las áreas de conocimiento que debe dominar un concursante de ICPC o IOI.

2. Conocimientos previos

Una **cadena de caracteres** es una secuencia de caracteres delimitada por comillas ("), como por ejemplo: "*Esto es una cadena de caracteres*". Dentro de la cadena, pueden aparecer caracteres en blanco y se pueden emplear las mismas secuencias de escape válidas para las constantes carácter. Por ejemplo, las comillas (") deben estar precedidas por (\), para no ser interpretadas como fin de la cadena; también la propia barra invertida (\). Es muy importante señalar que el compilador sitúa siempre un byte nulo (\0) adicional al final de cada cadena de caracteres para señalar el final de la misma. Así, la cadena "mesa"no ocupa 4 bytes, sino 5 bytes.

3. Desarrollo

Al considerar las cadenas como un tipo de datos, hay que definir cuáles son las operaciones que es posible hacer con ellas. En principio, podrían ser muchas y llegar a ser muy sofisticadas. Las siguientes son algunas de ellas:

1. Asignación: Consiste en asignar una cadena a otra.
2. Concatenación: Consiste en unir dos cadenas o más (o una cadena con un carácter) para formar una cadena de mayor tamaño.
3. Búsqueda: Consiste en localizar dentro de una cadena una subcadena más pequeña o un carácter.
4. Extracción: Se trata de sacar fuera de una cadena una porción de la misma según su posición dentro de ella.
5. Comparación: Se utiliza para comparar dos cadenas.

Tanto en C++ como Java las cadenas de caracteres se trabaja como si fueran objetos de clases y como clases al fin contienen un grupo de métodos que te permiten operar y manipular con la cadena de caracteres almacenadas en la variable string. La diferencia de la cadena de caracteres es que en C++ se utiliza el tipo de dato *string* mientras en Java es *String*.

4. Implementación

4.1. C++

4.1.1. Declaración y Asignación a un tipo string.

A un objeto que hayamos declarado como tipo string, se le pueden asignar otros objetos del mismo tipo, cadenas entre comillas dobles e incluso un único carácter a un tipo string, lo anterior lo ilustramos en el siguiente ejemplo de código.

```
#include <iostream>
#include <string>
using namespace std;

int main(){
   //forma #1 de inicializacion
   //aunque practicamente no se usa
   string cad_1("Hola mundo");
   //forma #2 de inicializar y asignar una cadena
   string cad_2 = "Segunda forma";

   //asignar/copiar cad_1 en cad_2
   cad_2 = cad_1;

   //asiganr un solo carcter a un tipo string
   cad_1 = 'P';

   return 0;
}
```

4.1.2. Acceso a un caracter del tipo string.

Aunque tenemos métodos como el .at() que nos retorna una referencia a esa posición dentro de la cadena, la forma más común y fácil de acceder a un caracter en una determinada posición que queramos es mediante el uso de [] (corchetes) cómo se hace con los arrays, de esta forma si tenemos un tipo string de nombre str al cual le asignamos la palabra «programa» y queremos que se nos retorne la letra que está en la posición 3 (contando desde cero, como siempre) entonces el código sería el siguiente.

```
#include <iostream>
#include <string>
using namespace std;

int main(){
  //definimos la cadena str
  string str;

  //le asignamos el contenido &quot;programa&quot;
```

```
  str = "programa";

  //mostramos en consola cual es el caracter
  //en la posicion 3 (contando desde 0)
  cout<<str[3]<<endl<<endl;

  return 0;
}
```

4.1.3. Comparaciones entre strings

La comparación entre objetos string se puede llevar a cabo fácilmente mediante el uso de los operadores ==, <=, >=, <, >, !=, que son los mismos que se usan para las operaciones de tipo lógicas, hay que aclarar que se distingue entre mayúsculas y minúsculas.

Se puede saber cuando una cadena está antes alfabéticamente hablando (es menor) o está después de la otra (es mayor) esto se debe a la organización de las letras mayúsculas y minúsculas en el estándar ascii, una buena referencia se puede encontrar en esta tabla.

En el siguiente ejemplo tenemos dos cadenas (str1 y str2) ambas con las cadenas *abcd* y *abcd* respectivamente, por lo tanto si las comparamos ambas cadenas deben ser iguales y el programa nos lo indicará en la terminal (consola).

```
#include <iostream>
#include <string>
using namespace std;

int main(){
   //creamos y asignamos las cadenas str1 y str2
   string str1="abcd", str2="abcd";
   cout<<endl;
   //comparamos si las cadenas son iguales
   if(str1==str2){
      cout<<"str1 es igual a str2"<<endl<<endl;
   }
   return 0;
}
```

4.1.4. Concatenación de Strings

El operador + (más) nos permite concatenar dos o mas cadenas, entonces en el siguiente ejemplo si tengo a str1, str2 que contienen las cadenas *Julio* y *Cesar* respectivamente puedo unir (concatenar) el contenido de str2 a str1 solo poniendo str1 = str1 + str2, o cómo más óptimo a la hora de escribir con el operador +=, entonces tenemos str1+=str2 y con esto, la cadena 1 debería contener mi nombre completo, el código es:

```
#include <iostream>
```

```
#include <string>
using namespace std;

int main(){
   //definimos e inicializamos ambas cadenas
   string str1="Julio ", str2="Cesar";

   //realizamos la concatenacion de ambas variables
   str1+=str2;

   //mostramos el nuevo contendio de str1
   cout<<"El nuevo contenido de str1 es: "<<str1<<endl;
   return 0;
}
```

4.1.5. Búsqueda de subcadenas o caracteres dentro del tipo string.

Dentro de las operaciones permitidas o implementadas en el lenguaje, se puede llevar a cabo las búsqueda de subcadenas o caracteres dentro de nuestro objeto string es decir que en una cadena como *esta es mi casa* podemos buscar y obtener la posición donde aparezca la palabra *mi* que sería la posición 8 (como siempre contando desde 0 en este lenguaje), este tipo de búsquedas se pueden llevar a cabo de izquierda a derecha o viceversa, con muchos de los métodos **.find** que implementa la clase string de C++, algunos de los más usados se presentan en el siguiente código, el cual puedes manipular para observar los diferentes comportamientos, la explicación de cada uno está en los comentarios del código.

```
#include <iostream>
#include <string>
using namespace std;

int main(){
   string str = "abcdefgh";

   /* a continuacion buscamos la subcadena "wkf" entre str, si no esta
      presente, retorna la constante npos que es equivalente a NULL en el
      tipo string, en este codigo se mostrara que la cadena no fue encontrada
       el metodo find() nos retorna la prosicion del primer caracter del
      arguento que aparece en la cadena str.*/
   if(str.find("wkf")==string::npos)
      cout<<"cadena no encontrada"<<endl;
   else
      cout<<"cadena encontrada.."<<endl;

   /* retorna la posicion del caracter que aparezca primero de esos 3 */
   cout<<str.find_first_of("cde")<<endl;

   /* retorna la posicion del caracter que aparezca de ultimo entre esos 3 */
   cout<<str.find_last_of("cde")<<endl;
```

```
    /* retorna la posicion del primer caracter en str que sea diferente a todos
        los del argumento        */
    cout<<str.find_first_not_of("cde")<<endl;

    /* retorna la posicion del ultimo caracter en str que sea diferente de
       todos los del argumento */
    cout<<str.find_last_not_of("cde")<<endl;

    return 0;
}
```

4.1.6. Obtener subcadenas de una cadena principal

Uno de los métodos que se usan frecuentemente es el que nos permite obtener una subcadena a partir de una principal, decir, si por ejemplo tengo una cadena *Hola Mundo* puedo instancias una segunda cadena y hacer que esta almacene la palabra «Mundo» de la cadena principal, esto lo logramos con el método .substr() que en su forma más básica recibe 2 parámetros, el primero de ellos indica la posición del primer caracter a obtener y el segundo parámetro indica cuantos caracteres tomaremos (a partir del que se tomó primero). Lo anterior lo ilustramos en el siguiente código, donde str1 contiene *Hola Mundo* y str2 va a contener la cadena *Mundo* que vamos a extraer de str1.

```
#include <iostream>
#include <string>
using namespace std;

int main(){
  //definimos los objetos string que vamos a usar
  string str1="Hola Mundo";
  string str2;

  //asignamos a str2 la subcadena que esta desde la posicion 5 "M"
  str2=str1.substr(5,5); //"Mundo"

  //mostramos los contenidos
  cout<<"El contenido de str1 es: "<<str1<<endl;
  cout<<"El contenido de str2 es: "<<str2<<endl;
  return 0;
}
```

4.1.7. Obtener y modificar el tamaño de una string.

Una de las cosas más importantes cuando tratamos con arrays, ya sean de tipos primitivos, o definidos por el usuario, es conocer la longitud que llegan a manejar dichos arreglos en determinadas etapas del código, en el caso del tipo string, tenemos dos métodos .length() y .size() ninguno

recibe parámetros y ambos nos retornan la longitud de la cadena (string) con la cual invoquemos el método, ambos tienen como retorno un tipo Int.

```
#include <iostream>
#include <string>
using namespace std;

int main(){
   //definimos la cadena y la inicializamos
   string str="Hola!";

   //mostramos los valores con diferentes metodos
   cout<<"Longitud con el metodo .length(): "<<str.length()<<endl;
   cout<<"Longitud con el metodo .size(): "<<str.size()<<endl;

   return 0;
}
```

Si queremos variar el tamaño (o longitud) de una cadena, tenemos el método .resize(), el cual recibe como parámetro un número entero que le indica cual es la nueva longitud de la cadena, obviamente podemos acortarla o hacerla más larga dependiendo de la necesidad de la aplicación, un ejemplo es el siguiente código, en el cual tenemos una cadena str que contiene la palabra «Hola» y cuyo tamaño inicial es 4, a continuación invocamos el método resize() y le indicamos que el nuevo tamaño será el doble de la longitud actual (usamos length() para aplicar lo que vimos antes).

```
#include <iostream>
#include <string>
using namespace std;

int main(){
   //definimos la cadena y la inicializamos
   string str="Hola!";

   //mostramos la dimension antes de usar resize()
   cout<<"nLa longitud antes de usar resize() es: "<<str.length()<<endl<<endl;

   //aplicamos el .resize() para el doble de la longitud
   str.resize(str.length()*2);

   //mostramos la nueva longitud
   cout<<"La nueva longitud de la cadena str es: "<<str.length()<<endl<<endl;

   //la cadena permanece inalterada
   cout<<"La cadena continua siendo: "<<str<<endl<<endl;

   return 0;
}
```

Luego de ejecutar el código, se observa como la longitud aumenta al doble, sin embargo el contenido de la cadena no se afecta, solo el espacio que podemos utilizar ahora, lo contrario ocurre si variamos su longitud para hacerlo menor, en ese caso si se desechan los caracteres que no alcancen a entrar en el nuevo tamaño.

4.1.8. Saber si el objeto string está vacío o contiene datos

Este método es bastante útil cuando tratamos con interfaces de usuario y tenemos que validar si algún campo de texto de la interfaz contiene o no contiene datos, el método .empty() presente en string, nos retorna un true si la cadena está vacía, o false si la cadena contiene datos.

En el siguiente código definimos una string str vacía, y luego invocamos el método empty() para verificar que no posea ningún contenido, en este caso como es obvio, la condición se cumplirá y el programa nos dirá que ese objeto string se encuentra vacío.

```
#include <iostream>
#include <string>
using namespace std;

int main(){
   //definimos la cadena y la dejamos sin contendio
   string str="";

   //preguntamos a .empty() si esta vacia
   if(str.empty()==true){
      cout<<"nLa cadena str se encuentra vacia.."<<endl<<endl;
   }else{
      cout<<"La cadena str contiene informacion.."<<endl;
   }

   return 0;
}
```

4.1.9. Leer cadenas desde el teclado

En C++ disponemos de dos métodos para capturar la entrada de texto desde el teclado, los cuales son el flujo cin y la función getline(), ¿cuál es la diferencia?, bueno, la diferencia radica en que con el flujo cin leemos una cadena solo hasta que encontremos un espacio, entonces si por ejemplo ingresamos desde el teclado la cadena *Julio César* entonces el objeto cin nos retornará solo «Julio» y lo que esté después del espacio se quedará en el buffer, por lo tanto el flujo cin, aunque no es común, lo podemos usar para cuando estamos seguros que el dato que se ingresará solo será una palabra (no ingresarán 'espacios'). Por otro lado tenemos el método getline(cin,Objeto_string) que recibe dos parámetros uno de ellos es 'cin' que le indica que la lectura se hará desde la entrada estándar (teclado) y el siguiente parámetro es el objeto string donde queremos que se almacene la información ingresada, así de simple.

Para ilustrar todo lo que se dijo anteriormente, ingresaremos desde el teclado la cadena *Hola*

Mundo y observaremos los resultados de capturar esa información con los dos métodos que se describieron.

```
#include <iostream>
#include <string>
using namespace std;

int main(){
   //definimos la cadena y la dejamos sin contendio
   string lectura1, lectura2;

   //se pide el ingreso de la cadena
   //dos veces para hacer dos lecturas
   cout<<"nIngrese la palabra: ";
   cin>>lectura1;
   //limpiamos el buffer antes de la siguiente lectura
   cin.ignore(256,'n');
   cout<<"nIngrese nuevamente la palabra: ";
   getline(cin,lectura2);

   //imprimendo el resultado de las lecturas
   cout<<"nCapturando con cin>>lectura1 se obtuvo:        "<<lectura1<<endl<<
      endl;
   cout<<"Capturando con getline(cin,lectura2) se obtuvo: "<<lectura2<<endl<<
      endl;
   return 0;
}
```

Cómo pueden ver en el código se hizo uso de cin.ignore(256,'n'), esto se usa para limpiar el buffer de entrada (bastante importante y te ahorrará dolores de cabeza) lee aquí como se usa y por qué.

4.1.10. El método .swap()

Este resulta ser un método bastante interesante y útil en algunas ocasiones, y es que recibe un parámetro de tipo string, es decir una cadena e intercambia su contenido con la cadena que invoca dicho método (.swap()), por ejemplo si tengo una cadena str1=César y otras str2=Julio y hago str1.swap(str2) ahora el contenido de las cadenas será: str1=Julio y str2=César cómo puedes notar los contenidos son cambiados entre las dos cadenas, observa y ejecuta el siguiente código:

```
#include <iostream>
#include <string>
using namespace std;

int main(){
   //definimos la cadena y la dejamos sin contendio
   string str1="Cesar";
   string str2="Julio";
```

```
    //mostramos sus contendios antes del intercambio con .swap()
    cout<<"nstr1 = "<<str1<<endl<<endl;
    cout<<"str2 = "<<str2<<endl<<endl;

    //realizamos el intercambio
    str1.swap(str2);

    cout<<"ttdespues del intercambio.."<<endl<<endl;

    //mostramos los nuevos contenidos
    cout<<"nstr1 = "<<str1<<endl<<endl;
    cout<<"str2 = "<<str2<<endl<<endl;

    return 0;
}
```

4.2. Java

En caso de Java el String presenta un grupo de métodos similares a los de C++ pero permiten realizar las mismas operaciones a manera de resumen vamos a verlos en forma tabular

Método	**Descripción**
length()	Devuelve la longitud de la cadena
indexOf(caracter)	Devuelve la posición de la primera aparición de carácter o cadena en caso de no encontrarse devuelve -1
lastIndexOf(caracter)	Devuelve la posición de la última aparición de carácter o cadena en caso de no encontrarse devuelve -1
charAt(n)	Devuelve el carácter que está en la posición n
substring(n1, n2)	Devuelve la subcadena comprendida entre las posiciones n1 y n2-1. En caso de no especificar el segundo parámetro n2 va hasta el final de la cadena
toUpperCase()	Devuelve la cadena convertida a mayúsculas
toLowerCase()	Devuelve la cadena convertida a minúsculas
equals(cad)	Compara dos cadenas y devuelve true si son iguales
equalsIgnoreCase(cad)	Igual que equals pero sin considerar mayúsculas y minúsculas
compareTo(OtroString)	Devuelve 0 si las dos cadenas son iguales. <0 si la primera es alfabéticamente menor que la segunda ó >0 si la primera es alfabéticamente mayor que la segunda.
compareToIgnoreCase(OtroString)	Igual que compareTo pero sin considerar mayúsculas y minúsculas.
valueOf(N)	Método estático. Convierte el valor N a String. N puede ser de cualquier tipo.

5. Complejidad

La mayoría de los metodos con que cuenta el tipo de dato *string* en ambos lenguajes sus complejidades dependiendo del metodo es O(1) y O(n) donde n es la cantidad de caracteres de la cadena.

6. Aplicaciones

El trabajo con cadenas de carácteres es una de las áreas de conocimiento que son abordados en los problemas de concursos por lo que es necesario conocer al menos el funcionamiento del tipo de dato **string** sus métodos y algoritmos que trabajan con este tipo de dato.

Como se pueden dar cuenta, así de fácil se pueden llegar a manipular las cadenas o el tipo string en C++ o Java, aunque hay una (muy) extensa lista de métodos e implementaciones, estas nos parecen una buena introducción, además de ser las más comunes y conocidas, pueden consultar más página de referencia oficial del lenguaje C++ para el caso de Java puede consultar la siguiente página web .

7. Ejercicios propuestos

A continuación una lista de ejercicios que se resuelven con el trabajo con cadenas de caracteres:

- DMOJ -Palíndromo Máximo.
- DMOJ - Reflexión por Grupos.
- DMOJ - Festival de Lectura de las Pulgas.
- DMOJ - La pronunciación de vocales.
- DMOJ - Tarea del Aula