GUÍA DE APRENDIZAJE PARA CONCURSANTES ICPC Y IOI: FUNCIONES DE STRING EN C++

1. Introducción

El trabajo con cadenas de caracteres 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. Es por eso que en la presente guía veremos los métodos que presenta esta colección de caracteres en el lenguaje de programación C++.

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

A continuación vamos a listar las diferentes funciones que presenta el *string* de C++ las cuales la vamos a agrupar de acuerdo al funcionamiento u objetivo de ellas:

3.1. Operaciones de entrada

- *cin.getline(str):* Función usada para leer la cadena de caracteres entrada por el usuario. La función extrae los caracteres de la entrada de *stream* mientras no encuentre un delimitador. El delimitador por defecto es el salto de línea ($\backslash n$). En el parámetro *str* será donde se almacenará la cadena de caracteres leída.
- *cin.getline(str,number):* Función con el mismo funcionamiento que la anterior lo que en este caso el parámetro *number* define la cantidad de caracteres ue deben ser leídos.
- *push_back(x):* Función utilizada para adicionar un carácter (parámetro *x* pude ser un char o un carácter literal) al final del string. Cuando esto ocurre el tamaño del string se incrementa en uno.
- *pop_back():* Función utilizada para remover un carácter del final del string. Cuando esto ocurre el tamaño del string se decrementa en uno.

3.2. Operaciones de capacidad

- *size():* Podemos encontrar la longitud de la cadena (número de caracteres). Esta función no toma ningún parámetro. Esta función devuelve el número de caracteres en el objeto de cadena.

- *length():* Podemos encontrar la longitud de la cadena (número de caracteres). Esta función no toma ningún parámetro. Esta función devuelve el número de caracteres en el objeto de cadena.
- *capacity():* La función capacity() devuelve el tamaño actual del espacio asignado para la cadena. El tamaño puede ser igual o mayor que el tamaño de la cadena. Asignamos más espacio que el tamaño de la cadena para acomodar nuevos caracteres en la cadena de manera eficiente.
- *resize(x):* Función usada para incrementar o decrementar el tamaño de un string al valor pasado por parámetro (x).
- *shrink_to_fit():* La función shrink_to_fit() se utiliza para disminuir la capacidad de la cadena para igualarla a la capacidad mínima de la cadena. Esta función nos ayuda a ahorrar memoria si estamos seguros de que no se requieren caracteres adicionales.
- *max_size():* Encuentra la longitud máxima de la cadena.
- *empty():* Comprueba si la cadena está vacía o no

3.3. Operaciones de acceso

- *at(index):* Generalmente, podemos acceder al carácter de una cadena usando el operador de subíndice de arreglo [] y la indexación. Pero std::string también tiene una función llamada at() que puede usarse para acceder a los caracteres de la cadena. index: Representa la posición del carácter en la cadena. Esta función devuelve el carácter presente en el index.
- *[index]:* El string se puede ver como un vector de char y a su vez un vector es un arreglo por lo que podemos utilizar [] para acceder al valor almacenado en una posición especificada. Al igual que la función at() devuelve el carácter presente en el index.
- *back():* Devuelve la referencia del último carácter.
- *front():* Devuelve la referencia del primer carácter.

3.4. Operaciones de iteraciones

- *begin():* Retorna un iterador que es un puntero que apunta al principio del string.
- *end():* Retorna un iterador que es un puntero que apunta al final del string. El iterador siempre apunta al carácter nulo que indica fin de la cadena.
- *rbegin():* La palabra clave rbegin significa el comienzo inverso. Se utiliza para señalar el último carácter de la cadena. La diferencia entre rbegin() y end() es que end() apunta al elemento siguiente al último elemento de la cadena, mientras que rbegin() apunta al último elemento de una cadena.
- *rend():* La palabra clave rend significa el final inverso. Se utiliza para señalar el primer carácter de la cadena.

3.5. Operaciones de manipulación

- *insert(index,str2):* La función insert() no solo nos permite agregar una cadena sino que también nos permite agregarla en la posición especificada. También es una función de la clase std::string. Donde str2: cadena a insertar. índex: posición de donde insertar la nueva cadena. Retorna una referencia a str1.
- *replace(index,size,str2):* La función replace() reemplaza la parte de la cadena con la otra cadena dada. A diferencia de insertar, se eliminan los caracteres de la parte donde se insertará la nueva cadena. Donde index: índice de dónde comenzar a reemplazar la nueva cadena. size: longitud de la parte de la cadena que se va a reemplazar. str2: nueva cadena que se va a insertar. Retorna una referencia a str1.
- *erase(start,end):* La función erase() es una función de la clase std::string que se utiliza para eliminar un carácter o una parte de la cadena. Donde start es la posición de inicio y end la posición final.
- *swap(str):* La función swap() intercambia una cadena con otra.
- *clear():* Elimina todos los elementos de la cadena.

3.6. Operaciones de generación

- *substr(start,end):* Podemos usar la función substr() para generar una parte de la cadena como un nuevo objeto de cadena. Es una función de la clase std::string. Donde start: Posición inicial de la subcadena que se generará. end: Final de la subcadena a generar.Devuelve el objeto de cadena recién creado.
- *strcpy(to,from):* Función encargada de copiar un arreglo de caracteres hacia otro arreglo de caracteres. Donde *to* es el arreglo hacia donde se va a copiar mientras en *from* es el arreglo desde donde se quiere copiar la información. La función asume que el arreglo *to* tiene una capacidad suficiente para almacenar todo el contenido.
- *setw(num):* La función de setw rellena una cadena con un carácter o espacio específico hasta un ancho determinado especificado en el parámetro *num*.
- *copy(str,length,position):* La función copy() se utiliza para copiar una subcadena a la otra cadena (matriz de caracteres) mencionada en los argumentos de la función. Se necesitan tres argumentos (mínimo dos), matriz de caracteres de destino, longitud a copiar y posición inicial en la cadena para comenzar a copiar.
- *assign():* Asigna un nuevo valor a la cadena.

3.7. Operaciones de concatenación

- *Operador +:* El operador + está sobrecargado en la clase std::string para realizar la concatenación de cadenas.

- *append(str2):* La función append() es otra función para concatenar dos cadenas. str2: esta función toma la cadena que se agregará como parámetro. Puede ser una cadena de estilo C o C++. Retorna referencia a la cadena final.
- *strcat(str1,str2):* Para usar la función strcat(), necesitamos incluir el archivo de encabezado cstring en nuestro programa. La función strcat() toma dos arreglos de caracteres como entrada. Concatena el segundo arreglo al final del primer arreglo.

3.8. Operaciones de comparación

- *Operador ==:* El operador de igualdad se puede utilizar para comparar las dos cadenas, ya que está sobrecargado para esta operación en la clase std::string. Esto devolverá verdadero si ambas cadenas son iguales, de lo contrario devuelve falso.
- *compare(str2):* La función compare() es una función de la clase std::string que se puede usar para comparar dos cadenas. str2: es la cadena a comparar. Puede ser una cadena de estilo C o C++. Si las cadenas son iguales, devuelve cero. Si str1 es mayor que str2, valor de retorno > 0. Si str2 es mayor que str1, el valor de retorno < 0.
- *compare(position,length,str2):* También podemos comparar la subcadena de str2 usando la función de compare(). dónde, position: posición de la primera subcadena de caracteres. length: longitud de la subcadena. str2: objeto de cadena a comparar.

3.9. Operaciones de búsquedas

- *find(str2):* Podemos usar la función find() de la clase std::string para comprobar si un carácter determinado o una subcadena está presente en la cadena o en una parte de la cadena. str2: puede ser una cadena de estilo C, una cadena de estilo C++ o un carácter que se va a buscar en la cadena. Devuelve el puntero a la primera ocurrencia del carácter o una subcadena en la cadena.
- *find_first_of(str):* Se utiliza para encontrar la primera aparición de la secuencia especificada por parámetros.
- *find_first_not_of(str):* Se utiliza para buscar en la cadena el primer carácter que no coincide con ninguno de los caracteres especificados en la cadena del parámetro.
- *find_last_of(str)* Se utiliza para buscar en la cadena el último carácter de la secuencia especificada por parámetro.
- *find_last_not_of(str):* Busca el último carácter que no coincide con la secuencia especificada por parámetro.

3.10. Operaciones de conversión

- *c_str():* La función c_str() es una función miembro que se utiliza para convertir la cadena de estilo C++, es decir, objetos std::string a una cadena de estilo C, es decir, una matriz

de caracteres. Esta función no toma ningún parámetro. Retorna un puntero al arreglo de caracteres equivalente.

- *stoi(str):* Función encargada de convertir una cadena de caracteres o un string en un valor número entero. El parámetro *str* debe contener un valor número entero en cadena de texto.
- *stod(str):* Función encargada de convertir una cadena de caracteres o un string en un valor número decimal. El parámetro *str* debe contener un valor número decimal en cadena de texto.
- *to_string(num):* Función encargada de convertir un valor número en una cadena de caracteres donde el parámetro *num* es una variable numerica ue contiene el valor que deseamos convertir a string.

4. Implementación

4.1. Operaciones de entrada

```
#include <iostream>
#include <bits/stdc++.h>
#include <string> // para la clase string
using namespace std;

int main(){
  // Declarando string
  string str;
  //Declarando un arreglo de char
  char str2 [12];
  // Entrada clasica del string por consola
  cin>>str;
  // Tomando el string de entrada por consola usando getline()
  getline(cin, str);
  // Otra variante de capturar el string por consola  con el
  // uso del getline() con un arreglo de char y la cantidad de caracteres
  // a leer
  cin.getline(str2,12);
  //Insertando un caracter al final del string
  str.push_back('s');
  //Eliminando el ultimo caracter
  str.pop_back();
  return 0;
}
```

4.2. Operaciones de capacidad

```
#include <iostream>
```

```
#include <bits/stdc++.h>
#include <string> // para la clase string
using namespace std;

int main(){
   // Declarando string con valor
   string str = "C++ Programming";
   //Invocando a la funcion length()
   cout << "La longitud de la cadena es: " << str.length() << endl;
   //Invocando a la funcion capacity()
   cout << "La capacidad de la cadena es: " << str.capacity() << endl;
   cout << "La cadena original es: " << str << endl;
   str.resize(10);
   cout << "La cadena despues de usar resize es: " << str << endl;
   str.resize(17);
   cout << "La capacidad de la cadena antes de usar shrink_to_fit es: "<<str.
      capacity() << endl;
   str.shrink_to_fit();
   cout << "La capacidad de la cadena despues de usar shrink_to_fit es: " <<
      str.capacity() << endl;
   if(str.empty()==true)
      cout<<"Cadena vacia"<<endl;
   else
      cout<<"Cadena no vacia"<<endl;
        return 0;
}
```

4.3. Operaciones de iteraciones

```
#include <iostream>
#include <bits/stdc++.h>
#include <string> // para la clase string
using namespace std;

int main(){
   //Incializando cadena
   string str = "Esto es una cadena de caracteres";
   //Declarando iterador
   string::iterator it;
   //Declarando iterator reverso
   string::reverse_iterator it1;
   //Visualizando la cadena
   cout<<"La cadena usando los iteradores : ";
   for(it=str.begin();it!=str.end(); it++)
      cout<<*it;
   cout<<endl;
   //Visualizando el inverso de la cadena
   cout<<""La cadena usando los iteradores inversos es: ";
```

```
   for(it1=str.rbegin();it1!=str.rend(); it1++)
      cout<<*it1;
   cout<<endl;
   return 0;
}
```

4.4. Operaciones de manipulación

```
#include <iostream>
#include <bits/stdc++.h>
#include <string> // para la clase string
using namespace std;

int main(){
   string str = "Hello, World!";
   str.replace(7,5,"Universe");  //Remplaza la subcadena "World" por "Universe
       "
   cout << str << endl;

   // Incializa la primera cadena
   string str1 = "Una cadena de caracteres";

   // Incializa la segunda cadena
   string str2 = "Otra cadena de caracteres";

   // Visualizando las cadenas antes de intercambiar
   cout << "La primera cadena antes de intercambiar: ";
   cout << str1 << endl;
   cout << "La segunda cadena antes de intercambiar : ";
   cout << str2 << endl;

   // using swap() to swap string content
   str1.swap(str2);

   // Visualizando las cadenas despues de intercambiar
   cout << "La primera cadena despues de intercambiar: ";
   cout << str1 << endl;
   cout << "La segunda cadena despues de intercambiar: ";
   cout << str2 << endl;

   string text2 = "Yo tengo un gato.";
   text2.insert(16, " negro"); // Inserta " negro" en la posicion 16
   cout<<text2<<endl;

   string text3 = "Esto es un ejemplo.";
   text3.erase(7, 3); // Borra "un "
   cout<<text3<<endl;
```

```
   return 0;
}
```

4.5. Operaciones de generación

```
#include <iostream>
#include <bits/stdc++.h>
#include <string> // para la clase string
using namespace std;

int main(){
  string str = "Hola";
  cout << setw(10) << setfill(' ') << str << endl;

  string str2 = "Hola, Mundo";
  string substr = str2.substr(6, 5);  //Extrae "Mundo" de la cadena original
  cout << "La subcadena es: " << substr << endl;

  char source[] = "Hola, Mundo"; //arreglo de caracteres de origen
  char destination[20]; // arreglo de caracteres de destino
  strcpy(destination, source); // Copia
  cout << "Cadena original: " << source <<endl;
  cout << "Cadena copiada: " << destination <<endl;

  // Inicializando 1era cadena
  string str3 = "Esto es una cadena sin sentido niguno";

  // Declarando arreglo de caracteres
  char ch[80];

  // usando copy() para copiar el contenido str3
  // en ch comenzando en la posicion 0 los siguientes
  // 13 de caracteres.
  str3.copy(ch, 13, 0);

  // Visualizando el arreglo de cadena
  cout << "La nueva cadena de caracteres copiada en el arreglo es: ";
  cout << ch << endl;
  return 0;
}
```

4.6. Operaciones de concatenación

```
#include <iostream>
#include <bits/stdc++.h>
#include <string> // para la clase string
using namespace std;
```

```
int main(){
   string str1 = "Hello";
   string str2 = " World!";
   string result = str1 + str2;
   cout << result << endl;

   string base = "Hello";
   base.append(" World!"); // Adiciona la cadena
   cout<<<base<<endl;

   char str3[50] = "Hello ";
   char str4[] = "Wordl!!!.";
   strcat(str3, str4);
   cout << str3 << endl;
   return 0;
}
```

4.7. Operaciones de comparación

```
#include <iostream>
#include <bits/stdc++.h>
#include <string> // para la clase string
using namespace std;

int main(){
   string str1 = "manzana";
   string str2 = "platano";
   if (str1 == str2) { cout << "Las cadenas son iguales" << endl;}
   else{ cout << "Las cadenas son iguales" << endl;}

   int result = str1.compare(str2);
   if (result == 0) { cout << "Las cadenas son iguales" << endl;}
   else if (result < 0){ cout << "str1 es lexicograficamente menor que str2"
      << endl; }
   else { cout << "str1 es lexicograficamente menor que str2" << endl; }
   return 0;
}
```

4.8. Operaciones de búsquedas

```
#include <iostream>
#include <bits/stdc++.h>
#include <string> // para la clase string
using namespace std;

int main(){
```

```
    string searchIn = "C++ Programming";
    size_t position = searchIn.find("Programming");
    if (position != string::npos) {
        cout << "Encontrado en la posicion:" << position << endl;
    }else {
        cout << "No encontrado" << endl;
    }
    return 0;
}
```

4.9. Operaciones de conversión

```
#include <iostream>
#include <bits/stdc++.h>
#include <string> // para la clase string
using namespace std;

int main(){
    string str = "123";
    int num = stoi(str);
    cout << num << endl;

    string str2 = "3.14";
    double num2 = stod(str2);
    cout << num2 << endl;

    int num3 = 42;
    string str3 = to_string(num3);
    cout << str3 << endl;

    string str4 = "C++";
    const char* cstr = str4.c_str();
    cout << cstr << endl;
    return 0;
}
```

5. Aplicaciones

Algunas aplicaciones comunes de las funciones de cadena en C++ incluyen:

1. **Manipulación de cadenas:** Las funciones de cadena en C++ permiten a los programadores manipular cadenas de texto, como concatenar, dividir, comparar y buscar subcadenas.
2. **Validación de entrada de usuario:** Al usar las funciones de cadena, los programadores pueden validar la entrada de usuario para garantizar que cumple con ciertos requisitos, como longitud mínima o caracteres permitidos.

3. **Formateo de texto:** Las funciones de cadena también se utilizan para formatear texto de salida, como alinear texto, agregar espacios en blanco o insertar caracteres especiales.
4. **Procesamiento de archivos de texto:** Al leer y escribir archivos de texto en C++, las funciones de cadena son esenciales para manipular el contenido del archivo, buscar patrones específicos o realizar operaciones de transformación.
5. **Análisis de datos:** En aplicaciones que requieren el análisis de datos estructurados en formato de texto, las funciones de cadena son fundamentales para extraer información relevante y realizar cálculos sobre ella.

En resumen, las funciones de cadena en C++ son muy versátiles y se utilizan en una amplia variedad de aplicaciones para manipular y procesar texto de manera eficiente.

El conocimientos y uso de las funciones que nos provee el string nos aporta dos elementos importantes en la programación competitiva, el primero es la reducción del tiempo de codificación de la solución ya que no perdemos tiempo en la implementación de ciertas funciones mientras el segundo elemento es la seguridad y eficiencia que ofrecen las funciones propias del lenguajes de programación .

6. Complejidad

Las complejidades de las funciones abordadas en esta guía varían según su uso pero oscilan entre O(1) y O(n) siendo n la cantidad de caracteres de la cadena. En cuanto a la complejidad temporal si es O(n).

7. Ejercicios

A continuación una lista de ejercicios que se resuelven con el trabajo con cadenas de caracteres y las funciones abordadas en la guía:

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