BA-UTN
INGENIERÍA EN SISTEMAS DE INFORMACIÓN

Algoritmos y Estructuras de Datos

Informe de Cadenas de Caracteres
c-string
Clase string

Lic. Hugo A. Cuello
julio-2016

ÍNDICE

Tratamiento de cadenas de caracteres en C / C++

Las cadenas de caracteres son tan importantes como los números en el tratamiento de los procesos que deba realizar una computadora. Ejemplos de tratamiento de cadenas pueden ser muy diversos, desde el reconocimiento de patrones conocido como *scanner*, que deba realizar una máquina teórica como lo son los Autómatas Finitos o una máquina de Pila para llevar a cabo el análisis sintáctico, conocido como *parser*, hasta los casos de búsqueda de palabras o frases, como lo es al escribir una línea de comandos, o buscar en la internet, la concatenación de cadenas como una de las operaciones esenciales y hacer más simple una comparación de dos o más claves, así también como ordenar una base de datos por diferentes criterios, mucho de los cuales incluyen cadenas de caracteres. Por lo tanto, hay muy buenas razones para darle importancia al tratamiento de cadenas.

El lenguaje C / C++ ofrece por medio de la librería **string.h** un conjunto de funciones para el tratamiento de cadenas de caracteres y por otro lado el lenguaje C++ en forma exclusiva, ofrece por medio de la clase string un conjunto de operadores, métodos, iteradores, constructores para el tratamiento de cadenas dinámicas con longitud variable.

Por lo tanto, el siguiente esquema establece estas formas diversas de definir tipos de string o sartas o cadenas de caracteres.

Forma "clásica" utilizada por el lenguaje C

Disposición o arreglo de caracteres

1. **char** cad[n];

Puntero a char:

2. **char** *pcad;

Forma "clase string" utilizada por el lenguaje C++ solamente

3. **string** str;

Cadenas como arreglo de caracteres

En la forma de arreglo de caracteres el espacio de almacenamiento para la cadena queda establecida en tiempo de compilación, el indicador n es un valor constante, no variable que indica la cantidad de caracteres que contendrá la cadena, la cual incluye un valor centinela o marca de fin de la cadena representado por el carácter '\0' o NULL. Por lo tanto la cadena contendrá realmente n – 1 caracteres.

Una cadena en el lenguaje C/C++ es una secuencia u ocurrencia, también llamada disposición o arreglo de caracteres, el cual finaliza cuando se encuentre un carácter especial denominado carácter nulo o NULL, representado por '\0'. En muchas ocaciones una cadena de caracteres está representado en el lenguaje C/C++ por un arreglo de caracteres, el cual debemos indicar la cantidad máxima que contendrá

incluido el carácter de terminación de la cadena o carácter nulo. Existe una relación muy estrecha entre un arreglo y un puntero, ya que el arreglo apunta al primer byte de ese conjunto de bytes que pertenecen a esa disposición de elementos. Por otra lado, estamos en condiciones de afirmar que el lenguaje no ofrece herramientas para manejar directamente una cadena de caracteres o vulgarmente conocida como strings o sartas. Si queremos copiar cadenas, o comparar cadenas, o determinar la longitud de una cadena entre otras operaciones estamos obligados a aplicar la función específica para llevar a cabo estas operaciones, como se verá más adelante. Las siguientes definiciones establecen un arreglo de caracteres o char y un puntero a carácter o char.

Ejemplos de tipo cadena

char cad[21], *pcad;

La variable **cad** es una disposición de caracteres, es un espacio asignado por el compilador de 21 bytes, las posiciones van desde 0 (cero) hasta la 19, el cual nos indica que la cadena más larga serán de 20 caracteres para este ejemplo y el espacio extra, el de la posición 20 reservado para el carácter nulo '\0', siempre y cuando la cadena contenga la cantidad máxima de caracteres que para este ejemplo es de 20 caracteres. La cadena vacía es la cadena que no contiene caracteres, y la marca de fin de la cadena '\0', se ubica en la posición cero.

Ejemplo

strcpy(cad,"abcde");

En este ejemplo observamos que la función primitiva *strcpy* copia una cadena en una variable de tipo cadena de caracteres. Las posiciones que van desde cero hasta cuatro contendrán los caracteres 'abcde' y la posición cinco tendrá la marca de fin de la cadena '\0', y esta marca se asigna de manera automática.

El siguiente gráfico detalla la situación de la variable cad:

Longitud física de 21 bytes, establecida en tiempo de compilación, es estática.

Longitud lógica, establecida en tiempo de ejecución, en este momento es de 5.

cad

a	**b**	**c**	**d**	**e**	**\0**															
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**

Si queremos averigüar el tamaño o espacio de almacenamiento físico para la variable cad, la función sizeof nos ofrece esta posibilidad:

cout << sizeof(cad) << endl; // Emite: 21.

Si quisieramos averigüar la longitud lógica de la cadena cad, la función primitiva empleada es strlen:

cout << strlen(cad) << endl; // Emite: 5.

Cadenas como puntero a caracter

Por otro lado el ejemplo para la variable **pcad**, decimos que es un puntero a char, no tiene espacio reservado en tiempo de compilación y es el programador el que deba establecer ese espacio en tiempo de ejecución. Para hacer esto se utiliza la palabra reservada **new** debemos indicarle el espacio a reservar, y si se localiza un espacio de memoria contiguo de ese tamaño retorna una dirección de memoria referida al primer byte del tamaño o bloque a reservar. Este espacio de almacenamiento se realiza en una parte de la memoria principal de la RAM, denominada HEAP o montículo, ya que es un área de memoria restante disponible para estos casos. Podría suceder que no se encuentre memoria disponible al tamaño requerido, aunque realmente haya espacio pero no contiguo. En ese caso **new** retornará un valor de dirección especial NULL o puntero que no apunta a nada.

Ejemplo de new con arreglo de caracteres

pcad = **new** char [21]; // se reserva en el heap un espacio de 21 bytes y retorna la dirección del primer byte de ese bloque de memoria contigua.

strcpy(pcad,"abcde");

cout << pcad << endl; // Emite: abcde

En cambio:

cout << *pcad << endl; // Emite: 'a'.
cout << *++pcad << endl; // Emite: 'b'.
cout << *--pcad << endl; // Emite: 'a'.
cout << *(pcad+3) << endl; // Emite: 'd'.

RAM que corresponde al heap o montículo

dirección de memoria en donde está el carácter 'a', en este caso hipotético:

AA00FF50

pcad entonces apunta a esa posición de memoria.

a	b	c	d	e	\0															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

cout << pcad[3] << endl; // Emite: 'd'.

cout << *(pcad+4) << endl; // Emite: 'e'.

Convertir cadena C++ (string str) a cadena C (char cad[n], char *cad)

strcpy(cad, str.**c_str**());

Funciones de la librería string.h

A continuación se presentan estas funciones, la convensión adoptada para los parámetros son: cad1 y cad3 son **char***, cad2 y cad4 son **const char***, car es **int**, n es de tipo **size_t**.

strcat(cad1,cad4)

Función que concatena la cadena cad4 al final de la cadena cad1. Retorna char* un puntero a char.

Codificación Cadenas Función: strcat(cad1,cad4)

```
// Uso de la función strcat
#include <iostream>
using namespace std;

int main () {
  char str[80];

  strcpy (str,"Estas ");
  strcat (str,"cadenas ");
  strcat (str,"estàn ");
  strcat (str,"concatenadas.");
  puts (str);
  return 0;
}
```

strcmp(cad3,cad4)

Función que compara las dos cadenas, si la cadena cad3 es menor que la cadena cad4, la función strcmp retorna un valor menor que cero, si son iguales, retorna cero y si es mayor la cadena cad3 que la cadena cad4, la función retorna un valor mayor que cero.

Codificación Cadenas Función: strcmp(cad3,cad4)

```
// Uso de la función strcmp
#include <iostream>
using namespace std;

main () {
  char cad1[21],
       cad2[21];

  strcpy(cad1,"Cadena uno");
```

```
  strcpy(cad2,"cadena uno");
  if (strcmp(cad1,cad2) == 0)
    cout << "Las cadenas cad1 y cad2 son iguales" << endl;
  else
    cout << "Las cadenas cad1 y cad2 son distintas" << endl;
   return 0;
}
```

strcpy(cad1,cad4)

Función que copia una cadena cad4 en la cadena cad1 hasta encontrar un NULL, '\0'. La función retorna un puntero a cad1.

Codificación Cadenas Función: strcpy(cad1,cad4)

```
// Uso de la función strcpy
#include <iostream>
using namespace std;

main() {
  char cad1[20];
        *espacio = " ";
        *cad2 = "Tratamiento"; // literar localizada por el compilador en t.ejecución
        *cad3 = "de cadenas";

  strcpy(cad1,cad2);
  strcpy(cad1,espacio);
  strcpy(cad1,cad3);
  cout << cad1 << endl;
  return 0;
}
```

```
// Uso de la función strcpy
#include <iostream>
using namespace std;

main () {
  char cad1[21],
       cad2[21];

  if (strcmp(strcpy(cad1,"Cadena uno"),strcpy(cad2,"Cadena uno")) == 0)
    cout << "Las cadenas cad1 y cad2 son iguales" << endl;
  else
    cout << "Las cadenas cad1 y cad2 son distintas" << endl;
  return 0;
```

```
}
```

strlen(cad3)

Función que retorna un valor entero que indica la cantidad de caracteres que contiene la cadena cad3.

cout << strlen("La cadena cuantos caracteres tiene") << endl;

Emite 34, ¿por qué?

Codificación Cadenas Función: strlen(cad3)

```
// Uso de la función strlen
#include <iostream>
using namespace std;

int main () {
  char szInput[256];

  cout << "Enter a sentence: ";
  gets (szInput);
  cout << "The sentence entered is " <<  "((unsigned) strlen(szInput))
      << " characters long." << endl;
  return 0;
}
```

strstr(cad3,cad4)

Busca la primer ocurrencia de una subcadena cad4, en una cadena cad3. Retorna un puntero a char si existió la ocurrencia de cad4 en cad3. Si no ocurrió entonces retorna NULL.

Ejemplos de strstr:

Codificación Cadenas Función: strstr(cad3,cad4)

```
// Uso de la función  strstr
#include <iostream>
using namespace std;

main() {
  char *cad1 = "Argentina pertenece al continente Americano";
        *cad2 = "continente";
        *pCad;

  pCad = strstr(cad1,cad2);
```

```
  cout << “La subcadena es: “ << pCad << endl;
  return 0;
}
```

strtok(cad1,cad2)

Retorna un puntero a la siguiente cadena en cad1. La cad2 representan separadores entre las cadenas. La primer llamada es cad1 pero en las sucesivas llamadas es NULL.

Codificación Cadenas Función: strtok(cad1,cad2)

```
// Uso de la función strtok
#include <iostream>
using namespace std;

int main () {
  char str[] ="- Esto, es un ejemplo de cadenas.";
  char * pch;

  cout << "Partiendo cadenas \"" << str << "\" en tokens:" << endl;
  pch = strtok (str," ,.-");
  while (pch != NULL) {
    cout << pch << endl;
    pch = strtok (NULL, " ,.-");
  }
  return 0;
}

/* Salida:
  Partiendo cadenas "- Esto, es un ejemplo de cadenas." en tokens:
  Esto
  es
  un
  ejemplo
  de
  cadenas
*/
```

strchr(cad3,car)

Retorna un puntero a la primer aparición de car en cad3, sino NULL.

Codificación Cadenas Función: strchr(cad3,car)

```
// Uso de la función strchr
#include <iostream>
using namespace std;

int main () {
```

```
  char str[] = "Esto es un ejemplo de cadenas";
  char * pch;

  cout << "Buscando el carácter  's' en: \"" << str << "\"..." << endl;
  pch=strchr(str,'s');
  while (pch!=NULL) {
    cout << "encontrado en " << pch - str+1 << endl;
    pch = strchr(pch + 1,'s');
  }
  return 0;
}

/* Salida:
  Buscando el caracter ‘s‘ en: " Esto es un ejemplo de cadenas"...
  found at 4
  found at 7
  found at 11
  found at 18
*/
```

strrchr(cad3,car)

Retorna un puntero a la última aparición de car en cad3, sino NULL.

Codificación Cadenas Función: strrchr(cad3,car)

```
// Uso de la función strrchr
#include <iostream>
using namespace std;

int main () {
  char str[] = "This is a sample string";
  char * pch;

  pch=strrchr(str,'s');
  printf ("Last occurence of 's' found at %d \n",pch-str+1);
  return 0;
}

/* Salida:
  Last occurrence of 's' found at 18
*/
```

strncpy(cad1,cad4,n)

Copia los primero n caracteres de cad4 en cad1, si cad4 tiene menos caracteres que lo indicado por n entonces rellena con ‘\0’.

Codificación Cadenas Función: strncpy(cad1,cad4,n)

```
// Uso de la función strncpy
#include <iostream>
using namespace std;

int main () {
  char str1[]= "To be or not to be";
  char str2[40];
  char str3[40];

  /* copy to sized buffer (overflow safe): */
  strncpy ( str2, str1, sizeof(str2) );
  /* partial copy (only 5 chars): */
  strncpy ( str3, str2, 5 );
  str3[5] = '\0';   /* null character manually added */
  puts (str1);
  puts (str2);
  puts (str3);
  return 0;
}

/* Salida:
  To be or not to be
  To be or not to be
  To be
*/
```

strpbrk(cad3,cad4)

Retorna un puntero a la primer aparición de algún carácter de cad4 en cad3, sino NULL.

Codificación Cadenas Función: strpbrk(cad3,cad4)

```
// Uso de la función strpbrk
#include <iostream>
using namespace std;

int main () {
  char str[] = "This is a sample string";
  char key[] = "aeiou";
  char * pch;

  printf ("Vowels in '%s': ",str);
  pch = strpbrk (str, key);
  while (pch != NULL) {
     printf ("%c " , *pch);
     pch = strpbrk (pch+1,key);
  }
  printf ("\n");
```

```
  return 0;
}

/* Salida
  Vowels in 'This is a sample string': i i a a e i
*/
```

strncat(cad1,cad3,n)

Agrega los primeros n caracteres de la cadena cad3 a la cadena de destino cad1, incorpora el terminador nulo.

Si la longitud de la cadena origen cad3 es menor que n, solo copia los caracteres de cad3 e incorpora el terminador nulo.

Codificación Cadenas Función: strncat(cad1,cad3,n)

```
// Uso de la función strncat
#include <iostream>
using namespace std;

int main () {
  char str1[20];
  char str2[20];

  strcpy (str1,"To be ");
  strcpy (str2,"or not to be");
  strncat (str1, str2, 6);
  puts (str1);
  strcpy(str1,“To be “);
  strncat(str1,str2,15);
  printf(str1);
  return 0;
}

/* Salida
  To be or not
  To be or not to be
*/
```

strncmp(cad3,cad4,n)

Compara los primeros n caracteres entre cad3 y cad3, la comparación continúa si las posiciones correspondientes en ambas cadenas los caracteres son iguales, en cuanto ocurra una diferencia, o alcance la posición indicada o una de las cadenas finalizó, entonces detiene la comparación.

Codificación Cadenas Función: strncmp(cad3,cad4,n)

```
// Uso de la función strncmp
#include <iostream>
using namespace std;

int main () {
  char str[][5] = { "R2D2" , "C3PO" , "R2A6" };
  int    n;

  puts ("Looking for R2 astromech droids...");
  for (n=0 ; n<3 ; n++)
    if (strncmp (str[n],"R2xx",2) == 0)
      printf ("found %s\n",str[n]);
  return 0;
}

/* Salida
  Looking for R2 astromech droids...
  found R2D2
  found R2A6
*/
```

strspn(cad2,cad4)

Retorna la cantidad de caracteres contenidos en cad2 que son parte de cad4.

Codificación Cadenas Función: strspn(cad2,cad4)

```
// Uso de la función strspn
#include <iostream>
using namespace std;

int main () {
  int i;
  char strtext[] = "129th";
  char cset[] = "1234567890";

  i = strspn (strtext,cset);
  printf ("The initial number has %d digits.\n",i);
  return 0;
}
```

```
/* Salida
  The initial number has 3 digits.
*/
```

strlwr(cad1) –NO ANSI-

Convierte la cad1 a minúsculas. Retorna un puntero a cad1.

Codificación Cadenas Función: strlwr(cad1)

```
// Uso de la función strlwr
#include <iostream>
using namespace std;

main() {
   char cad[51];

   strcpy(cad,"La CaDEnA 12+15 CONVERTIDA en miNUsCUlA");
   strlwr(cad);
   printf(cad);
   return 0;
}
```

strupr(cad1) –NO ANSI-

Convierte la cad1 a mayúsculas. Retorna un puntero a cad1.

Codificación Cadenas Función: strupr(cad1)

```
// Uso de la función strupr
#include <iostream>
using namespace std;

main() {
   char cad[51];

   strcpy(cad,"La CaDEnA 12+15 convertida en mAYuscULa");
   strupr(cad);
   printf(cad);
   return 0;
}
```

strrev(cad1) –NO ANSI-

Invierte la cad1. Retorna un puntero a cad1.

Codificación Cadenas Función: strrev(cad1)

```
// Uso de la función strrev
#include <iostream>
using namespace std;

main() {
  char cad[51];

  strcpy(cad,"Plateria");
  strrev(cad);
  printf(cad);
  return 0;
}
```

tolower(car) <ctype.h>

Retorna car a minúscula si es letra mayúscula, sino retorna el mismo carácter car.

Codificación Cadenas Función: tolwr(car)

```
// Uso de la función tolower
#include <iostream>
using namespace std;

int main () {
  int i=0;
  char str[]="Test String.\n";
  char c;

  while (str[i]) {
    c=str[i];
    putchar (tolower(c));
    i++;
  }
  return 0;
}

/* Salida
  test string.
*/
```

toupper(car) <ctype.h>

Retorna car a mayúscula si es letra minúscula, sino retorna el mismo carácter car.

Emite “continente Americano”, ¿por qué?

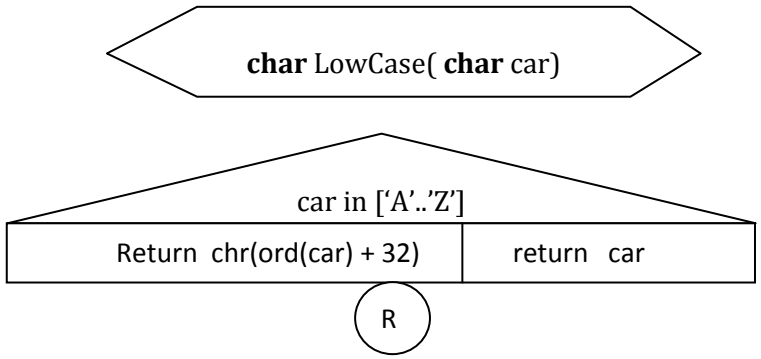
Codificación Cadenas Función: toupper (car)

```
// Uso de la función toupper
#include <iostream>
using namespace std;

int main () {
  int i=0;
  char str[]="Test String.\n";
  char c;

  while (str[i]) {
    c=str[i];
    putchar (toupper(c));
    i++;
  }
  return 0;
}

/* Salida
  TEST STRING.
*/
```

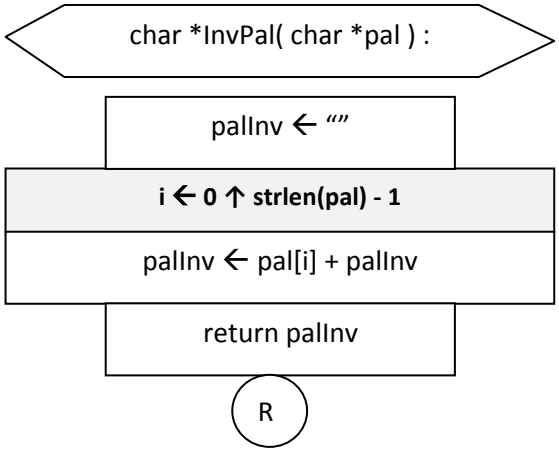
Funciones de cadenas definidas por el usuario

Al igual que con las funciones numéricas en que el usuario puede crear sus propias funciones, sucede lo mismo con las cadenas de caracteres. A continuación se presentan algunas funciones de cadenas.

Función que retorna el carácter enviado convertido a minúscula si es alfabético mayúscula, sino el mismo carácter.



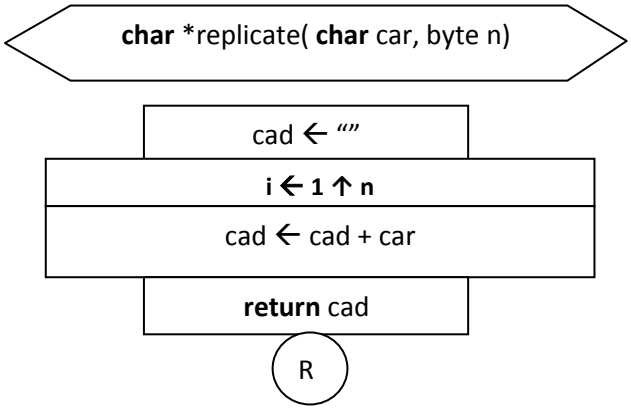
Función que retorna invertida una palabra enviada:



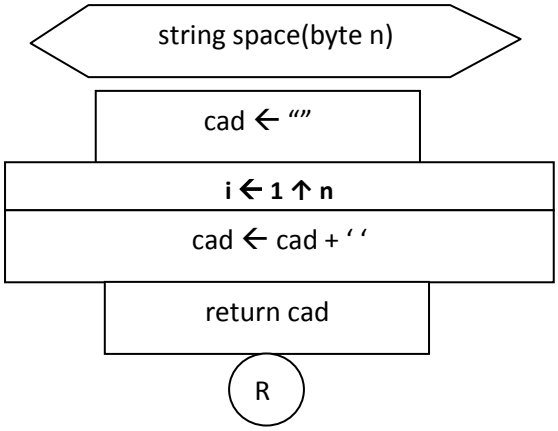
Una posible aplicación de la función *InvPal* sería comprobar si una palabra es palíndromo, es decir, capicúa, p.e.: capicua ← pal = InvPal(pal), habiendo declarado a capicua de tipo boolean, contendrá false si la palabra pal no es capicúa, sino será true. Pero, ¿podría ocurrir que al invertir una cadena, palabra de un lenguaje, la nueva cadena creada sea una palabra que pertenezca a ese mismo lenguaje?.

p.e.: palreves ← invPal(‘lámina’), obviando el tilde, asigna ‘animal’, o esta otra, palreves ← invPal(‘roma’), asigna ‘amor’.

Función que dado un caracter car y un número n, retorna el carácter car repetido n veces, es decir, retorna una cadena de longitud n, en donde cada componente es el caracter car.



También podría realizarse la siguiente función que retorna espacios en blanco:



Una posible aplicación de la función space o replicate podría ser rellenar una variable de cadena con espacios a derecha, a partir de la longitud lógica + 1 hasta el final, p.e.: cad ← 'Algoritmos'. cad ← cad + space(pred(sizeof(cad)) – length(cad)).

```
// Uso de la función strlen2.
#include <iostream.h>
#include <conio.h>

int strlen2(char *cad) {
  char *pCad = cad;

  while (*pCad != '\0')
    pCad++;
  return pCad – cad:
} // strlen2

void main() {
  char pal[] = "La cadena tiene una cantidad de caracteres";

  cout << "La longitud de la cadena es de " << strlen2(pal) << " caracteres" << endl;
  getch();
}
```

```
// Uso de la función strcpy2.
#include <iostream.h>
#include <conio.h>

int strcpy2(char *cad1, char *cad2) {
  char *cad3 = cad1;

  while (*cad3++ = *cad2++) ;
  return cad1;
} // strcpy2

void main() {
  char pal1[31],
       pal2[] = "Cadena hasta 30 caracteres";

  clrscr();
  cout << "La copia en cad1 de la cad2 es: " << strcpy2(pal1,pal2) << endl;
  getch();
}
```

```
// Uso de la función strcmp2 versión con arreglos.
#include <iostream.h>
#include <conio.h>

int strcmp2(char *cad1, char *cad2) {
  int i;

  for (i = 0; cad1[i] == cad2[i]; i++ )
    if (cad1[i] == '\0')
      return 0;
  return cad1[i] - cad2[i];
} // strcmp2

void main() {
  char pal1[] = "Cadena hasta 40 caracteres de longitud",
       pal2[] = "Cadena hasta 30 caracteres";
  int cmpPal;

  clrscr();
  cmpPal = strcmp2(pal1,pal2);
  if (cmpPal == 0)
    cout << "Las palabras son iguales" << endl;
  else
    if  (cmpPal < 0)
      cout  << "La palabra pal1 es menor que la palabra pal2" << endl;
    else
      cout << "La palabra pal1 es mayor que la palabra pal2" << endl;
  getch();
}
```

```
// Uso de la función strcmp2 versión con punteros.
#include <iostream.h>
#include <conio.h>

int strcmp2(char *cad1, char *cad2) {

  for (; *cad1 == *cad2; cad1++, cad2++)
    if (*cad1 == '\0')
      return 0;
  return *cad1 - *cad2;
} // strcmp2

void main() {
```

```
  char pal1[] = "Cadena hasta 40 caracteres de longitud",
       pal2[] = "Cadena hasta 30 caracteres";
  int cmpPal;

  clrscr();
  cmpPal = strcmp2(pal1,pal2);
  if (cmpPal == 0)
    cout << "Las palabras son iguales" << endl;
  else
    if  (cmpPal < 0)
      cout  << "La palabra pal1 es menor que la palabra pal2" << endl;
    else
      cout << "La palabra pal1 es mayor que la palabra pal2" << endl;
  getch();
}
```

```
char Lower(char car) { // (int car) es =
  if (car >= 'A' && car <= 'Z')
    return car + 'a' - 'A';
  else
    return car;
} // Lower
```

```
void Invertir(char *cad) {
  char aux;

  for (unsigned i = 0, j = strlen(cad) - 1; i < j; i++, j--)
      aux = cad[i],
      cad[i] = cad[j],
      cad[j] = aux;
} // Invertir
```

```
char *InvPal(char cad[]) {
  char aux;

  for (unsigned i = 0, j = strlen(cad) - 1; i < j; i++, j--)
      aux = cad[i],
      cad[i] = cad[j],
      cad[j] = aux;
  return cad;
} // InvPal
```

```
char *CmbCar(char cad[], char car1, char car2) {
  for (unsigned i = 0; i < strlen(cad); i++)
     if (cad[i] == car1)
        cad[i] = car2;
  return cad;
} // CmbCar
```

```
string replicate(char car, unsigned n) {
  string cad = "";

  for (unsigned i = 1; i <= n; i++)
     cad = cad + car;
  return cad;
} // replicate
```

```
char *replicate2(char car,unsigned n) {
   char *pcad;
   unsigned i;

   pcad = new char [n];
   for ( i = 0; i < n; i++)
      pcad[i] = car;
   pcad[i] = '\0';
   return pcad;
} // replicate2
```

```
char *space(unsigned n) {
   char *pcad;
   unsigned i;

   pcad = new char [n];
   for ( i = 0; i < n; i++)
      pcad[i] = ' ';
   pcad[i] = '\0';
   return pcad;
}//space
```

Tratamiento de cadenas al estilo de C++

Clase string

El lenguaje C++ ofrece otra variante al tratamiento de cadenas que lo hace semejante a otros lenguajes de programación como el Pascal o Basic, en el cual al momento de copiar o comparar o concatenar como así también otras operaciones entre cadenas, no es necesario utilizar funciones para su tratamiento, como lo es en los casos anteriormente visto para el lenguaje C.

Ejemplos de la clase string

Definición de variables de la clase string

```
string str1, str2, str3;
```

Copiar una cadena

```
str1 = "Copiar cadena";
```

Concatenar una cadena

```
str1 = str1 + " de caracteres";
```

Comparar cadenas

```
if (str1 == str2)
  cout << "Las cadenas son iguales" << endl;
if(str1 > str2)
  cout << "La cadena str1 es mayor a la cadena str2" << endl;
```

Los operadores relacionales permitidos son:

(>, <, >=, <=, ==, !=)

los cuales corresponden a:

(mayor, menor, mayor o igual, menor o igual, igual, distinto)

Además diremos que la definición de las variables de tipo string serán punteros a un espacio en la memoria dinámica -heap- para almacenar el contenido al que apunta y además este espacio para el puntero al string también podrá crecer o disminuir según las necesidades del momento.

A diferencia de la forma clásica que poseen un valor centinela al final de la cadena para indicar la finalización, representado por el carácter NULL o '\0', las cadenas de la clase string no terminan con ningún valor centinela.

Si averiguamos el espacio asignado en la definición de una variable de la clase string, a través del operador **sizeof**, este nos devolverá que su tamaño es de **16 bytes**.

El tamaño de almacenamiento de un objeto string es de 16 bytes

Podríamos interpretar que el contenido de una variable **string** pudiera contener la dirección en donde apunta al heap o memoria dinámica, pensemos que esta dirección

pesa 4 bytes, de otra dirección también al heap que apunta al final o incluso más allá del final de la cadena de caracteres asignada al heap, entonces tenemos otros cuatro 4 bytes más. Con esto completamos solo la mitad del peso de la variable string, es decir, 16 – (4 **puntero al inicio** + 4 **puntero al final**) = 8 bytes que restan del espacio máximo asignado. Otro aspecto importante de información será cuál es el tamaño actual de la cadena de caracteres, y además el espacio sobrante antes de determinar el redimensionamiento del área de almacenamiento. Suponiendo que estas otras dos componentes más debe ser determinado por valores de tipo **int**, el cual pesa otros 4 bytes, entonces, tenemos 4 bytes para indicar la **longitud de la cadena**, más otros 4 bytes para indicar la **longitud restante** antes de redimensionar el espacio de almacenamiento, nos dan otros 8 bytes, y que sumados a los 8 bytes de punteros al inicio y final del almacenamiento en el heap que le fuera asignado, completamos los 16 bytes que pesa una variable de tipo string.

Con este tipo de cadenas se evitan ciertas situaciones adversas que podrían ocurrir con el manejo tradicional de cadenas en C, por ejemplo, quedarnos sin espacio de almacenamiento al asignar una cadena a una variable de arreglo de caracteres, o de asignar una cadena a una variable puntero a char el cual no se le asigno anticipadamente de un espacio de almacenamiento, por medio de la función de disponibilidad de memoria en el heap, como lo es con malloc en el lenguaje C o con new en C++.

Las cadenas de caracteres en C++ son más versátiles que las cadenas en C. Nos otorgan mayor seguridad al momento de operar con las cadenas. Permiten crear espacios de almacenamiento en forma dinámica y este espacio dinámico una vez asignado, puede más adelante crecer o disminuir de acuerdo a las necesidades del momento; por lo que este espacio decimos que es dinámico y variable, contrario, al manejo en C que serán dinámico pero una vez creado ese espacio no podrá ser ni ampliado ni reducido, ya que será fijo.

Las cadenas en C++ son mas seguras y confiables

El espacio asignado es dinámico y variable

No obstante, este tipo de cadenas no podrán ser empleadas como tipo de datos para archivos binarios, ya que se espera de ellos, cada una de las componentes tengan longitud fija. A pesar de esto, se podrá trabajar internamente en la memoria RAM con el tipo de cadenas string y al momento de necesitar guardar en el archivo, tenemos una herramienta muy necesaria, y es la que convierte una cadena de la forma en C++, o sea, string, a la forma tradicional del lenguaje C del tipo c-string, por medio de la función brindada por C++, el cual es, c_str(), que es una función miembro. Su uso se muesta en el siguiente ejemplo:

```
char cad[21];
string str;
str = “abcdefghijklmnopqrstuvwxyz”;
strncpy(cad,str.c_str(),20);
```

```
cout << '*' << cad << '*' << endl; // Emite: *abcdefghijklmnopqrst*
str = "abcdefghij";
strncpy(cd,str.c_str(),20);
cout << '*' << cad << '*' << endl; // Emite: *abcdefghij*
```

Ejemplos de definición de variables de la clase string

```
string str1, str2;
string str3  ("Hola, mundo");
string str4 = "Todo Ok";
string str5 (10,'*');
string str6 = 'a';
```

Ejemplos de la clase string

Codificación clase string Constructor

```
// string constructor

/*
default (1)              string();
copy (2)                 string (const string& str);
substring (3)            string (const string& str, size_t pos, size_t len = npos);
from c-string (4)        string (const char* s);
from sequence (5)        string (const char* s, size_t n);
fill (6)                 string (size_t n, char c);
range (7)                template <class InputIterator>
                             string  (InputIterator first, InputIterator last);
*/

#include <iostream>
using namespace std;

int main () {
  string s0 ("Initial string");

  // constructors used in the same order as described above:
  string s1;
  string s2 (s0);
  string s3 (s0, 8, 3);
  string s4 ("A character sequence", 6);
  string s5 ("Another character sequence");
  string s6a (10, 'x');
  string s6b (10, '*');      // 42 is the ASCII code for '*'
  string s7 (s0.begin(), s0.begin()+7);
  cout << "s1: " << s1 << "\ns2: " << s2 << "\ns3: " << s3;
```

```
  cout << "\ns4: " << s4 << "\ns5: " << s5 << "\ns6a: " << s6a;
  cout << "\ns6b: " << s6b << "\ns7: " << s7 << endl;
  return 0;
}

/* Salida:
  s1:
  s2: Initial string
  s3: str
  s4: A char
  s5: Another character sequence
  s6a: xxxxxxxxxx
  s6b: **********
  s7: Initial
*/
```

Codificación clase string Operador de asignación

```
// string assigning

/*
string (1)          string& operator= (const string& str);
c-string (2)        sring& operator= (const char* s);
character (3)       string& operator= (char c);
*/

#include <iostream>
using namespace std;

int main () {
  string str1, str2, str3;
  str1 = "Test string: ";   // c-string
  str2 = 'x';               // single character
  str3 = str1 + str2;       // string

  cout << str3  << '\n';
  return 0;
}

/* Salida:
  Test string: x
*/
```

Codificación clase string Función begin - end

```
// string::begin/end

/*
     iterator begin();
     const_iterator begin() const;

     iterator end();
     const_iterator end() const;
*/

#include <iostream>
using namespace std;

int main ()
{
  string str ("Test string");
  for ( std::string::iterator it=str.begin(); it!=str.end(); ++it)
    cout << *it;
  cout << '\n';

  return 0;
}

/* Salida:
  Test string
*/
```

Codificación clase string Función rbegin - rend

```
// string::rbegin/rend

/*
     reverse_iterator rbegin();
     const_reverse_iterator rbegin() const;

     reverse_iterator rend();
      const_reverse_iterator rend() const;
*/

#include <iostream>
using namespace std;

int main () {
  string str ("now step live...");
  for (std::string::reverse_iterator rit=str.rbegin(); rit!=str.rend(); ++rit)
    cout << *rit;
  return 0;
}
```

```
/* Salida:
  ...evil pets won
*/
```

Codificación clase string Función size()

```
// string::size

/*
size_t size() const;
*/

#include <iostream>
using namespace std;

int main () {
  string str ("Test string");
  cout << "The size of str is " << str.size() << " bytes.\n";
  return 0;
}

/* Salida:
  The size of str is 11 bytes
*/
```

Codificación clase string Función resize()

```
// resizing string

/*
void resize (size_t n);
void resize (size_t n, char c);
*/

#include <iostream>
using namespace std;

int main () {
  string str ("I like to code in C");

  cout << str << '\n';
  unsigned sz = str.size();
  str.resize (sz+2,'+');
  cout << str << '\n';
  str.resize (14);
  cout << str << '\n';
  return 0;
```

```
}

/* Salida:
  I like to code in C
  I like to code in C++
  I like to code
*/
```

Codificación clase string Función clear()

```
// string::clear

/*
void clear();
*/

#include <iostream>
using namespace std;

int main () {
  char c;
  string str;

  cout << "Ingresar lineas de texto. Entrar un punto (.) para finalizar:\n";
  do {
    c = cin.get();
    str += c;
    if (c=='\n') {
       cout << str;
       str.clear();
    }
  } while (c != '.');
  return 0;
}
```

Codificación clase string Función de conversión c_str()

```
// strings and c-strings

/*
const char* c_str() const;
*/

#include <iostream>
using namespace std;

#include <cstring>
```

```
int main () {
  string str ("Please split this sentence into tokens");

  char * cstr = new char [str.length()+1];
  strcpy (cstr, str.c_str());

  // cstr now contains a c-string copy of str

  char * p = std::strtok (cstr," ");
  while (p!=0)
  {
    std::cout << p << '\n';
    p = std::strtok(NULL," ");
  }

  delete[] cstr;
  return 0;
}

/* Salida:
  Please
  split
  this
  sentence
  into
  tokens
*/
```

Codificación clase string Función find()

```
// string::find

/*
string (1)                    size_t find (const string& str, size_t pos = 0) const;
c-string (2)                  size_t find (const char* s, size_t pos = 0) const;
buffer (3)                    size_t find (const char* s, size_t pos, size_t n) const;
character (4)                 size_t find (char c, size_t pos = 0) const;
*/

#include <iostream>
using namespace std;

int main () {
  string str ("There are two needles in this haystack with needles.");
  string str2 ("needle");

  // different member versions of find in the same order as above:
  size_t found = str.find(str2);
  if (found!=std::string::npos)
```

```
  cout << "first 'needle' found at: " << found << '\n';

 found=str.find("needles are small",found+1,6);
 if (found!=std::string::npos)
  cout << "second 'needle' found at: " << found << '\n';

 found=str.find("haystack");
 if (found!=std::string::npos)
  cout << "'haystack' also found at: " << found << '\n';

 found=str.find('.');
 if (found!=std::string::npos)
  cout << "Period found at: " << found << '\n';

 // let's replace the first needle:
 str.replace(str.find(str2),str2.length(),"preposition");
 cout << str << '\n';

 return 0;
}

/*
Notice how parameter pos is used to search for a second instance of the same search string. Output:
first 'needle' found at: 14
second 'needle' found at: 44
'haystack' also found at: 30
Period found at: 51
There are two prepositions in this haystack with needles.
*/
```

Codificación clase string función substr()

```
// string::substr

/*
string substr (size_t pos = 0, size_t len = npos) const;
*/

#include <iostream>
using namespace std;

int main () {
 string str="We think in generalities, but we live in details.";
                                // (quoting Alfred N. Whitehead)

 string str2 = str.substr (3,5);     // "think"

 size_t pos = str.find("live");      // position of "live" in str
```

```
  string str3 = str.substr (pos);     // get from "live" to the end

  cout << str2 << ' ' << str3 << '\n';

  return 0;
}

/* Salida:
  think live in details.
*/
```

Codificación clase string Función replace()

```
// replacing in a string

/*
string (1)        string& replace (size_t pos,  size_t len,  const string& str);
                  string& replace (iterator i1, iterator i2, const string& str);
substring (2)     string& replace (size_t pos,  size_t len,  const string& str,
                  size_t subpos, size_t sublen);
c-string (3)      string& replace (size_t pos,  size_t len,  const char* s);
                  string& replace (iterator i1, iterator i2, const char* s);
buffer (4)        string& replace (size_t pos,  size_t len,  const char* s, size_t n);
                  string& replace (iterator i1, iterator i2, const char* s, size_t n);
fill (5)          string& replace (size_t pos,  size_t len,  size_t n, char c);
                  string& replace (iterator i1, iterator i2, size_t n, char c);
range (6)         template <class InputIterator>
                  string& replace (iterator i1, iterator i2,
                                   InputIterator first, InputIterator last);
*/

#include <iostream>
using namespace std;

int main () {
  string base="this is a test string.";
  string str2="n example";
  string str3="sample phrase";
  string str4="useful.";

  // replace signatures used in the same order as described above:

  // Using positions:                 0123456789*123456789*12345
  string str=base;           // "this is a test string."
  str.replace(9,5,str2);          // "this is an example string." (1)
  str.replace(19,6,str3,7,6);     // "this is an example phrase." (2)
  str.replace(8,10,"just a");     // "this is just a phrase."     (3)
  str.replace(8,6,"a shorty",7);  // "this is a short phrase."    (4)
```

```
  str.replace(22,1,3,'!');        // "this is a short phrase!!!"  (5)

  // Using iterators:                                    0123456789*123456789*
  str.replace(str.begin(),str.end()-3,str3);                  // "sample phrase!!!"      (1)
  str.replace(str.begin(),str.begin()+6,"replace");           // "replace phrase!!!"     (3)
  str.replace(str.begin()+8,str.begin()+14,"is coolness",7);  // "replace is cool!!!"    (4)
  str.replace(str.begin()+12,str.end()-4,4,'o');              // "replace is cooool!!!"  (5)
  str.replace(str.begin()+11,str.end(),str4.begin(),str4.end());// "replace is useful."    (6)
  scout << str << '\n';
  return 0;
}

/* Salida:
  replace is useful.
*/
```

Codificación clase string Función append()

```
// appending to string

/*
string (1)              string& append (const string& str);
substring (2)           string& append (const string& str, size_t subpos, size_t sublen);
c-string (3)            string& append (const char* s);
buffer (4)              string& append (const char* s, size_t n);
fill (5)                string& append (size_t n, char c);
range (6)               template <class InputIterator>
                         string& append (InputIterator first, InputIterator last);
*/

#include <iostream>
using namespace std;

int main () {
  string str;
  string str2="Writing ";
  string str3="print 10 and then 5 more";

  // used in the same order as described above:
  str.append(str2);                 // "Writing "
  str.append(str3,6,3);               // "10 "
  str.append("dots are cool",5);        // "dots "
  str.append("here: ");               // "here: "
  str.append(10u,'.');                // ".........."
  str.append(str3.begin()+8,str3.end());  // " and then 5 more"
  //str.append<int>(5,0x2E);              // "....." genera error
  cout << str << '\n';
  return 0;
}
```

```
/* Salida:
  Writing 10 dots here: ..... and then 5 more.....
*/
```

Codificación clase string Función getline()

```
// extract to string getline

/*
(1)                              istream& getline (istream& is, string& str, char delim);
(2)                              istream& getline (istream& is, string& str);
*/

#include <iostream>
using namespace std;

int main () {
  string name;

  cout << "Please, enter your full name: ";
  getline (cin,name);
  scout << "Hello, " << name << "!\n";

  return 0;
}

/* Salida:
  Hello, John AC!
*/
```

Codificación clase string Operador de concatenación +

```
// concatenating strings

/*
string (1)        string operator+ (const string& lhs, const string& rhs);
c-string (2)       string operator+ (const string& lhs, const char*   rhs);
                   string operator+ (const char*   lhs, const string& rhs);
character (3)      string operator+ (const string& lhs, char         rhs);
                   string operator+ (char         lhs, const string& rhs);
*/

#include <iostream>
using namespace std;

main () {
  string firstlevel ("com");
```

```
  string secondlevel ("cplusplus");
  string scheme ("http://");
  string hostname;
  string url;

  hostname = "www." + secondlevel + '.' + firstlevel;
  url = scheme + hostname;

  cout << url << '\n';

  return 0;
}

/* Salida:
  http://www.cplusplus.com
*/
```

Codificación clase string Función erase()

```
// string::erase

/*
sequence (1)                string& erase (size_t pos = 0, size_t len = npos);
character (2)               iterator erase (iterator p);
range (3)                   iterator erase (iterator first, iterator last);
*/

#include <iostream>
using namespace std;

int main () {
  string str ("This is an example sentence.");
  cout << str << '\n';
                                // "This is an example sentence."
  str.erase (10,8);                     //          ^^^^^^^^
  cout << str << '\n';
                                // "This is an sentence."
  str.erase (str.begin()+9);            //         ^
  cout << str << '\n';
                                // "This is a sentence."
  str.erase (str.begin()+5, str.end()-9);  //      ^^^^^
  cout << str << '\n';
                                // "This sentence."
  return 0;
}

/* Salida:
  This is an example sentence.
  This is an sentence.
```

```
  This is a sentence.
  This sentence.
*/
```

Codificación clase string Función append()

```
// appending to string

/*
string (1)            string& append (const string& str);
substring (2)         string& append (const string& str, size_t subpos, size_t sublen);
c-string (3)          string& append (const char* s);
buffer (4)            string& append (const char* s, size_t n);
fill (5)              string& append (size_t n, char c);
range (6)             template <class InputIterator>
                        string& append (InputIterator first, InputIterator last);
*/

#include <iostream>
using namespace std;

int main () {
  string str;
  string str2="Writing ";
  string str3="print 10 and then 5 more";

  // used in the same order as described above:
  str.append(str2);                       // "Writing "
  str.append(str3,6,3);                   // "10 "
  str.append("dots are cool",5);          // "dots "
  str.append("here: ");                   // "here: "
  str.append(10u,'.');                    // ".........."
  str.append(str3.begin()+8,str3.end());  // " and then 5 more"
  //str.append<int>(5,0x2E);              // "....." genera error
  cout << str << '\n';
  return 0;
}

/* Salida:
  Writing 10 dots here: ..... and then 5 more.....
*/
```

Codificación clase string Función copy()

```
// copy

/*
size_t copy (char* s, size_t len, size_t pos = 0) const;
*/
```

```
#include <iostream>
#include <string>

int main () {
  char buffer[20];
  string str ("Test string...");
  size_t length = str.copy(buffer,6,5);
  buffer[length]='\0';
  cout << "buffer contains: " << buffer << '\n';
  return 0;
}

/* Salida:
  buffer contains: string
*/
```

Codificación clase string Función rfind()

```
// string::rfind

/*
string (1)                  size_t rfind (const string& str, size_t pos = npos) const;
c-string (2)                size_t rfind (const char* s, size_t pos = npos) const;
buffer (3)                  size_t rfind (const char* s, size_t pos, size_t n) const;
character (4)               size_t rfind (char c, size_t pos = npos) const;
*/

#include <iostream>
using namespace std;

#include <cstddef>

int main () {
  string str ("The sixth sick sheik's sixth sheep's sick.");
  string key ("sixth");

  size_t found = str.rfind(key);
  if (found!=string::npos)
    str.replace (found,key.length(),"seventh");

  cout << str << '\n';

  return 0;
}

/*
  The sixth sick sheik's seventh sheep's sick.
*/
```

Codificación clase string Función compare()

```
// comparing apples with apples

/*
string (1)          int compare (const string& str) const;
substrings (2)   int compare (size_t pos, size_t len, const string& str) const;
                      int compare (size_t pos, size_t len, const string& str,
                          size_t subpos, size_t sublen) const;
c-string (3)       int compare (const char* s) const;
                      int compare (size_t pos, size_t len, const char* s) const;
buffer (4)          int compare (size_t pos, size_t len, const char* s, size_t n) const;
*/

#include <iostream>
using namespace std;

int main () {
  string str1 ("green apple");
  string str2 ("red apple");

  if (str1.compare(str2) != 0)
    cout << str1 << " is not " << str2 << '\n';

  if (str1.compare(6,5,"apple") == 0)
    cout << "still, " << str1 << " is an apple\n";

  if (str2.compare(str2.size()-5,5,"apple") == 0)
    cout << "and " << str2 << " is also an apple\n";

  if (str1.compare(6,5,str2,4,5) == 0)
    cout << "therefore, both are apples\n";

  return 0;
}
/*  Salida:
  green apple is not red apple
  still, green apple is an apple
  and red apple is also an apple
  therefore, both are apples
*/
```

Funciones de Verificación y Conversión de caracteres type.h

isalpha(car) : Función que verifica si el contenido del caracter car es un caracter letra mayúscula o minúscula, esto es, A-Z, a-z, en estos casos la función retorna cero por falso o 1 por verdadero.

isupper(car) : Función que verifica si el contenido del caracter car es un caracter letra mayúscula, esto es, A-Z, en estos casos la función retorna cero por falso o 1 por verdadero.

islower(cad) : Función que verifica si el contenido del caracter car es caracter letra minúsucla, esto es, a-z, en estos casos la función retorna cero por falso o 1 por verdadero.

isdigit(car) : Función que verifica si el contenido del caracter car es dígito, esto es, 0-9, en estos casos la función retorna cero por falso o 1 por verdadero.

isalnum(car) : Función que verifica si el contenido del caracter car es letra A-A, a-z o dígito 0-9, en estos casos la función retorna cero por falso o 1 por verdadero.

isspace(car) : Función que verifica si el contenido del caracter car es espacio, tabulador, nueva línea, retorno, avance de línea o tabulador vertical, en estos casos la función retorna cero por falso o 1 por verdadero.

isascii(car) : Función que verifica si el contenido del carácter car está entre 0 y 0x7F, en este caso retorna un valor distinto de cero, sino cero.

iscntrl(car) : Función que verifica si el contenido del carácter car está entre 0 y 0x1F o 0x7F (tecla DEL), en este caso retorna un valor distinto de cero, sino cero.

isgraph(car) : Función que verifica si el carácter car es cualquier carácter imprimible distinto de espacio, en este caso retorna un valor distinto de cero, sino cero.

isprint(car) : Función que verifica si el carácter car está en el intervalo [0x20, 0x7e].

ispunct(car) : Función que verifica si el carácter car es un carácter de puntuación o un espacio, en este caso retorna un valor distinto de cero, sino cero.

isxdigit(car) : Función que verifica si el contenido del carácter car es un dígito hexadecimal 0-9, A-F, a-f

toupper(car) : Función que convierte a letra mayúscula, si el contenido del carácter car es letra minúsucla, retornando el carácter convertido, o en caso contrario, el mismo carácter.

tolower(car) : Función que convierte a letra minúscula, si el contenido del carácter car es letra mayúscula, retornando el carácter convertido, o en caso contrario, el mismo carácter.

Codificación de cadenas definidas por el usuario

```
/*
   Id. Programa: MisLibsCad.cpp
   Autor.....: Lic. Hugo Cuello
```

```
   Fecha.......: ago-2014
   Comentario..: Funciones para el tratamiento de cadenas
*/

#include<iostream>
using namespace std;

char *mesCad[] = {"","ENERO","FEBRERO","MARZO","ABRIL",
                  "MAYO","JUNIO","JULIO","AGOSTO",
                  "SEPTIEMBRE","OCTUBRE","NOVIEMBRE","DICIEMBRE"
                 };

char LowCase(char car) {

  if (car >= 'A' && car <= 'Z')
    return car + 'a' - 'A';
  else
    return car;
} // LowCase

string replicate(char car, unsigned n) {
  string cad = "";

  for (unsigned i = 1; i <= n; i++)
    cad += car;
  return cad;
} // replicate

char *replicate(char car, int n) {
  char *cad;
  int i = 0;

  cad = new char [n + 1];
  for ( ; i < n; i++)
    cad[i] = car;
  cad[i] = '\0';
  return cad;
} //replicate

char *space(int n) {
  char *cad;
  int i = 0;

  cad = new char [n + 1];
  for ( ; i < n; i++)
    cad[i] = ' ';
  cad[i] = '\0';
  return cad;
} //space
```

```
string CmbCar(string cad, char car1, char car2) {

  for (unsigned i = 0; i <= cad.length(); i++)
    if (cad[i] == car1)
      cad[i] = car2;
  return cad;
} // CmbCar

string InvPal(string cad) {
  string palInv = "";

  for (unsigned i = 0; i <= cad.length(); i++)
    palInv = cad[i] + palInv;
  return palInv;
} // InvPal

char *InvPal(char *pal) {
  char aux;
  int j;

  for (int i = 0, j = strlen(pal) - 1; i < j; i++, j--) {
    aux = pal[i];
    pal[i] = pal[j];
    pal[j] = aux;
  }
  return pal;
} //InvPal

char *InvPalV(char pal[]) {
  char *pcad,
       *pcadF,
       aux;
  int j;

  pcadF = pal;
  while (*pcadF)
    pcadF++; //recorre pal hasta el final
  pcadF--;
  for (pcad = pal; pcad < pcadF; pcad++, pcadF--) {
    aux = *pcad;
    *pcad = *pcadF;
    *pcadF = aux;
  }
  return pal;
} //InvPalV

char *strlcat(char *cadDes, char *cadOri) {
  char *cadAux;

  cadAux = new char [strlen(cadDes) + strlen(cadOri) + 1];
```

```
  strcpy(cadAux,cadOri);
  strcat(cadAux,cadDes);
  strcpy(cadDes,cadAux);
  delete cadAux;
  return cadDes;
} //strlcat

string MesStr(unsigned mes) {

  switch (mes) {
    case  1: return "Enero";
    case  2: return "Febrero";
    case  3: return "Marzo";
    case  4: return "Abril";
    case  5: return "Mayo";
    case  6: return "Junio";
    case  7: return "Julio";
    case  8: return "Agosto";
    case  9: return "Septiembre";
    case 10: return "Octubre";
    case 11: return "Noviembre";
    case 12: return "Diciembre";
    default: return "";
  }
} // MesStr

char *MesStrP(unsigned mes) {

  switch (mes) {
    case  1: return "Enero";
    case  2: return "Febrero";
    case  3: return "Marzo";
    case  4: return "Abril";
    case  5: return "Mayo";
    case  6: return "Junio";
    case  7: return "Julio";
    case  8: return "Agosto";
    case  9: return "Septiembre";
    case 10: return "Octubre";
    case 11: return "Noviembre";
    case 12: return "Diciembre";
    default: return "";
  }
} // MesStrP

char *NomMes(unsigned int mes) {
    char *MesStr;

    MesStr = new char [11];
    switch (mes) {
```

```
      case  1: strcpy(MesStr,"Enero"); break;
      case  2: strcpy(MesStr,"Febrero"); break;
      case  3: strcpy(MesStr,"Marzo"); break;
      case  4: strcpy(MesStr,"Abril"); break;
      case  5: strcpy(MesStr,"Mayo"); break;
      case  6: strcpy(MesStr,"Junio"); break;
      case  7: strcpy(MesStr,"Julio"); break;
      case  8: strcpy(MesStr,"Agosto"); break;
      case  9: strcpy(MesStr,"Septiembre"); break;
      case 10: strcpy(MesStr,"Octubre"); break;
      case 11: strcpy(MesStr,"Noviembre"); break;
      case 12: strcpy(MesStr,"Diciembre"); break;
   }
   return MesStr;
} // NomMes

main() {
   char *pCad;

   pCad = NomMes(12);
   cout << pCad << endl;
   delete [] pCad;
   return 0;
}
```

```
/* strcpy example */
//char * strcpy ( char * destination, const char * source );

#include <iostream>
using namespace std;

int main () {
  char str1[] = "Sample string";
  char str2[40];
  char str3[40];

  strcpy (str2,str1);
  strcpy (str3,"copy successful");
  cout << "str1: " << str1 << endl << "str2: " << str2 << endl << "str3: " << str3 << endl;
  return 0;
}

/* Salida:
  str1: Sample string
  str2: Sample string
  str3: copy successful
*/
```

```
/* strncpy example */

#include <iostream>
using namespace std;

int main () {
  char str1[]= "To be or not to be";
  char str2[40];
  char str3[40];

  /* copy hasta el tamaño del buffer (overflow safe): */
  strncpy ( str2, str1, sizeof(str2) );

  /* copia parcial (solo 5 cars): */
  strncpy ( str3, str2, 5 );
  str3[5] = '\0';   /* caracter null agregado manualmente */

  puts (str1);
  puts (str2);
  puts (str3);

  return 0;
}

/*  Salida:
  To be or not to be
  To be or not to be
  To be
*/
```

Sub-cadena e insertar

```
/* strstr example */
// const char * strstr ( const char * str1, const char * str2 );

#include <iostream>
using namespace std;

int main () {
  char str[] ="This is a simple string";
  char * pch;

  pch = strstr (str,"simple");
  strncpy (pch,"sample",6);
  puts (str);
  return 0;
}

/*
This example searches for the "simple" substring in str and replaces that word for "sample".
```

Salida:

This is a sample string
*/

"CADENAS DE CARACTERES I - estilo clásico de C-"

TABLAS – CADENAS ESTILO C

Tabla Cadenas de Caracteres estilo C

#Ord.	Función	Significado	Ejemplos	ANSI
cad1 y **cad2** son de tipo char *; **cad3** y **cad4** son de tipo const char *; **n** es de tipo size_t; y **car** es de tipo int convertido a char. **pcad** es char *.				
1	char ***strcat**(cad1,cad4)			**SI**
	Concatena la cad4 al final de cad1. El primer carácter de cad4 se superpone sobre el carácter '\0'de cad1 y agrega el carácter '\0' después del último carácter de cad4. El espacio físico de cad1 debe soportar la nueva longitud. **Retorna** cad1.		**strcpy**(cad1,"Super"); cout << **strcat**(cad1,"man"); **Emite**: Superman.	
2	char ***strchr**(cad3,car)			SI
	Retorna un puntero a la primer aparición de car en cad3 o NULL si no está.		cout << boolalpha << (**strchr**("abcdabcdabcdefg",'d')!= NULL); **Emite**: true Cout << **strchr**("abcdabcdabcdefg", 'd'); **Emite**: dabcdabcdefg.	
3	int **strcmp**(cad3,cad4) / int **strcoll**(cad1,cad2)			SI
	Compara la cad3 con cad4. **Retorna** < 0 si cad3 < cad4, 0 si cad3 = cad4 o > 0 si cad3 > cad4. cad3[i] cmp cad4[i]		cout << boolalpha << **strcmp** ("abcd","abxdef") != 0; **Emite**: true La diferencia del código ASCII entre caracteres correspondientes cad3[i] – cad4[i].	
4	char ***strcpy**(cad1,cad4)			SI
	Copia la cad4 en cad1, incluye '\0'. **Retorna** cad1.		cout << **strcpy**(cad1,"Hola"); cout << boolalpha << **strcmp**(strcpy(cad1,"Hola"), "Hola"== 0; **Emite**: true	
5	size_t **strcspn**(cad3, cad4)			SI
	Retorna el índice del primer carácter en cad1 que se corresponde con algún carácter de cad2.		cout << **strspn**("cadena o sarta", "edst"); **Emite**: 2, ya que el carácter 'd' esta en cad2.	
6	int **stricmp**(cad3,cad4)			NO
	Compara la cad3 con cad4. **Retorna** <0 si cad3<cad4, 0 si cad3 = cad4 o >0 si cad3>cad4. **Ignora** minúsculas de mayúsculas.		cout << boolalpha << **stricmp** ("abcd","aBCd") == 0; **Emite**: true La diferencia del código ASCII entre caracteres correspondientes, salvo en los alfabéticos que considera el mismo código ASCII sin importar si es mayúscula y/o minúscula.	
7	size_t **strlen**(cad3)			SI
	Retorna la longitud de la cad3.		cout << strlen("abcde"); **Emite**: 5.	

#Ord.	Función	Significado	Ejemplos	ANSI
8	char ***strlwr**(cad1)			NO
	Convierte la cadena cad1 a minúsculas. **Retorna** la cadena cad1 convertida a minúsculas.		strcpy(cad1,"aBCde34+2xZYt"); cout << **strlwr**(cad1); **Emite**: "abcde34+2xzyt"	
9	char***strncat**(cad1,cad4,n)			**SI**
	Concatena la cad4 al final de cad1, pero hasta n caracteres, agrega '\n', **Retorna** cad1.		**strcpy**(cad1,"Super"); cout << **strncat**(cad1,"mancito",3); **Emite**: Superman.	
10	int **strncmp**(cad3,cad4,n)			SI
	Compara la cad3 con cad4. **Retorna** <0 si cad3<cad4, 0 si cad3 = cad4 o >0 si cad3>cad4, pero hasta n caracteres.		cout << boolalpha << **strncmp** ("abcdefghij","abcdexyzhij",4); **Emite**: false.	
11	char ***strncpy**(cad1,cad4,n)			SI
	Copia hasta n caracters de cad4 a cad1, si cad4 tiene menos de n caracteres rellena con '\0'. **Retorna** cad1.		cout << **strncpy**(cad1,"Funciones",7); **Emite**: Funcion. cout << **strncpy**(cad1,"Func",7); **Emite**: Func\0\0\0. pero emite Func.	
12	int **strnicmp**(cad3,cad4,n)			NO
	Compara la cad3 con cad4, sin importar las minúsculas con respecto a las mayúsculas. **Retorna** <0 si cad3<cad4, 0 si cad3 = cad4 o >0 si cad3>cad4.		cout << boolalpha << **strnicmp** ("abcdefghij","abcDExyZhij",4) != 0; **Emite**: false.	
13	char ***strnset**(cad1,car,n)			SI

	Pone el valor car en los primeros n caracteres de cad1, sobreescribiendo los n primeros caracteres. Si cad1 tiene menos caracteres que n, entonces solo sobre escribe sobre esos caracteres. **Retorna** cad1.		strcpy(cad1,"abcdefghij");
			cout << **strnset**(cad1,'*',5);
			Emite: *****fghij
			strcpy(cad1,"abc");
			cout << **strnset**(cad1,'*',5);
			Emite: ***	

14	char ***strpbrk**(cad3,cad4)			SI
	Retorna un puntero al primer carácter de la cadena apuntada por cad1, que se corresponde con algún carácter en la cadena cad2. Si no hay correspondencia retorna NULL.		cout << strpbrk("esto es una prueba","absj"); **Emite**: sto es una prueba	
15	char***strrchr**(cad3,car)			SI
	Retorna un puntero a la última ocurrencia en la cad3. Si no encuentra **retorna** NULL.		cout << **strrchr**("esto es una prueba",'s'); **Emite**: s una prueba	
16	char ***strrev**(cad1)			NO
	Invierte la cadena cad1. **Retorna** cad1.		strcpy(cad1,"abcde"); cout << **strrev**(cad1); **Emite**: edcba	
17	char ***strset**(cad1,car)			NO
	Pone car en todos los caracteres de cad1, sobreescribiendolos. **Retorna** cad1.		strcpy(cad1,"abcde"); cout << **strset**(cad1,'*'); **Emite**: *****	
18	size_t **strspn**(cad1, cad2)			SI
	Retorna el índice del primer carácter en cad1 que no se corresponde con nungún carácter de cad2.		cout << **strspn**("cadena o sarta", "eacdst"); **Emite**: 4, ya que el carácter n no esta en cad2.	
19	char ***strstr**(cad3,cad4)			SI
	Retorna un puntero a la primer aparición de cad4 en cad3 o NULL si no está.		cout<<**strstr**("abcdefghijklmnfghop" , "fgh"); // fghijklmnfghop.	

#Ord.	Función	Significado	Ejemplos	ANSI
			pcad=**strstr**(“abcdefghijabdefgxyz”, “defg”); cout << pcad << endl; **Emite**: defghijabdefgxyz;	
20	char ***strtok**(cad1,cad2)			SI

	Retorna un puntero a la siguiente cadena de cad1, los caracteres de cad2 representan separadores o delimitadores entre las cadenas. La primer invocación el primer argumento es cad1, en próximas invocaciones se indica con NULL o ‘\0’.		char *token;
			
			token = **strtok**(“Esto.es una cadena,de caracteres”,” ,.“);
			
			while (token) {
			cout << *token << endl;
			token = **strtok**(NULL,” ,.”);
			}	

21	char ***strupr**(cad1)			NO
	Convierte la cadena cad1 a mayúsculas. **Retorna** cad1.		strcpy(cad1,”aBCde34+2xYz”; cout << **strupr**(cad1); **Emite**: ABCDE34+2XYZ	

"CADENAS DE CARACTERES II – Clase string-"

TABLAS – CADENAS CLASE STRING

Tabla Cadena de Caracteres estilo C++

Función	Significado	Ejemplos

La clase string permite tratar cadenas de caracteres al estilo de otros lenguajes de programación como el Pascal o Basic, otorgando las facilidades para la manipulación de cadenas de caracteres. Al definir una variable se le asigna de un espacio dinámico variable, es decir, al asignar un nuevo contenido dinámicamente se le asigna el espacio para poder almacenar el contenido asignado, pudiendo ampliar o reducer el espacio de acuerdo a la necesidad del momento. El programador se olvida de solicitor el espacio requerido, ya que ese mecanismo es automático.
La clase string ofrece un conjunto de herramientas que se verán a continuación.

1.Constructor de objeto string.

default (1)	string();
copy (2)	string (const string& str);
substring (3)	string (const string& str, size_t pos, size_t len = npos);
from c-string (4)	string (const char* s);
from sequence (5)	string (const char* s, size_t n);
fill (6)	string (size_t n, char c);
range (7)	template <class InputIterator> string (InputIterator first, InputIterator last);

string s0 ("Initial string");	Define s0 e inicializa con la constante literal "Initial string".
string s1;	Define s1 cuyo contenido es la cadena vacía, de longitud 0.
string s2 (s0);	Define s2 copiando el contenido de la variable s0.
string s3 (s0,8,3);	Define s3 copiando de s0 los caracteres "str".
string s4 ("Another character sequence");	Define s4 copiando la cadena "A character sequence"
string s5 ("A character sequence",6);	Define s5 copiando la cadena "A char".
string s6 (10,'x');	Define s6 copiando 10 caracteres x's.
string s7 (s0.begin(),s0.begin()+7);	Define s7 copiando la cadena "Initial".

2.Operador =

string (1)	string& operator= (const string& str);
c-string (2)	string& operator= (const char* s);
character (3)	string& operator= (char c);

```
string str1, str2, str3;
char cad[21];

str1 = "Test string: ";   // c-string
str2 = 'x';               // single character
strcpy(cad,"C-string");
str3 = cad; //copia en str3, la cadena "C-string".
str3 = str1 + str2; //str3 contiene la cadena
                                    "Test string: x".
```

3.Iteradores

begin() / end() **iterator begin();** **const_iterator begin() const;**	string str ("Test string"); for (string::iterator it = str.begin(); it != str.end(); ++it) cout << *it;

	cout << '\n'; **Emite**: Test string
rbegin() / rend() **reverse_iterator rbegin();** **const_reverse_iterator rbegin() const;**	string str ("now step live..."); for(std::string::reverse_iterator rit=str.rbegin();rit!= str.rend(); ++rit) cout << *rit; **Emite**: …evil pets won
4.Capacity	
size() / length() **size_t size() const;**	string str ("Test string"); cout << "El tamaño de str es " << str.**size**() << " bytes.\n"; cout << "La longitud de str es " << str.**length**() << " bytes.\n"; **Emite**: El tamaño de str es 11 bytes **Emite**: La longitud de str es 11 bytes
max_size() **size_t max_size() const;**	string str ("Test string"); cout << "size: " << str.size() << "\n"; cout << "length: " << str.length() << "\n"; cout << "capacity: " << str.capacity() << "\n"; cout << "max_size: " << str.**max_size**() << "\n"; **Emite**: size: 11 length: 11 capacity: 15 max_size: 4294967291
resize() **void resize (size_t n);** **void resize (size_t n, char c);**	string str ("I like to code in C"); cout << str << '\n'; unsigned sz = str.size(); str.resize (sz+2,'+'); cout << str << '\n'; str.**resize** (14); cout << str << '\n'; **Emite**: I like to code in C I like to code in C++ I like to code
capacity() **size_t capacity() const;**	string str ("Test string"); cout << "size: " << str.size() << "\n"; cout << "length: " << str.length() << "\n"; cout << "capacity: " << str.**capacity**() << "\n"; cout << "max_size: " << str.max_size() << "\n"; **Emite**: size: 11 length: 11 capacity: 15 max_size: 429496729

reserve()
void reserve (size_t n = 0);
	string str;
	
	ifstream file ("test.txt",std::ios::in|std::ios::ate);
	if (file) {
	ifstream::streampos filesize = file.tellg();
	str.**reserve**(filesize);
	
	file.seekg(0);
	while (!file.eof())
	str += file.get();

	cout << str; }

clear()
void clear();
	char c;
	string str;
	cout << "Please type some lines of text. Enter a dot (.) to finish:\n";
	do {
	c = cin.get();
	str += c;
	if (c=='\n') {
	cout << str;
	str.clear();
	}
	} while (c!='.');

empty()
bool empty() const;
	string content;
	string line;
	cout << "Please introduce a text. Enter an empty line to finish:\n";
	do {
	getline(cin,line);
	content += line + '\n';
	} while (!line.empty());
	cout << "The text you introduced was:\n" << content;

5.Acceso a elementos	
Operador [ ] **char& operator[] (size_t pos);** **const char& operator[] (size_t pos) const;**	string str ("Test string"); for (int i=0; i<str.length(); ++i) cout << str[i]; **Emite**: Test string

at()
char& at (size_t pos);
const char& at (size_t pos) const;
	string str ("Test string");
	for (unsigned i = 0; i < str.length(); ++i)
	cout << str.at(i);
	Emite: Test string

6.Modificadores	
Operador += **string (1)** **c-string (2)** **character (3)**	**string& operator+= (const string& str);** **string& operator+= (const char* s);** **string& operator+= (char c);**

string name ("John");
string family ("Smith");
name += " K. "; // c-string
name += family; // string
name += '\n'; // character
cout << name;
Emite: John K. Smith	

append() **string (1)** **substring (2)** **c-string (3)** **buffer (4)** **fill (5)** **range (6)**	string& append (const string& str); string& append (const string& str, size_t subpos, size_t sublen); string& append (const char* s); string& append (const char* s, size_t n); string& append (size_t n, char c); template <class InputIterator> string& append (InputIterator first, InputIterator last);
string str; string str2="Writing ";	

```
string str3="print 10 and then 5 more";

// used in the same order as described above:
str.append(str2);                 // "Writing "
str.append(str3,6,3);             // "10 "
str.append("dots are cool",5);    // "dots "
str.append("here: ");             // "here: "
str.append(10u,'.');              // ".........."
str.append(str3.begin()+8,str3.end());  // " and then 5 more"
str.append<int>(5,0x2E);          // "....."
cout << str << '\n';
```

Emite: Writing 10 dots here: ..... and then 5 more.....

push_back()

void push_back (char c);

```
string str;
ifstream file ("test.txt",ios::in);
if (file) {
  while (!file.eof()) str.push_back(file.get());
 }
cout << str << '\n';
```

Emite: el contenido del archivo test.txt

assign()

string (1)	string& assign (const string& str);
substring (2)	string& assign (const string& str, size_t subpos, size_t sublen);
c-string (3)	string& assign (const char* s);
buffer (4)	string& assign (const char* s, size_t n);
fill (5)	string& assign (size_t n, char c);
range (6)	template <class InputIterator> string& assign (InputIterator first, InputIterator last);

```
string str;
string base="The quick brown fox jumps over a lazy dog.";

// used in the same order as described above:

str.assign(base);
cout << str << '\n';

str.assign(base,10,9);
cout << str << '\n';       // "brown fox"

str.assign("pangrams are cool",7);
cout << str << '\n';       // "pangram"

str.assign("c-string");
cout << str << '\n';       // "c-string"

str.assign(10,'*');
std::cout << str << '\n';        // "**********"

str.assign<int>(10,0x2D);
cout << str << '\n';       // "----------"

str.assign(base.begin()+16,base.end()-12);
cout << str << '\n';       // "fox jumps over"
```

insert()	
string (1)	string& **insert** (size_t pos, const string& str);
substring (2)	string& **insert** (size_t pos, const string& str, size_t subpos, size_t sublen);
c-string (3)	string& **insert** (size_t pos, const char* s);
buffer (4)	string& **insert** (size_t pos, const char* s, size_t n);
fill (5)	string& **insert** (size_t pos, size_t n, char c);
	void ins**e**rt (iterator p, size_t n, char c);
single character (6)	iterator **insert** (iterator p, char c);
range (7)	template <class InputIterator> void **insert** (iterator p, InputIterator first, InputIterator last);

erase()	
sequence (1)	string& erase (size_t pos = 0, size_t len = npos);
character (2)	iterator erase (iterator p);
range (3)	iterator erase (iterator first, iterator last);

```
string str ("This is an example sentence.");
cout << str << '\n';
                              // "This is an example sentence."
str.erase (10,8);                    //           ^^^^^^^^
cout << str << '\n';
                              // "This is an sentence."
str.erase (str.begin()+9);           //          ^
cout << str << '\n';
                              // "This is a sentence."
str.erase (str.begin()+5, str.end()-9);  //       ^^^^^
cout << str << '\n';
                              // "This sentence."
```

replace()	
string (1)	string& replace (size_t pos, size_t len, const string& str); string& replace (iterator i1, iterator i2, const string& str);
substring (2)	string& replace (size_t pos, size_t len, const string& str, size_t subpos, size_t sublen);
c-string (3)	string& replace (size_t pos, size_t len, const char* s); string& replace (iterator i1, iterator i2, const char* s);
buffer (4)	string& replace (size_t pos, size_t len, const char* s, size_t n); string& replace (iterator i1, iterator i2, const char* s, size_t n);
fill (5)	string& replace (size_t pos, size_t len, size_t n, char c); string& replace (iterator i1, iterator i2, size_t n, char c);
range (6)	template <class InputIterator> string& replace (iterator i1, iterator i2, InputIterator first, InputIterator last);

```
string base="this is a test string.";
string str2="n example";
string str3="sample phrase";
string str4="useful.";

// replace signatures used in the same order as described above:

// Using positions:                 0123456789*123456789*12345
string str=base;           // "this is a test string."
str.replace(9,5,str2);          // "this is an example string." (1)
str.replace(19,6,str3,7,6);     // "this is an example phrase." (2)
str.replace(8,10,"just a");     // "this is just a phrase."     (3)
str.replace(8,6,"a shorty",7);  // "this is a short phrase."    (4)
str.replace(22,1,3,'!');        // "this is a short phrase!!!"  (5)

// Using iterators:                                 0123456789*123456789*
```

```
str.replace(str.begin(),str.end()-3,str3);                  // "sample phrase!!!"      (1)
str.replace(str.begin(),str.begin()+6,"replace");           // "replace phrase!!!"     (3)
str.replace(str.begin()+8,str.begin()+14,"is coolness",7);  // "replace is cool!!!"    (4)
str.replace(str.begin()+12,str.end()-4,4,'o');              // "replace is cooool!!!"  (5)
str.replace(str.begin()+11,str.end(),str4.begin(),str4.end());// "replace is useful."   (6)
cout << str << '\n';
```

swap()

void swap (string& str);

```
string buyer ("money");
string seller ("goods");

cout << "Before the swap, buyer has " << buyer;
cout << " and seller has " << seller << '\n';

seller.swap (buyer);

cout << " After the swap, buyer has " << buyer;
cout << " and seller has " << seller << '\n';
```

7.Operaciones con string

c_str()

const char* c_str() const;

```
string str ("Please split this sentence into tokens");

char * cstr = new char [str.length()+1];
strcpy (cstr, str.c_str());

// cstr now contains a c-string copy of str

char * p = strtok (cstr," ");
while (p!=0) {
  cout << p << '\n';
  p = strtok(NULL," ");
}
delete[] cstr;
```

copy()

size_t copy (char* s, size_t len, size_t pos = 0) const;

```
char buffer[20];
string str ("Test string...");
size_t length = str.copy(buffer,6,5);
buffer[length]='\0';
cout << "buffer contains: " << buffer << '\n';
```

Emite: buffer contains: string

find()

string (1)	size_t find (const string& str, size_t pos = 0) const;
c-string (2)	size_t find (const char* s, size_t pos = 0) const;
buffer (3)	size_t find (const char* s, size_t pos, size_t n) const;
character (4)	size_t find (char c, size_t pos = 0) const;

```
string str ("There are two needles in this haystack with needles.");
string str2 ("needle");

// different member versions of find in the same order as above:
size_t found = str.find(str2);
if (found != string::npos)
   cout << "first 'needle' found at: " << found << '\n';

found=str.find("needles are small",found+1,6);
if (found != string::npos)
   cout << "second 'needle' found at: " << found << '\n';

found=str.find("haystack");
```

```
  if (found != string::npos)
     cout << "'haystack' also found at: " << found << '\n';

  found=str.find('.');
  if (found != string::npos)
     cout << "Period found at: " << found << '\n';

  // let's replace the first needle:
  str.replace(str.find(str2),str2.length(),"preposition");
  cout << str << '\n';
```

rfind()

string (1)	size_t rfind (const string& str, size_t pos = npos) const;
c-string (2)	size_t rfind (const char* s, size_t pos = npos) const;
buffer (3)	size_t rfind (const char* s, size_t pos, size_t n) const;
character (4)	size_t rfind (char c, size_t pos = npos) const;

```
  string str ("The sixth sick sheik's sixth sheep's sick.");
  string key ("sixth");

  size_t found = str.rfind(key);
  if (found != string::npos)
    str.replace (found,key.length(),"seventh");

  cout << str << '\n';
```

Emite: The sixth sick sheik's seventh sheep's sick.

find_first_of()

string (1)	size_t find_first_of (const string& str, size_t pos = 0) const;
c-string (2)	size_t find_first_of (const char* s, size_t pos = 0) const;
buffer (3)	size_t find_first_of (const char* s, size_t pos, size_t n) const;
character (4)	size_t find_first_of (char c, size_t pos = 0) const;

```
  string str ("Please, replace the vowels in this sentence by asterisks.");
  size_t found = str.find_first_of("aeiou");
  while (found != string::npos) {
    str[found] = '*';
    found = str.find_first_of("aeiou",found+1);
  }
  cout << str << '\n';
```

Emite: Pl**s*, r*pl*c* th* v*w*ls *n th*s s*nt*nc* by *st*r*sks.

find_last_of()

string (1)	size_t find_last_of (const string& str, size_t pos = npos) const;
c-string (2)	size_t find_last_of (const char* s, size_t pos = npos) const;
buffer (3)	size_t find_last_of (const char* s, size_t pos, size_t n) const;
character (4)	size_t find_last_of (char c, size_t pos = npos) const;

```
  cout << "Splitting: " << str << '\n';
  size_t found = str.find_last_of("/\\");
  cout << " path: " << str.substr(0,found) << '\n';
  cout << " file: " << str.substr(found+1) << '\n';
}

int main () {
  string str1 ("/usr/bin/man");
  string str2 ("c:\\windows\\winhelp.exe");

  SplitFilename (str1);
  SplitFilename (str2);
```

Emite:

```
Splitting: /usr/bin/man
path: /usr/bin
file: man
Splitting: c:\windows\winhelp.exe
path: c:\windows
file: winhelp.exe
```

find_first_not-of()

string (1)	size_t find_first_not_of (const string& str, size_t pos = 0) const;
c-string (2)	size_t find_first_not_of (const char* s, size_t pos = 0) const;
buffer (3)	size_t find_first_not_of (const char* s, size_t pos, size_t n) const;
character (4)	size_t find_first_not_of (char c, size_t pos = 0) const;

```
string str ("look for non-alphabetic characters...");

size_t found = str.find_first_not_of("abcdefghijklmnopqrstuvwxyz ");

if (found != string::npos) {
  cout << "The first non-alphabetic character is " << str[found];
  cout << " at position " << found << '\n';
}
```

Emite: The first non-alphabetic character is - at position 12

find_last_not_of()

string (1)	size_t find_last_not_of (const string& str, size_t pos = npos) const;
c-string (2)	size_t find_last_not_of (const char* s, size_t pos = npos) const;
buffer (3)	size_t find_last_not_of (const char* s, size_t pos, size_t n) const;
character (4)	size_t find_last_not_of (char c, size_t pos = npos) const;

```
string str ("Please, erase trailing white-spaces   \n");
string whitespaces (" \t\f\v\n\r");

size_t found = str.find_last_not_of(whitespaces);
if (found!=std::string::npos)
  str.erase(found+1);
else
  str.clear();            // str is all whitespace

cout << '[' << str << "]\n";
```

Emite: [Please, erase trailing white-spaces]

substr()

string substr (size_t pos = 0, size_t len = npos) const;

```
string str="We think in generalities, but we live in details.";
                                   // (quoting Alfred N. Whitehead)

string str2 = str.substr (3,5);     // "think"

size_t pos = str.find("live");      // position of "live" in str

string str3 = str.substr (pos);     // get from "live" to the end

cout << str2 << ' ' << str3 << '\n';
```

Emite: think live in details.

compare()

string (1)	int compare (const string& str) const;
substrings (2)	int compare (size_t pos, size_t len, const string& str) const; int compare (size_t pos, size_t len, const string& str, size_t subpos, size_t sublen) const;
c-string (3)	int compare (const char* s) const; int compare (size_t pos, size_t len, const char* s) const;

buffer (4)	int compare (size_t pos, size_t len, const char* s, size_t n) const;

```
string str1 ("green apple");
string str2 ("red apple");

if (str1.compare(str2) != 0)
  cout << str1 << " is not " << str2 << '\n';

if (str1.compare(6,5,"apple") == 0)
  cout << "still, " << str1 << " is an apple\n";

if (str2.compare(str2.size()-5,5,"apple") == 0)
  cout << "and " << str2 << " is also an apple\n";

if (str1.compare(6,5,str2,4,5) == 0)
  cout << "therefore, both are apples\n";
```

Emite:

green apple is not red apple
still, green apple is an apple
and red apple is also an apple
therefore, both are apples

8.Constante miembro

static const size_t npos = -1;	Es el máximo valor de tipo size_t Al asignar el valor -1 su complemento representa el valor más grande possible, ya que, size_t es un tipo entero sin signo. En las cadenas de tipo string es utilizado para alcanzar el fin de la cadena.

9.Sobrecarga de funciones no miembros

Operador +

string (1)	string operator+ (const string& lhs, const string& rhs);
c-string (2)	string operator+ (const string& lhs, const char* rhs);
	string operator+ (const char* lhs, const string& rhs);
character (3)	string operator+ (const string& lhs, char rhs);
	string operator+ (char lhs, const string& rhs);

```
string firstlevel ("com");
string secondlevel ("cplusplus");
string scheme ("http://");
string hostname;
string url;

hostname = "www." + secondlevel + '.' + firstlevel;
url = scheme + hostname;

cout << url << '\n';
```

Emite: http://www.cplusplus.com

Operadores relacionales

(1)

bool operator== (const string& lhs, const string& rhs);
bool operator== (const char* lhs, const string& rhs);
bool operator== (const string& lhs, const char* rhs);

(2)

bool operator!= (const string& lhs, const string& rhs);
bool operator!= (const char* lhs, const string& rhs);

```
        bool operator!= (const string& lhs, const char*   rhs);
(3)
        bool operator<  (const string& lhs, const string& rhs);
        bool operator<  (const char*   lhs, const string& rhs);
        bool operator<  (const string& lhs, const char*   rhs);
(4)
        bool operator<= (const string& lhs, const string& rhs);
        bool operator<= (const char*   lhs, const string& rhs);
        bool operator<= (const string& lhs, const char*   rhs);
(5)
        bool operator>  (const string& lhs, const string& rhs);
        bool operator>  (const char*   lhs, const string& rhs);
        bool operator>  (const string& lhs, const char*   rhs);
(6)
        bool operator>= (const string& lhs, const string& rhs);
        bool operator>= (const char*   lhs, const string& rhs);
        bool operator>= (const string& lhs, const char*   rhs);
```

```
string foo = "alpha";
string bar = "beta";

if (foo==bar) cout << "foo and bar are equal\n";
if (foo!=bar)  cout << "foo and bar are not equal\n";
if (foo< bar)  cout << "foo is less than bar\n";
if (foo> bar)  cout << "foo is greater than bar\n";
if (foo<=bar) cout << "foo is less than or equal to bar\n";
if (foo>=bar) cout << "foo is greater than or equal to bar\n";
```

Emite: foo and bar are not equal
foo is less than bar
foo is less than or equal to bar

swap() **void swap (string& x, string& y);**	string buyer ("money"); string seller ("goods"); cout << "Before the swap, buyer has " << buyer; cout << " and seller has " << seller << '\n'; swap (buyer,seller); cout << " After the swap, buyer has " << buyer; cout << " and seller has " << seller << '\n'; **Emite**: Before the swap, buyer has money and seller has goods After the swap, buyer has goods and seller has money
Operador >> -extractor- **istream& operator>> (istream& is, string& str);**	string name; cout << "Please, enter your name: "; cin >> name; cout << "Hello, " << name << "!\n"; **Emite**: Please, enter your name: Juan Hello, Juan
Operador << -insertor- **ostream& operator<< (ostream& os, const string& str);**	string str = "Hello world!"; cout << str << '\n'; **Emite**: Hello world!
getline() (1)	Extracts characters from is and stores them into str until the delimitation character delim is found (or the newline character, '\n', for (2)).

istream& getline (istream& is, string& str, char delim); **(2)** **istream& getline (istream& is, string& str);**	The extraction also stops if the end of file is reached in is or if some other error occurs during the input operation. If the delimiter is found, it is extracted and discarded (i.e. it is not stored and the next input operation will begin after it). Note that any content in str before the call is replaced by the newly extracted sequence. Each extracted character is appended to the string as if its member push_back was called. std::string name; cout << "Please, enter your full name: "; getline (cin,name); cout << "Hello, " << name << "!\n"; **Emite**: Please, enter your full name: Juan García Hello, Juan García!

Sitios web de obtención de información utilizadas en esta documentación

Sitios web de información empleada en el documento

http://c.conclase.net/curso/
Teoría con ejemplos de los temas explicados de C / C++ en castellano.

http://www.cplusplus.com/
Teoría con ejemplos de los temas explicados de C / C++ en inglés.

http://pseint.sourceforge.net/
Software que permite codificar en lenguaje natural y generación de los gráficos algorítmicos en programación lineal o la forma Nassi – Shneiderman.