Programación II

Cadenas de caracteres

Cadenas de caracteres

En C/C++ los vectores de caracteres son utilizados para almacenar cadenas de caracteres, estos es, palabras o contenido alfanumérico.

```
/// Diferentes formas de declarar cadenas
char cadena1[10] = "Hola";      /// Tamaño fijo
char cadena2[] = "Mundo";       /// Tamaño automático
char cadena3[] = {'H', 'o', 'l', 'a', '\0'};
const char* cadena4 = "Hola";   /// Puntero a constante
```

Cadenas de caracteres

Tienen un tratamiento especial distinto del resto de los vectores (por ejemplo se puede hacer un ***cin***>> o un ***cout***<< sobre el nombre del vector).

El tratamiento especial consiste en asignarle un carácter adicional en el ingreso ***'\0'*** para indicar el fin de la cadena.

Cadenas de caracteres

Si bien se puede trabajar carácter a carácter (utilizando cada posición del vector) para utilizar cadenas se utilizan las funciones de la librería ***cstring***.

```
strlen(str1);       /// Longitud: 4
strcpy(str2, str1); /// Copiar: str2 = "Hola“
strcat(str1, str2); /// Concatenar: str1 = "HolaHola"
strcmp(str1, str2); /// Comparar: 0 si iguales
```

Clase std::string (STL)

En C++ la clase std::string es un componente fundamental de la Biblioteca Estándar (STL), diseñada para gestionar secuencias de caracteres. Permite gestionar datos de texto, ofreciendo ventajas significativas sobre los vectores de caracteres tradicionales de C

```
string s1 = "Hola";
string s2(" Mundo");
string s3 = s1 + s2;  // "Hola Mundo“
cout << s3 << endl;
cout << "Longitud: " << s3.length() << endl;
```

¿Por qué usar getline() en lugar de cin?

¿Qué es el buffer?

El **buffer** es una zona de memoria temporal donde se almacenan los datos que el usuario escribe, antes de que el programa los lea.

Teclado → Buffer → Programa

Cuando escribimos "Hola Mundo" y presionas ENTER

```
cin >> "Hola Mundo";
```

cin >> variable lee datos del buffer hasta encontrar un espacio o salto de línea, pero no consume el espacio/enter.

¿Por qué usar getline() en lugar de cin?

Ejemplo 1

```
string palabra;
cin >> palabra;  // Usuario escribe: "Hola Mundo"
```

Proceso

Buffer: ['H','o','l','a',' ','M','u','n','d','o','\n']
cin >> lee: "Hola" (hasta el espacio)
Buffer queda: [' ','M','u','n','d','o','\n']

Resultado: palabra = "Hola"

¿Por qué usar getline() en lugar de cin?

Ejemplo 2

```
int edad;
string nombre;
cout << "Edad: ";
cin >> edad;        // Usuario escribe: 25 + ENTER
cout << "Nombre: ";
cin >> nombre;      // ¡Problema!
```

Proceso

Buffer inicial: ['2','5','\n']
cin >> edad lee 25 (número), deja \n
Buffer queda: ['\n']
cin >> nombre encuentra \n y se detiene

Resultado: nombre queda vacío

¿Por qué usar getline() en lugar de cin?

Comportamiento de getline()

getline(cin, variable) lee toda la línea hasta encontrar \n y consume/saca el \n del buffer.

```
string frase;

getline(cin, frase);  // Usuario escribe: "Hola Mundo"
```

Proceso

Buffer: ['H','o','l','a',' ','M','u','n','d','o','\n']
getline() lee: "Hola Mundo" (todo hasta \n)
Consume el \n y lo elimina del buffer
Buffer queda: [] (vacío)

Resultado: frase = "Hola Mundo"

cin >> y getline()

Caso problemático:

```
int edad;
string nombre;

cout << "Edad: ";
cin >> edad;            // Lee número pero deja \n

cout << "Nombre: ";
getline(cin, nombre);  // Encuentra \n inmediatamente
```

cin >> y getline()

Proceso

1.Usuario escribe: **25** + **ENTER**
Buffer: ['2','5','\n’]
2**. cin >> edad** ejecuta:
Lee los caracteres '**2**' y '**5**' (los convierte a número 25)
Se detiene al encontrar \n
Deja **\n** en el buffer
Buffer queda: ['\n’]
3. **getline(cin, nombre)** ejecuta:
Ve el buffer: ['\n’]
Piensa: "¡Ya encontré el salto de línea!"
Lee **0 caracteres** (cadena vacía)
Consume el \n (lo saca del buffer)
Buffer queda: []

Resultado: nombre = "" (cadena vacía)

cin >> y getline(): Solución

```
int edad;
string nombre;

cout << "Edad: ";
cin >> edad;

// Limpiar TODO lo que haya en el buffer hasta encontrar \n
cin.ignore();

cout << "Nombre: ";
getline(cin, nombre);  // Ahora funciona correctamente
```

¿Por qué funciona?

cin.ignore() remueve el **\n** que quedó en el buffer

getline() ahora encuentra buffer vacío y espera nueva entrada