Teoria (Para el ejercicio 7 Parte 1)

Cuando se lee una línea desde `cin`, se lee hasta el primer carácter `delimitador`: `espacio`, `tabulador` o `nueva línea`. En este caso, los espacios entre palabras no se leen:

```language-cpp
string name;
string surname;
cout << "Enter your name:" << endl;
cin >> name >> surname;
cout << "Hello, " << name << " " << surname << endl;
```

Si el usuario ingresa la cadena `John Doe`, entonces la palabra `John` se guardara en la variable de `name` y la palabra `Doe` en la variable de `surname`.
Para leer la línea completa, no solo la palabra actual, use la función `getline`. Lee caracteres del flujo de entrada en una cadena hasta que se encuentra un carácter de `final de línea` o se produce un error.
Este programa lee dos líneas: un nombre y una dirección. Las líneas de entrada pueden constar de varias palabras:

```language-cpp
string full_name, address;
getline(cin, full_name);
getline(cin, address);
cout << "Hello, " << full_name << " from " << address << endl;
```

Metemos los siguientes datos al programa de arriba:

```
Harry Potter
4 Privet Drive, Little Whinging, Great Britain
```

La salida del programa es:

```
Hello, Harry Potter from 4 Privet Drive, Little Whinging, Great Britain
```

En este programa, `cin` actúa como parámetro de la función `getline`. En `C++`, además de `cin`, hay otros flujos de entrada con los que puede trabajar `getline`. Se aprendera sobre ellos en lecciones futuras.

Teoria Parte 2

Las cadenas, `strings` permiten que un programa funcione con datos textuales: nombres de personas y nombres de animales, el contenido de documentos de texto, direcciones de correo electrónico y páginas web.

Las cadenas en C++ están representadas por el tipo de `string`. Para usarlo, debe incluir la biblioteca `<string>`. En esta lección, aprenderá sobre los elementos de las cadenas: `char`. Una cadena es una secuencia arbitraria de unidades elementales, símbolos del alfabeto. En cadena, cada elemento es de tipo `char`. En la mayoría de las plataformas modernas, un `char` tiene `ocho bits` y puede tener hasta $2^8 = 256$ valores diferentes. Esto es suficiente para representar los caracteres del alfabeto inglés, números, signos de puntuación y una serie de caracteres de servicio.
Para representar caracteres cirílicos, idiomas de Medio Oriente, jeroglíficos y emoji, un carácter ya no es suficiente. Según la codificación utilizada, es posible que se requieran dos o más caracteres.
En C++, el tipo `char` almacena un número igual al código del carácter. Por ejemplo, la letra `A` mayúscula en inglés en la mayoría de las plataformas tiene un código de carácter de `65`. Los caracteres literales se usan a menudo para representar caracteres, en los que el carácter está encerrado entre comillas simples. Entonces el código se vuelve más claro:

```language-cpp
char letter_a = 'A';    //  Se entiende que en el codigo se guarda la letra A
char letter_a_too = 65; // Tambien se guarda la letra A, pero no lo podemos saber a simple vista

char letter_b = letter_a + 1; // La letra B tiene el codigo siguiente a la letra A (65 + 1)
```

Para acceder a caracteres individuales de una cadena, se define una operación de indexación. Para hacer referencia a un carácter de cadena, debe especificar su número (índice) entre corchetes `[]`. La indexación de elementos de cadenas y matrices en C++ comienza desde `cero`. Después del último carácter de la cadena hay un carácter de servicio con el código `0`. Es necesario para la compatibilidad con las cadenas C.

```language-cpp
string greeting = "Hello";
char ch = greeting[1];
cout << ch; // Obtenemos el simboo e. Ya que el primer simbolo seria con el indice 0 (la letra H)
```

El operador de indexación también se puede usar para cambiar los caracteres de una cadena:

```language-cpp
// En greeting estaba la palabra Hello
greeting[0] = 'Y'; // Ahora en greeting se contiene "Yello"
```

Ten cuidado. El índice de caracteres debe estar en el rango `[0; longitud de la string)`. Si especifica un índice fuera de este rango, el comportamiento del programa será indefinido.

Los operadores `+` y `+=` pueden agregar no solo cadenas, sino también caracteres individuales a una cadena:

```language-cpp
// En greeting habia la palabra Yello
greeting += 'w';   // Dentro de la string greeting ahora se tiene "Yellow".
Esta liena de codigo equivale a greeting = greeting + 'w'

cout << greeting << endl;
```

El tipo `char` no almacena una cadena, sino un código de carácter. Por lo tanto, si agrega dos valores del tipo `char`, no será la concatenación de caracteres en una cadena, sino la adición de códigos de caracteres. El resultado será un número:

```language-cpp
cout << 'A' + 'B' << endl; // Se obtiene el numero 131, el cual es igual a
la suma de 65 y 66 (los codigos para A y B, respectivamente)
```

Teoria (Ejercicio 9 y 10)

Seguramente ha notado en algunos ejercicios anteriores de C++ el uso de `""s`-literals cuando se trabaja con cadenas. Ahora, aprenderá sobre situaciones en las que tales literales protegen su código de errores.

Concatenación de literales de cadena

Ya sabes que el operador `+` se usa para concatenar cadenas. Funciona bien con variables, incluso cuando se inicializan con literales de cadena regulares, sin la s después de las comillas:

```language-cpp
#include <iostream>
using namespace std;

int main() {
    string i_know = "I know "; // Simple string sin la al final `s`
    string cpp = "C++";        // Simple string sin la al final `s`
    cout << i_know + cpp << endl;
}
```

Sin embargo, usar `+` para agregar literales de cadena normales no funcionará:

```language-cpp
#include <iostream>
using namespace std;

// Pruebe este codigo en el compilador
// Que observa?
int main() {
    cout << "I know " + "C++" << endl;
}
```

El error obtenido se debe a que el operador `+` no se puede usar para agregar tipos `const char [8]` y `const char [4]`. Los literales de cadena `"I know"` y `"C++"` son literales de cadena `C` o `cadenas C`. El compilador los convierte en matrices de caracteres, que se rellenan con el código de carácter 0. Es por eso que el tamaño de la matriz es uno más que el número de caracteres dentro de las comillas. Sea como fuese, las matrices de caracteres no son cadenas y no tienen operaciones `+`.

Para corregir el error, use `""s`-literals, que crean directamente un objeto de tipo cadena. Cuando el tipo de al menos uno de los sumandos es cadena, la operación de suma funcionará como se esperaba:

```language-cpp
#include <iostream>
using namespace std;

int main() {
    cout << "I know "s + "C++"s << endl;
    // Lo siguiente tambien funciona
    // cout << "I know"s + "C++" << endl;
    // cout << "I know" + "C++"s << endl;
}
```

Adición de cadenas y números

JavaScript, Java, C# y varios otros lenguajes le permiten agregar cadenas y números. Pero en C++, no puede agregar cadenas y números:

```language-cpp
// Intentelo en el compilador y revise el error
string s = "Hello"s;
string error = s + 1; // Error!
```

Las cadenas y los números son tipos diferentes, y para ellos, la operación `+` realiza acciones completamente diferentes: concatenación de cadenas y adición de números. En `JavaScript`, si uno de los argumentos de una operación de suma es una cadena, el otro argumento se convierte implícitamente en una cadena. Entonces las cadenas se concatenan. En `C++`, la conversión de un número a una cadena y viceversa debe hacerse explícitamente. Las funciones para esto son:

- `stoi` - String to Integer
- `stod` - String to Double
- `to_string` - To String

```language-cpp
// Pruebe este codigo en el compilador (use cout para verificar las variables)
int x = stoi(x_str);      // string → int
double y = stod(x_str);   // string → double
```

```
string s = to_string(x);  // int or double → string

int z = 42;
string t = "Hello"s + to_string(z); // En t se guardara el valor "Hello42"
```

Las funciones `stoi` y `stod` fallarán si la cadena que se les pasa no se puede convertir a un número entero o real. Dichos errores deben manejarse adecuadamente, de lo contrario, el programa fallará. Cómo procesarlos exactamente, se vera en lecciones futuras.

Al agregar cadenas y números, se produce un error de compilación. En particular, esto sucede cuando se suman `""s`-literales y enteros. Los `""s`-literals protegen contra los errores ocasionales que son inherentes a los literales de cadena normales.
`C++` heredó de `C` la capacidad de agregar cadenas literales simples a los números. En este caso, el código compila sin errores, pero cuando se ejecuta, se obtiene un resultado inusual, que incluso puede provocar que el programa se rompa:

```language-cpp
string s = "Hello" + 1; // En la variable s tendremos el valor "ello"
string q = 1 + "Hello"; // En la variable q tambien habra el valor "ello"

// Aquí el comportamiento del programa será indefinido,
// ya que el argumento numérico excede la longitud de la cadena:
// string e = "Hello" + 10000;
```

Mientras tanto, siempre use `""s`-literals cuando trabaje con cadenas. Protegerán su código de sorpresas.

Tamaño de string

La función de `size` le permite averiguar el tamaño de la cadena. La sintaxis para llamarlo: primero viene el objeto de cadena, luego el nombre de la función se escribe a través del punto:

```language-cpp
// La funcion size es invocado en una cadena creada por un literal de
cadena "Hello"s
int length = "Hello"s.size(); // El valor de length es igual a 5

string greeting = "Hi"s;
// La funcion size se llama en la string greeting
int greeting_size = greeting.size(); // El valor de greeting_size es 2
greeting = "Bye"s;
greeting_size = greeting.size(); // El valor de greeting_size es 3
```

Estas funciones se denominan funciones miembro o métodos. Veremos más información sobre los métodos más adelante en el curso.
El tamaño de la cadena devuelta por el método de tamaño se calcula en elementos de tipo

`char`. Esta no es necesariamente la misma cantidad de caracteres gráficos que verá el usuario:

```cpp
cout << "Hello"s.size() << endl // 5
     << "Привет"s.size() << endl // 12 — cirílico en UTF-8
     << "👋🏽"s.size() << endl; // 8 — emoji
```

Note que si una cadena consta únicamente de letras latinas, números, espacios y signos de puntuación básicos, su tamaño será igual a su longitud. Por lo tanto, a menudo hablaremos de la longitud de una cadena, es decir, de su tamaño.