C++ para Principiantes

Alex Dannis Lipa Quispe

2025-05-29

Table of contents

1 INTRODUCCIÓN

Bienvenio/a a este libro de "C++ Para Principiantes", darte la cálida vienvenida a este libro que fue echa especialmente para personas que a si como yo, teinen cierta dificultad entender y escribir código de manera fluida. Este es un pequeño paso para mi persona ya que será mi primer libro de programación y espero que pueda hacer muchos más.

Este libro tomará conceptos básicos sobre la programación, pero tocara varios temas importantes, empezando desde el hola mundo terminando en loq ue tenga que acabar… El libro tendrá explicación detallada sobre cada parte del código de una manera clara y fáscil de entender por lo que te pido que solo tengas los ánimos para poder aprender de una manera más divertida.

C++ es un lenguaje de programación de propósito general creado por **Bjarne Stroustrup** en los años 80, como una extensión del lenguaje C. Su objetivo era combinar la eficiencia y el control de C con características más modernas, como la **programación orientada a objetos**.

Desde entonces, C++ se ha convertido en uno de los lenguajes más utilizados del mundo, especialmente en áreas donde se necesita rendimiento, como:

- Desarrollo de **videojuegos**
- **Sistemas operativos**
- Programas que necesitan acceso directo a hardware
- **Software financiero** de alto rendimiento
- Aplicaciones **científicas y de ingeniería**

Características principales de C++

- **Compilado**: el código debe convertirse a lenguaje máquina antes de ejecutarse.
- **Tipado estático**: debes declarar el tipo de cada variable.
- **Eficiente y rápido**: ideal para tareas exigentes.
- **Multiplataforma**: puedes usarlo en Windows, Linux y macOS.
- **Paradigma múltiple**: permite programación estructurada, orientada a objetos y genérica.

2 Variables y Tipos de Datos

Los tipos de datos son un tipo de valor que se asignan a las variables para que puedan almacenar un tipo de valor en específico.

2.1 variables:

Las variables pueden ser cualquier tipo de palabra excepto las palabras reservadas del leguaje de programación que estes usando, como buena practica te recomiendo que las variables que uses sean fáciles de leer como por ejemplo “numero”, “nombre”. Esto ayudara a que tu código sea fácil de entender en tu entorno de trabajo.

2.2 Datos numéricos:

- **float :** ocupa 32 bits, puede contener desde 1.4 x 10^{-45} hasta el valor de 3.4028235 x 10^{38}.
- **double:** ocupa 64 bits, puede contener desde 4.9 x 10^{324} hasta el valor de 1.7976931348623157 x 10^{308}.
- **short :** ocupa 16 bits, puede contener desde -32768 hasta el valor de 32767.
- **int :** ocupa 32 bits, puede contener desde -2147483648 hasta el valor de 2147483647.
- **long :** ocupa 64 bits, puede ocupar desde -9223372036854775805 hasta el valor de 9223372036854775807.

2.3 Cadena de caracteres:

- **char :** ocupa 8 bits, puede contener desde 0 (carácter nulo ‘\0’) hasta el valor de 127 (signed char) o 255 (unsigned char).
- **string :** ocupa 8 bits por cada carácter, puede contener desde “ ” (cadena vacía) hasta el valor que depende de la memoria disponible.

Estos son los tipos de datos que asignaremos a nuestras variables en diferentes ejemplos y explicaciones de código.

3 Operadores

3.1 Matemáticos:

Son operadores que conocemos desde la primaria, las cuales son:

- Suma: denotado por (+).
- Resta: denotado por = (-).
- Multiplicación: denotado por (*).
- División: denotado por (/).
- Residuo: denotado por (%).

Estos operadores nos sirven para que nuestros programas puedan ejecutar las operaciones aritméticas que nosotros conocemos por decir:

OPERADORES ARITMÉTICOS	OPERACIÓN	RESULTADO
Suma	8 + 9	17
Resta	10 - 9	1
Multiplicación	7 * 2	14
División	21 / 3	7
Residuo	21 % 2	0

3.2 Relacionales:

Son los signos que se usan para comparar dos valores, de los cuales si se cumple te va un resultado que puede ser **verdadero** ó **falso,** sé que lo has visto en estos caso:

Operador	Símbolo	Operación	Resultado
Mayor que	>	4 > 7	Falso
Menor que	<	9 < 1	Falso
Mayor igual que	>=	7 >= 2	Verdadero
Menor igual que	<=	8 <= 8	Verdadero
Igual	==	7 == 1	Falso

Operador	Símbolo	Operación	Resultado
Diferente	!=	3 != 9	Verdadero

3.3 Lógicos:

En matemáticas son conocidas como tabla proposicional en este oportunidad solo tomaremos los valores de (y), (o) y (negación). Sé que eres un génio en ese tema. Te muestro un ejemplo:

Donde F es falso y V es verdadero.

Operador "Y"	Salida	Operador "O"	Salida	Negacio "!"	Salida
V y V	V	V o V	V	!(V y F)	V
V y F	F	V o F	V	!(V y V)	F
F y V	F	F o V	V	!(F o V)	F
F y F	F	F o F	F	!(F o F)	V

3.4 Asignación:

Lo usamos en programación para asignar valor a una variable, como la siguiente manera:

Asignamos una variable de nombre "a" de tipo entero -> **int a**.

si queremos asignar un valor a la variables ponemos "=" -> **a = 5** -> el valor de **a** será 5.

Ahora si queremos restar un valor es decir **a - 1 = 4** (Recordemos que el valor de **a** es **5**), para asignar el nuevo valor a **"a"** es como poner -> **a = a -1**, como vemos que el valor de **a** se repite dos veces, se puede escribir de esta manera -> **a -= 1**, esto es lo mismo a -> **a = a -1** entonces el operador de **"-="** quita un valor a la variable asignada.

De igual manera para sumar un valor a la variables lo hacemos de esta manera -> a = a +1, esto nos da como resultado **6** (el valor de **a** es **5**), la oparación se puede escribir de la siguiente manera -> a += 1 que seria exáctamente como -> a = a + 1.

4 Entradas y Salidas

4.1 Inicio:

Para empezar a escribir un código es c++ nesecitas tener un lugar compilador o un lugar donde escribir código eso te lo dejo a tu elección.

Primeramente cada progrma de inicia con:

```
#include <iostream>
```

la palabra **#include <iostream>** esto incluye la libreria para poder poder la entrada y salida de valores los cuales son:

```
cin >> //esto se utiliza para entrar o digitar valores a tu programa.
cout << //esto se utiliza para mostrar mensaje en tu programa.
```

La estructura principal es de esta manera:

```
#include <iostream>
int main(){  //Esta es la función principal que leerá el programa


    return 0; //Si el programa funciona bien retornara 0
}
```

4.2 Salidas:

Para mostrar tu primer hola mundo en c++ se hace de la siguiente manera:

```
#include <iostream>
int main(){
    std::cout << "Hola mundo"<< std::endl;
    return 0;
}
```

Este fracmento de código imprimirá tu primir “Hola mundo”, recuerda que para enviar un mensaje es importante que este en comillas dobles **“ “** y si solo va a tener una palabra debe de estar en comillas simples **’ ’** .

Para recortar el código si se te hace estresante puedes poner el **using namespace std** lo que hace en manera simple es abreviar u omitir el **std::** el código quedará de la siguiente manera:

```
#include <iostream>
using namespace std;
int main(){
    cout << "Hola mundo"<<endl;
    return 0;
}
/* cabe recalcar que para proyectos grandes el "using namespace std" ara tu programa menos c
```

Los mensajes que pongo en los códigos no se lee puesto que son comentarios, si tu deseas poner un comentario lineal usa // y para poner mensajes multilineas debes de usar /**/ escribiendo el mensaje dentro de asteriscos.

4.3 Entradas:

Para guardar y mostrar valores para compilar el código hacemos esto:

```
#include <iostream>
using namespace std;
int main(){
    int valor;
    cout << "Digite un valor: ";
    cout << "El valor es: "<<valor<<endl;
    return 0;
}
```

Este código loq ue hace es declarar una variables de nombre **valor** de tipo entero (int) con el mensaje de “Digite un valor” luego en tu programa el usuario digitará un valor numérico este valor se almacenará en **“valor”** y para imprimir el valor introducido se usa un cout seguido de el nombre de la variable a mostrar, cabe recalcar que si no se asigna un valor a la variable imprimira un dato basura o que no tenga nada que ver.

```
Digite un valor: 15
El valor es: 15
```

Para hacer operaciones de hace de la siguiente manera:

```cpp
#include <iostream>
using namespace std;
int main(){
    int num1 = 8;
    int num2 = 4;
    int suma, multiplicacion;
    suma = num1 + num2;  //en este caso res será igual a 12.
    multiplicacion = num1 * num2;
    // el endl es como un salto de linea.
    cout << suma <<" || "<< multiplicacion<<endl;
    // tambien lo podemos poner de manera directa.
    cout << num1 - num2<<"  || "<< num1 / num2endl;
    return 0;
}
```

Hicimos las operaciones básicas que todos conocemos (+, -, *, /), el fragmento de código se mostrará de la siguiente manera:

```
12 || 32
4 || 2
```

Tienes que ser creativo para poder realizar cualquier tipo de operaciones aritméticas por ejemplo realizaremos un ejercicio simple que es resolver esta ecuación: La fórmula es: $a \times \left(\frac{b}{c}\right)$ empezemos.

```cpp
#include <iostream>
using namespace std;
int main(){
    float a,b,c;
    float resultado;
    cout<<"Digite 3 numeros separados por espacios o saltos de linea: ";
    cin>>a>>b>>c;
    //En este caso primero se ejecuta la divisón y luego la multiplicaión.
    resultado = a*(b/c);
    cout << "El resultado es: "<<resultado<<endl;
    return 0;
}
```

Lo que hicimos es declarar 4 variables de tipo flotante para no perder los decimales del resultado, tener en cuenta que **resultado** alamcena el resultado y lo imprimimos como se muestra acontinuación:

```
Digite 3 numeros separados por espacios o saltos de linea: 3 4 5
El resultado es: 12
```

Para finalizar es recomendable que tengas cuidado con tus operaciones que tu programa va a realizar.

5 Estructuras de Control

5.1 if-else

Una breve explicación sobre ese tema es que cuando la condición dentro de **if** sea verdadera se ejecuta una parte del código, en cambio si la afirmación es falsa se ejecuta el código dentro de **else** la estructura principal es la siguiente:

```
#include <iostream>
using namespace std;
int main(){
    if(condición){
        acción  // solo entrara en esta parte del código se la condición es verdadera
    }
    else{
        acción // Este código se ejecuta si la afirmación es falsa
    }
    return 0;
}
```

Un ejemplo básico para entender aún más la condicional **if-else**.

```
#include <iostream>
using namespace std;
int main(){
    if(3>4){
        cout << "3 es mayor que 4."<<endl;
    }
    else{
        cout << "4 es mayor que 3."<< endl;
    }
    return 0;
}
```

En este ejemplo vemos que la afirmación de **3>4** es falsa por lo que se ejecuta el código que esta dentro de **else** como se muestra a continuación.

4 es mayor que 3.

Un ejercicio muy practico es comprobar si un numero es par o impar. En este código utilizaremos el operador **mod** (%) el código quedara de la siguiente manera:

```
#include <iostream>
using namespace std;
int main(){
    int numero;
    cout << "Digite un numero entero: ";
    cin >> numero;
    if((numero % 2) == 0){
        cout << "El numero es par."<<endl;
    }
    else{
        cout << "El numero es impar."<<endl;
    }
    return 0;
}
```

En este ejercicio solicitamos al usuario que ingrese un número entero, el cual almacenamos en una variable de tipo entero llamada **numero**.

La lógica del programa se basa en la operación módulo (%), que calcula el **residuo** de una división:

```
flowchart LR
  A["Solicitar número (numero = entrada)"] --> B[numero = 4]
  B --> C{"numero % 2 == 0?"}
  C -->|Verdadero| D[Mostrar "El número es par"]
  C -->|Falso| E[Mostrar "El número es impar"]
```

para mayor entendimiento -> **4 % 2** <- el resultado nosda como resultado -> **0** <- por lo que la afirmación quedaría de la siguiente manera -> **0 == 0** <- y esta afirmación es verdadera e imprimiria el fragmento de código que esta dentro de **if**. Encanbio si la entrada fuera 5 esto pasaría -> **(5 % 2) == 0** -> **1 == 0** -> esta afirmación es falsa por loque se imprime el código dentro de **else**.

Digite un numero entero: 4
El numero es par.

Digite un numero entero: 5
El numero es impar.

5.2 switch-case:

De una manera fácil de entender, esta estructura lo que hace es verificar una serire de posibles valores es decir el usuario digite una dato y ese dato se va comparando con diferentes valores, la estructura es la siguiente:

```
#include <iostream>
using namespace std;
int main(){
    switch(valor){
        case dato1: // si el valor es igual a dato1 ejecuta la acción.
            acción
            break; /* una vez realizada la accion la palabra reservada "break" ara
                      salir de la sentencia switch*/
        case dato2:
            acción
            break;
        case dato3:
            acción
            break;
        default:    // si ninguno de los opciones se cumple se ejecuta la accion
            accion
            break;
    }
    return 0;
}
```

Como se vé es fácil de entender, como un ejemplo practico será realizar un progrma que dado una entrada decir si tiene que dar un exámen, que no tiene que dar un exámen o que digite un valor determinado:

```
#include <iostream>
using namespace std;
int main(){
    char opcion; // la opción solo podrá contener un string 'a', 'b', etc;
    cout << "Diste el examen? (s/n): ";
    cin >> opcion; // solo podras ingresa si-> s ó no->n
    switch(opcion){
        case 's':
            cout << "Espera tu resultado."<<endl;
            break;
        case 'n':
```

```
            cout << "Dirigete al salon A3."<<endl;
            break;
        default:
            cout << "Ingresaste un valor invalido."<<endl;
            break;
    }
    return 0;
}
```

Diste el examen? (s/n): s
Espera tu resultado.

Lo que hace el programa es verificar si el caracter ingresado es 's' encaso de ser a si lo que hace es imprimir **Espera tu resultado.**, en caso de ser 'n' imprime el código que esta por debajo es decir **Dirigete al salon A3.** y en caso de no ingresar ninguno de los dos se imprimirá **Ingresate un valor invalido.** Con tu creatividad sé que podras crear programas con más opciones como por ejemplo hacer una calculadora que dada una opción hacer una suma, una resta o lo que tu quieras.

5.3 Ternario:

El operador ternario es parecido al **if-else** pero de manera resumida es decir tiene esta estructura:

```
#include <iostream>
using namespace std;
int main(){
    (condicion) ? accion(si es verdadera) : acción si es falsa;
    return 0;
}
```

Si la condición es verdadera se ejecuta el código que esta despues de -> **?** si la condición es falsa se ejecuta el código que esta después de -> **:**. Ejemplo:

```
#include <iostream>
using namespace std;
int main(){
    int a=5, b=6;
    int resultado;
    resultado = (a>b) ? a:b;
    cout << "El numero mayor es: " << resultado<<endl;
    return 0;
}
```

Primeramente lo que hicimos es declarar 3 variables entereas como se muestra ojo el operador ternario retorna un valor de acuerdo asus parámetros, que este caso son **a** y **b** primeramente compara los valores de la siguiente manera: (a>b)? -> esta operación es como -> **5>6**, esta condición es falsa por lo que retorno el código que esta de lado derecho de -> **:** que es la **b**, dondo como resultado:

```
El numero mayor es: 6
```

Puedes probrar de diferente manera como en este otro ejemplo:

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

int main() {
    int a = 10, b = 20;
    string resultado;
    // Operador ternario
    resultado = (a > b) ? "Mayor." : "Menor o igual.";
    cout << resultado << endl;
    return 0;
}
```

En este caso retornamos una cadena de caracteres y como sabemos que para poder guardar cadenas usamos la palabra reservada **string** seguido del nombre de la varibale, como vemos en la salida dara **Menor o igual.** por que el valor de **a** es menor que el valor de la variable **b**

```
Menor o igual.
```

Puedes hacer una serie de ejercicio para tener más familiarización con las estructuras de control, para poder a si desarrollar tu lógica de programación.

6 Bucles o Ciclos

Los **bucles** son una forma eficiente de repetir una serie de instrucciones sin tener que escribirlas múltiples veces.

Un ejemplo básico sería imprimir cinco veces el mensaje **"Hola mundo"**.

En este ejercicio, aprenderemos a hacerlo utilizando los diferentes tipos de **bucles o ciclos** que ofrece C++.

6.1 for:

El bucle **for** tiene 3 parámetros las cuales son:

```
for(asignació; condición; iteración){
    acción se la condición es verdadera;
}
```

La tarea que pusimos al inicio lo vamos a resolver, quedará de la siguiente manera:

```
#include <iostream>
using namespace std;
int main(){
  // La asignación si se puede hacer dentro del bucle
    for(int i=0; i<5; i++){
        cout<< "Hola mundo."<<endl;
    }
    return 0;
}
```

Lo que hace el código es recorrer del 0 la 4, ¿Por qué desde el cero? por que la asignamos el valor de 0 en el inicio -> **int i=0;** ¿Por qué hasta 4? porque una vez llegado hasta el valor de **5<5** esto nos dara una condición falsa por lo que saldra de bucle, que hace el **i++** lo que hace es recorrer uno por uno cada interación es decir a **i** lo va a ir sumando **+1** hasta que la condición sea falsa, el código tendra la siguiente salida:

```
Hola mundo.
Hola mundo.
Hola mundo.
Hola mundo.
Hola mundo.
```

Para tener más información te envito a ver este video ->

6.2 while:

Este bucle al igual que el for primeramente compara su condición y si es verdad se ejecuta el código y si no se salta y sus partes son:

```
While(condición){
    acción a realizar si la condición es verdadera;
}
```

En esta parte tienes que tener cuidado porque si tienes un mínimo margen de error el bucle se ejecutará infinitamente. Vamos a realizar el ejercicio anterior de igual manera que la anterior:

```
#include <iostream>
using namespace std;
int main(){
    int i=0;
    while(i<5){
        cout << "Hola mundo."<<endl;
        i++; // esta parte es crucial ya que sin ella se crearía un bucle infinito
    }
    return 0;
}
```

Que es lo que esta pasando, primeramente declaramos una variables de tipo entera llamada **i** que tiene como valor el **0** en la condicón que se toma en cuenta es esta -> **(i<5)**, ya sabesmos que es verdadera y pasará a ejecutarse el código, esta parte del código es fundamental -> **i++** ¿Por qué? porque sin ella el valor de **i** seguirá siendo un **0** y asi hasta el infinito y más alla.

```
Hola mundo.
Hola mundo.
Hola mundo.
Hola mundo.
Hola mundo.
```

Para tener más información te envito a ver este video ->

6.3 do-while:

El bucle **do-while**, este bucle lo que hace primeramente es hacer la acción y luego verificar la condición y sus partes son estas:

```
do{
    acción
}While(condición);
```

Como ves primeramente hace una acción y luego lo compara, pero bueno vamos a realizar la tarea:

```
#include <iostream>
using namespace std;
int main(){
    int i = 0;
    do{
        cout << "Hola mundo."<<endl;
        i++; // sin esto el buclue se hace infinitamente
    }while(i<5);
    return 0;
}
```

Primeramente se ejecuta el código y recien va a la condición, esto es muy bueno por si quieres asegurarte almenos una ejecución en tu código, la salida es:

```
Hola mundo.
Hola mundo.
Hola mundo.
Hola mundo.
Hola mundo.
```

Para mayor entendimiento ver el siguienete video -> Como viste la mayor parte de los casoss de bucles funcionan con inicializaciones, pero no siempre es de esa manera ya que puede ser que solo se pare con un determinado caracter. Si te quedaste con la interrogante te envito a ver estos videos que te haran entender de una mejor manera estos tres tipos de bucles (en los videos se tomará en cuenta las palabras revervadas **break** y **continue**) ->

7 Arreglos o Vectores

Un **arreglo** (también llamado **vector** en programación básica) es una **estructura de datos** que permite almacenar múltiples valores **del mismo tipo** en una sola variable.

Cada valor dentro del arreglo se guarda en una **posición o índice**, comenzando desde el **índice 0**.

Los arreglos son útiles cuando se necesita **manejar colecciones de datos**, como listas de números, nombres o resultados, sin tener que declarar muchas variables por separado.

7.1 Vectores:

Como se mencionó anteriormente los vectores almacenan una cantidad de valores de un solo tipo estos son sus partes y un par de ejemplos:

```
// se tiene el tipo de dato entero -> int arreglo[5] es necesario que
tenga los corchetes y la cantidad de valores que vamos a utilizar.

    int arreglo[5] = {1,2,3,4,5};

// en este ejemplo lo asignamos de manera directa en el código.
```

Para poder mostrar el contenido de un arreglo puedes hacerlo con diferentes tipos de bucles, pero generalmente se utiliza el bucle **for**. Vamos a mostrar el código completo para mostrar el contenido del arreglo:

```
#include <iostream>
using namespace std;
int main(){
    int arreglo[5] = {1,2,3,4,5};
    for(int i=0; i<5; i++){
        cout << arreglo[i]<<" ";
    }
    return 0;
}
```

Recordemos lo siguiende, los arreglos tienen índices que inician desde 0 que esta caso sería:

```
tipo nombre[cantidad]      valores
int     arreglo[5]     = {1, 2, 3, 4, 5};

Los índices son ->        0, 1, 2, 3, 4
```

Entonces lo que recorre el bucle son los indices del vector que en el ejemplo el vector tiene de nombre **arreglo**, ahora mostraremos la salida estandar:

```
1 2 3 4 5
```

Para mayor entendimiento te envito a ver este video -> Hagamos un ejercicio de entrada y salida de arreglos de tipo entero (int):

```
#include <iostream>
using namespace std;
int main(){
    int arreglo[5];
    cout<<"Digite 5 numeros: ";
    for(int i=0; i<5; i++){
        cin>>arreglo[i];
    }
    cout<<"Mostrando el arreglo."<<endl;
    for(int i=0; i<5; i++){
        cout<<arreglo[i]<<" ";
    }
    return 0;
}
```

Lo que hicimos primeramente es asignar un arreglo como es en este caso -> **int arreglo[5]** que almacerá 5 valores enteros de tipo de entero, esta será la compilación exacta del código:

```
Digite 5 numeros: 1 2 3 4 5
Mostrando el arreglo.
1 2 3 4 5
```

Como se puede observar es es una manera fácil de de guardar datos y mostrarlos en un arregle, te envito a que puedas prácticar y desarrollar tu lógica de programación de manera constante.

7.2 Matrices:

La matriz es una estructura de datos en un plano bidimensional que tiene filas y columnas, como en los arreglos las matrices empiezan por los índces de [0][0], Para formar una matriz necesitamos bucles

anidados ¿Qué son? son unos bucles dentro de otro como en este ejemplo:

```
    for(int i=0; i<3; i++){
        for(int j=0; j<3; j++){
            cout<<[i][j]<< " ";
        }
        cout << endl;
    }
```

Esta es la manera en la que se imprime una matriz de tipo 3 por 3, como se puede observar tenemos dos bucles for uno adentro de otro. Para guardar e imprimer los datos se hacen de esta manera:

```
#include <iostream>
using namespace std;
int main(){
    int matriz[3][3];
    cout << "Digite un valor para la pariz: "<<endl;
    for(int i=0; i<3; i++){
        for(int j=0; j<3; j++){
            cout << "["<<i<<"]["<<j<<"]: ";
            cin >> matriz[i][j];
        }
    }
    cout << "Mostrando la matriz."<<endl;
    for(int i=0; i<3; i++){
        for(int j=0; j<3; j++){
            cout << matriz[i][j] <<" ";
        }
        cout << endl;
    }
    return 0;
}
```

Bien, en este pequeño ejercicio que hicimos es declarar una matriz de tipo 3 por 3 guardamos los datos y los mostramos en pantalla, el ejemplo nos quedará de la siguente manera:

```
Digite un valor para la pariz:
[0][0]: 1
[0][1]: 2
[0][2]: 3
[1][0]: 4
[1][1]: 5
[1][2]: 6
[2][0]: 7
[2][1]: 8
[2][2]: 9
Mostrando la matriz.
1 2 3
4 5 6
7 8 9
```

Esta es la manera en la que debes de guardar los datos en la matriz y mostrarlos. Si te quedaste con ganas de entender más a fondo te envito a que veas este video ->

8 Cadenas de Texto

8.1 char[n]:

Una manera de guardar cadenas de texto es usar en vector de tipo char, es una manera un poco eficiente, pero esta bien si tu programa no requiere de mucho texto en las entradas, se guarda de la siguiente manera:

```
#include <iostream>
using namespace std;
int main(){
    char nombre[100]; //cuardamos memoría para 100 carácteres
    cout << "Digite su nombre: ";
    cin.getline(nombre,100,'\n');
    cout << "Su nombre guardado es: "<<nombre<<endl;
    return 0;
}
```

En este caso, para poder guardar las cadenas de texto utilizamos **`cin.getline()`**, que en este caso usa 3 parámetros:
en el primero nos preguntamos ¿Dónde lo quieres guardar? (en este caso es en **nombre**),
en el segundo, ¿cuántos espacios tiene? (ahí ponemos la cantidad que reservamos en el inicio),
y por último, ¿cuándo queremos que termine? En ese lugar pondremos un salto de línea que es **`\n`**, es decir, que al apretar la tecla **Enter** se guardará la cantidad de caracteres antes de dicha acción.
Ejemplo de compilación:

```
Digite su nombre: Maria Gonzales Perez
Su nombre guardado es: Maria Gonzales Perez
```

Algunas carácteristicas especiales que puedes usar son las siguientes para que algunas de estas funciones correctamente es necesario usar **#include<cstdlib>** (donde var es una variables):

Funciones clásicas de manejo de `char[]` (cstring)

> Estas funciones trabajan con arreglos de caracteres (`char[]`), no con objetos `string`. Se requiere incluir la cabecera `#include <cstring>` y en algunos casos `#include <cstdlib>`.

Método	¿Para qué sirve?	Ejemplo	Resultado
`strlen(var)`	Obtener la longitud de una cadena	`int num = strlen("alex");`	`num = 4`
`strcpy(dest, src)`	Copiar el contenido de una cadena en otra	`strcpy(var1, "Alex");`	`var1 = "Alex"`
`strcmp(a, b)`	Comparar dos cadenas	`strcmp("hola", "hola") == 0`	`true` si son iguales
`strcat(a, b)`	Concatenar dos cadenas	`strcat(var1, "Alex");`	Si `var1 = "Hola "`, entonces `"Hola Alex"`
`strrev(var)`	Invertir una cadena	`char r[10] = strrev(var1);`	Si `var1 = "dia"`, entonces `"aid"`
`strupr(var)`	Convertir a mayúsculas	`strupr(var1);`	Si `var1 = "paz"`, será `"PAZ"`
`strlwr(var)`	Convertir a minúsculas	`strlwr(var1);`	Si `var1 = "PAZ"`, será `"paz"`
`strncmp(a, b, n)`	Comparar los primeros `n` caracteres de dos cadenas	`strncmp("alex", "a", 1) == 0`	`true` porque `'a' == 'a'`
`atoi(var)`	Convertir texto a entero	`int num = atoi("2");`	`num = 2`
`atof(var)`	Convertir texto a número con decimales	`float num = atof("2.3");`	`num = 2.3`

Nota: `strrev`, `strupr` y `strlwr` no son funciones estándar en C++, pero funcionan en algunos compiladores como **Dev-C++**.

Hagamos un ejercico simple que será verificar si una palabra tiene la 'a' minuscula en su nombre:

```
#include <iostream>
#include <cstring> // lo utilizamos para usar strlen()
using namespace std;
int main(){
    char nombre[20];
    bool bandera = false;
    int cantidad = 0;
    cout << "Digite su nombre: ";
    cin.getline(nombre, 20, '\n');
    cantidad = strlen(nombre);
    for(int i=0; i<cantidad; i++){
        if(nombre[i] == 'a'){
            bandera = true;
```

```
            break;
        }
    }
    if(bandera){
        cout << "Su nombre contiene almenos una a."<<endl;
    }else{
        cout << "Su nombre no contiene la vocal a."<<endl;
    }
    return 0;
}
```

Ente programa verifica si almenos tu nombre contiene una **a** usamos 3 variables de diferentes tipos que son **char[20]** para almacenar un nombre de máximo 20 carácteres, **bool** para verificar si hay almenos una **a** en el nombre y un **int** para saber exáctamente cuantos carácteres tiene el nombre introducido. La compilación del programa será de esta manera:

```
Digite su nombre: alex
Su nombre contiene almenos una a.
```

Si deseas ver más ejercicios de este tipo te invito a ver este video ->

8.2 string

Otra manera de guardar cadenas de texto es usar los string que una manera más fácil de alamacenar las palabras y se entien de mejor manera, a qui te muestro un ejemplo:

```
#include <iostream>
#include <string>  //Agregamos esta librería para poder usar las strings
using namespace std;
int main(){
    string nombre;
    cout << "Digite su nombre: ";
    getline(cin, nombre); // de esta manera es como se guarda los strings
    cout << "Su nombre guardado es: "<<nombre<<endl;
    return 0;
}
```

Para guardar los string se usa la palabra reservada **getline()** que en este caso usamos dos parámetros, para saber que poner en el primer parámetro nos preguntamos ¿Qué queremos hacer? En este caso queremos guardar y por ello ponemos **cin** en el segundo parámetro va ¿Dónde lo quieres guardar? por esa razón va el nombre de la variable de tipo string. Algunas carácteristicas básicas de los string son estos:

Métodos comunes de `string` en C++

MÉTODO	¿Para qué sirve?	Ejemplo	Resultado
`.empty()`	Verifica si la cadena está vacía	`string s = ""; s.empty();`	`true`
`.length()`	Devuelve el número de caracteres de la cadena	`string s = "Hola"; s.length();`	`4`
`.size()`	Igual que `.length()`	`s.size();`	`4`
`.substr(pos, n)`	Devuelve una subcadena desde `pos` con `n` caracteres	`s.substr(1, 2);`	`"ol"`
`.find()`	Devuelve la posición de una subcadena o carácter	`s.find("la");`	`2`
`.append(str)`	Añade `str` al final de la cadena	`s.append(" mundo");`	`"Hola mundo"`
`.insert(pos, s)`	Inserta una subcadena `s` en la posición `pos`	`s.insert(2, "xx");`	`"Hoxxla"`
`.erase(pos, n)`	Elimina `n` caracteres desde la posición `pos`	`s.erase(1, 2);`	`"Ha"` (de "Hola")
`.replace(pos,n,s)`	Reemplaza `n` caracteres desde `pos` por `s`	`s.replace(0, 4, "Hi");`	`"Hi"`
`toupper(c)`	Convierte un carácter a mayúscula	`toupper('a');`	`'A'`
`tolower(c)`	Convierte un carácter a minúscula	`tolower('A');`	`'a'`
`isalnum(c)`	Verifica si un carácter es alfanumérico	`isalnum('3');`	`true`
`isalpha(c)`	Verifica si un carácter es una letra	`isalpha('A');`	`true`
`isdigit(c)`	Verifica si un carácter es un dígito	`isdigit('8');`	`true`

> **Nota:** funciones como `toupper()`, `isdigit()`, etc., requieren incluir la librería `#include <cctype>`.

Como modo de práctica hagámos un programa que verifique si la primera y última palabar inician con un caracter, si es haci que imprima "hola muno" y si no mostrar "buenas noches".

```
#include <iostream>
#include <cctype> // para usar isalpha()
#include <cstring>
using namespace std;
int main(){
    string palabra;
    int tam = 0;
    cout << "Digite una palabra: ";
```

```
    getline(cin, palabra);
    tam = palabra.length();
    if(isalpha(palabra[0]) && isalpha(palabra[tam-1])){
        cout << "hola mundo."<<endl;
    }else{
        cout << "buenas noches."<<endl;
    }
    return 0;
}
```

En este lo que hicimos es verificar si una palabra empieza y termina con un caracter, como en los vectores las string tienen índices e inician de igual manera pos el 0. Por ese motivo en la estructura de control **if** pusimos `palabra[0]` y `palabra[tam-1]` si pusiera la palabra **casa** lo que la condicional verificaría sería esto: `c y a` ¿Son string? si, y por ello el resultado sería **true** e imprimiría "hola mundo" y en caso contrario "buenas noches", a qui te muestro un ejemplo de compilación:

```
Digite una palabra: casa
hola mundo.
---> caso contrario <---
Digite una palabra: casa7
buenas noches.
```

Si te quedaste con intriga y deseas saber más te invito a que le des un vistaso a este video ->

9 Estructuras

Las estructuras o struct en C++ son una forma de definir un tipo de dato personalizado que puede contener varios elementos de diferentes tipos. Una estructura es similar a una clase, pero se utiliza principalmente para almacenar datos y no para definir comportamientos.

9.1 Definición de estructura:

Para definir una estructura en c++, se utiliza la palabra reservada `struct` seguida del nombre de la estructura y los elementos que la componen.

```
struct Persona{
    string nombre;
    int edad;
    string direccion;
};
```

Uso de una estructura: Una vez definida la estructura, puedes crear variables de este tipo y acceder a sus parámetros.

```
  Persona persona; //creamos una variables de la estructura "Persona"
  persona.nombre = "juan";
  persona.edad = 15;
  persona.direccion = "calle 13";
```

Para acceder a las varibales de la estructura primeramente se pone las variables que definimos anteriormente seguida de un punto y en que valor de la estructura la deseamos guardar, tal y cual esta en el ejemplo de arriba. **Ventajas de las estructuras:** Las estructuras son útiles cuando necesitas almacenar varios datos relacionados en una sola unidad. Algunas ventajas de las estructuras son:

- Permiten organizar y estructurar los datos de manera lógica.
- Facilitan el acceso y la manipulación de los datos.
- Pueden ser utilizados como parámetros de funciones o como valores de retorno. Completemos el programa, pero ahora que el usuario completo los direferentes campos.

```cpp
#include <iostream>
#include <cstring>
using namespace std;
struct Persona{
    string nombre;
    int edad;
    string direccion;
}persona; // la variable lo podemos colocar aqui.
int main(){
    cout << "Digite su nombre: ";
    getline(cin, persona.nombre);
    cout << "Digite su edad: ";
    cin >> persona.edad;
    fflush(stdin); // lo usamos para vaciar el buffer.
    cout << "Digite su direccion: ";
    getline(cin, persona.direccion);
    cout<<"\nMostrando los datos de la persona."<<endl;
    cout << "El nombre es "<<persona.nombre<<endl;
    cout << "La edad es "<<persona.edad<<endl;
    cout << "La direccion es "<<persona.direccion<<endl;
    return 0;
}
```

En este caso guardamos y mostramos los datos que el usuario nos digita, el resultado nos quedaría de la siguiente manera:

```
Digite su nombre: alex
Digite su edad: 15
Digite su direccion: calle 13

Mostrando los datos de la persona.
El nombre es alex
La edad es 15
La direccion es calle 13
```

9.2 Estructuras anidadas:

Las estructuras anidadas en C++ se refieren a la capacidad de definir una estructura dentro de otra estructura. Esto permite crear estructuras más complejas y jerárquicas, donde una estructura puede contener otra estructura como uno de sus miembros. **Ejemplo de estructura anidada**

```
struct Direccion{
    string calle;
    string ciudad;
    string pais;
};
struct Persona{
    string nombre;
    int edad;
    Direccion direccion;
}persona; //declaramos la variable.
```

En este ejemplo, la estructura `Persona` contiene esa estructura `Direccion` como uno de sus miembros. Puedes acceder a los miembros de la estructura anidada utilizando el operador de punto `(.)`. **ventajas de las estructuras anidadas** Las estructuras anidadas son útiles cuando necesitas representar datos complejos y jerárquicos. Algunas ventajas de las estructuras anidadas son: - Permiten crear estructuras de datos más complejas y realistas. - Facilitan la organización y el acceso a los datos. - Pueden mejorar la legibilidad y la mantenibilidad del código. **Aplicaciones de las estructuras anidadas** Las estructuras anidadas se pueden utilizar en una variedad de aplicaciones, como:

- Representar datos de personas, como direcciones y contactos.
- Modelar estructuras de datos complejas, como árboles o grafos.
- Crear estructuras de datos para juegos o simulaciones.

Vamos a completar el código e igualmente pediremos al usuario que lo rellene:

```
#include <iostream>
#include <cstring>
using namespace std;
struct Direccion{
    string calle;
    string ciudad;
    string pais;
};
struct Persona{
    string nombre;
    Direccion direccion; //con estos parámetros designamos a la
                         //estructura anidada
    int edad;
}persona;
int main(){
    cout << "Digite su nombre: ";
    getline(cin, persona.nombre);
```

```
    cout << "Digite su calle: ";
    getline(cin, persona.direccion.calle);
    cout << "Digite su ciudad: ";
    getline(cin, persona.direccion.ciudad);
    cout << "Digite su pais: ";
    getline(cin, persona.direccion.pais);
    cout << "Digite su edad: ";
    cin >> persona.edad;
    cout<<"\nMostrando los datos de la persona\n";
    cout << "Nombre "<<persona.nombre<<endl;
    cout << "Calle " <<persona.direccion.calle<<endl;
    cout << "Ciudad "<<persona.direccion.ciudad<<endl;
    cout << "Pais " <<persona.direccion.pais<<endl;
    cout << "Edad " <<persona.edad<<endl;
}
```

Lo que hicimos es completar el ejemplo que pusimos en el inicio. La salida estandar nos quedaría de esta manera:

```
Digite su nombre: alex
Digite su calle: calle 13
Digite su ciudad: puno
Digite su pais: peru
Digite su edad: 15

Mostrando los datos de la persona
Nombre alex
Calle calle 13
Ciudad puno
Pais peru
Edad 15
```

Si te quedaste con ganas de más te invito a que veas este video donde aremos más ejercicios para que entiendas un poco más ->

10 Funciones

En pocas palabras las funciones consiste en dividir tareas para que este más legible y como buena práctica de programación es fundamental aprendertela y para ello veremos una structura básica:

```
En el tipo de la función, si se requiere retornar un tipo de valor en espesífico
puedes
usar (int, float, bool) que son las más comunes.

Existen funciones que no retornan nada y el más comun es el "void".

La manera en la que creamos la funcion puede y no puede tener valores es decir:
int sumar(int n, int m) //En este caso la función recebe dos parámetros de tipo
                        //entero.
void pedir() // Y en esta otra la función no recibe parámetros.
```

Si te diste cuenta en este libro mayormente verás que usaremos mayormente la función de tipo `void` para pedir datos en nuestro programas, y cualquier otro tipo de función para realizar una tarea en específica, recuerda que en los parámetros puede ir el nombre que tu quieras, pero ten un cuenta que debes de espesificar muy bien el tipo de valor que tomará. Un aspecto de buena práctica es prototipar la función y despues de la función principal que es el **main** realizamos el código que realizará esa función en espesífica. Vamos a realizar un programa que retorme la suma de numeros enteros.

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

int suma(int, int); // este será nuestro prototipo de nueestra función

int main(){
    int num1, num2;
    int resultado; // como la funcion retorna un valor, ese valor lo guardamos en
                   // esta variables.
    cout << "Digite el primer numero: ";
    cin >> num1;
    cout << "Digite el segundo numero: ";
    cin >> num2;
```

```
    resultado = suma(num1, num2); // llamamos a la funcion poniendo el nombre de la
                                  // funcion seguida de los dos parámetros.
    cout << "El resultado es: "<<resultado<<endl; // finalmente lo imrpimimos
    return 0;
}
//realizamos el cuerpo de la funcion.
int suma(int a, int b){ // a y b son variables.
    int resultado;
    resultado = a + b;
    return resultado;   // la palabra "return" es como decir vota el resultado.
}
```

El programa es básico que lo que hace es una simple suma, pero para programas más grandes en fundamental hacerlo con funciones y en todo caso es necesario que lo aprendas como una buena práctica.

10.1 Paso de Parámetros

Para pasar arreglos, matrices, cadenas de texto se requiere de otro tipo de parámetros de que son:

```
int arreglo(int a[], int tamanio);
//El a[] es para pasar el arreglo y para saber la cantidad ponemos el "tamanio".

int matriz(int a[][100], int filas, int columnas);
//En este caso en necesario poner el tamaño máximo de columnas, seguida del total
// de filas y columnas

int cadenas(string nombre);
// para mandar de parámetro cadenas de texto.

int caracteres(char n);
//para poder mandar vocales y carácteres únicos.
```

Como ves estas son los parámetros que debes de aprender, pero para serlo más interesante vamos a crear un programa que haga la acción de una serie de opciones, lo vamos a realizar solo con funciones para que asi retroalimentes más tu conocimiento. Las opciones que habran en el `MENU` seran las siguientes:

```
------------>MENU<------------
1. OPERACIONES ARITMÉTICAS.
2. OPERACIONES CON ARREGLOS.
3. OPERACIONES CON MATRICES.
4. JUEGO DE "ADIVINA EL NUMERO".
5. SALIR.
```

Cabe racalacar que cada tema será de una manera casi profesional, porque también tengo mucho que aprender, pero por el motivo por el cual hago esto es para que no tengas complicaciones futuras y que lo entiendas de la mejor manera. haciendo esto aprenderas conceptos fundamentales.

10.2 Programa N°1

Para poder realizar este programa es fundamental retroceder y ver los temas anteriores de manera detenida para que luego se que haga fácil entender este pequeño programa.