Unidad 3: Entrada/Salida y Estructuras de Control (con detalles ampliados)

3.0 Entrada y Salida

Entrada de datos:

1. **cin**: Se utiliza para capturar datos desde el teclado. Es parte de la librería <iostream> y utiliza el operador de extracción (>>).
    - **Sintaxis**:

```
cin >> variable;
```

Puedes capturar varios valores en una sola línea:

```
cin >> variable1 >> variable2;
```

    - **Ejemplo detallado**:

```
int edad;
cout << "Introduce tu edad: ";
cin >> edad;  // Captura la edad ingresada por el usuario
cout << "Tu edad es: " << edad << endl;  // Muestra la edad
```

2. **gets**: Captura una cadena de caracteres con espacios desde el teclado. Se debe incluir la librería <cstdio> para su uso.
    - **Sintaxis**:

```
gets(cadena);
```

    - **Ejemplo detallado**:

```
char nombre[50];
cout << "Introduce tu nombre: ";
gets(nombre);  // Captura toda la línea de texto ingresada
cout << "Hola, " << nombre << endl;
```

3. **getchar** y **getch**: Capturan un solo carácter. getchar es parte de <cstdio>, mientras que getch es de <conio.h>.
    - **getchar()** Sintaxis:

variable = getchar();

- **getch()** Sintaxis:

variable = getch();

- **Ejemplo detallado**:

```
char letra;
cout << "Presiona una tecla: ";
letra = getchar();  // Captura el primer carácter ingresado
cout << "Tecla presionada: " << letra << endl;
```

4. **scanf**: Permite leer datos con formato específico, más flexible que cin. Se encuentra en la librería <cstdio>.
    - **Sintaxis**:

scanf(formato, &variable);

- **Ejemplo detallado**:

```
float altura;
printf("Introduce tu altura en metros: ");
scanf("%f", &altura);  // Lee un número en punto flotante
printf("Tu altura es: %.2f metros\n", altura);  // Imprime con 2 decimales
```

Salida de datos:

1. **cout**: Utilizado para mostrar datos en pantalla. Es parte de <iostream> y utiliza el operador de inserción (<<).

- **Sintaxis**:

```
cout << valor;
```

- **Ejemplo detallado**:

```
cout << "Hola, Mundo!" << endl;  // Muestra "Hola, Mundo!"
```

2. **puts**: Imprime una cadena de texto y añade un salto de línea automático. Se encuentra en <cstdio>.
   - **Sintaxis**:

```
puts(cadena);
```

   - **Ejemplo detallado**:

```
char mensaje[] = "Hola, bienvenido!";
puts(mensaje);  // Imprime el mensaje con salto de línea
```

3. **putchar**: Imprime un solo carácter.
   - **Sintaxis**:

```
putchar(caracter);
```

   - **Ejemplo detallado**:

```
putchar('A');  // Imprime 'A'
putchar('\n');  // Imprime un salto de línea
```

4. **printf**: Imprime datos con formato, más flexible que cout.
    - **Sintaxis**:

printf(formato, variable);

    - **Ejemplo detallado**:

```
int edad = 20;
printf("Tu edad es: %d\n", edad);  // %d es para enteros
```

3.1 Estructuras Secuenciales

Las instrucciones se ejecutan una tras otra en el orden en que aparecen.

- **Ejemplo detallado**:

```
int num1, num2, suma;
cout << "Introduce dos números: ";
cin >> num1 >> num2;  // Captura dos números
suma = num1 + num2;   // Suma los dos números
cout << "La suma es: " << suma << endl;  // Muestra el resultado
```

3.2 Estructuras Selectivas

Permiten tomar decisiones dentro del flujo del programa, evaluando condiciones.

Selección Simple (if):

- **Sintaxis**:

```
if (condición) {
```

```
   // Código a ejecutar si la condición es verdadera
}
```

- **Ejemplo detallado**:

```
int edad;
cout << "Introduce tu edad: ";
cin >> edad;
if (edad >= 18) {
   cout << "Eres mayor de edad." << endl;  // Se ejecuta si la condición es verdadera
}
```

Selección Doble (if-else):

- **Sintaxis**:

```
if (condición) {
   // Código si la condición es verdadera
} else {
   // Código si la condición es falsa
}
```

- **Ejemplo detallado**:

```
if (edad >= 18) {
   cout << "Eres mayor de edad." << endl;
} else {
   cout << "Eres menor de edad." << endl;
}
```

If Anidados:

Los **if anidados** permiten hacer evaluaciones más complejas al introducir varios bloques if dentro de otros.

- **Sintaxis**:

```
if (condición1) {
   if (condición2) {
      // Código a ejecutar si ambas condiciones son verdaderas
   } else {
      // Código a ejecutar si condición2 es falsa
   }
} else {
   // Código a ejecutar si condición1 es falsa
}
```

- **Ejemplo detallado**: Supongamos que estamos verificando la edad de una persona y si tiene un permiso especial:

```
int edad;
bool tienePermiso;

cout << "Introduce tu edad: ";
cin >> edad;

cout << "¿Tienes permiso especial? (1 = Sí, 0 = No): ";
cin >> tienePermiso;

if (edad >= 18) {
   if (tienePermiso) {
      cout << "Puedes entrar con permiso especial." << endl;
```

```
    } else {
        cout << "Puedes entrar." << endl;
    }
} else {
    if (tienePermiso) {
        cout << "Puedes entrar con permiso especial a pesar de ser menor de edad." << endl;
    } else {
        cout << "No puedes entrar porque eres menor de edad y no tienes permiso." << endl;
    }
}
```

En este ejemplo, el programa evalúa primero si la persona tiene 18 años o más. Si no es mayor de edad, verifica si tiene un permiso especial para tomar una decisión final.

Selección Múltiple (switch):

- **Sintaxis**:

```
switch (variable) {
    case valor1:
        // Código si variable es igual a valor1
        break;
    case valor2:
        // Código si variable es igual a valor2
        break;
    default:
        // Código si ninguno de los casos anteriores se cumple
        break;
}
```

- **Ejemplo detallado**:

```
int opcion;
cout << "Elige una opción (1-3): ";
cin >> opcion;
switch (opcion) {
   case 1: cout << "Opción 1 seleccionada." << endl; break;
   case 2: cout << "Opción 2 seleccionada." << endl; break;
   case 3: cout << "Opción 3 seleccionada." << endl; break;
   default: cout << "Opción no válida." << endl; break;
}
```

Operador Ternario (?:):

- **Sintaxis**:

```
variable = (condición) ? valor1 : valor2;
```

- **Ejemplo detallado**:

```
int numero = 10;
cout << (numero % 2 == 0 ? "Par" : "Impar") << endl;
```

3.3 Estructuras Iterativas

Permiten repetir bloques de código mientras una condición se cumple.

Ciclo while:

- **Sintaxis**:

```
while (condición) {
```

```
   // Código a ejecutar mientras la condición sea verdadera
}
```

- **Ejemplo detallado**:

```
int i = 0;
while (i < 5) {
   cout << "i vale: " << i << endl;
   i++;  // Incrementa i en cada iteración
}
```

Ciclo do-while:

- **Sintaxis**:

```
do {
   // Código a ejecutar
} while (condición);
```

- **Ejemplo detallado**:

```
int i = 0;
do {
   cout << "i vale: " << i << endl;
   i++;
} while (i < 5);
```

Ciclo for:

- **Sintaxis**:

```
for (inicialización; condición; incremento) {
   // Código a ejecutar
}
```

- **Ejemplo detallado**:

```
for (int i = 0; i < 5; i++) {
   cout << "i vale: " << i << endl;
}
```

Instrucciones para Limpiar Pantalla y Posicionar el Cursor

Limpiar la pantalla

```
system("cls");  // Limpiar pantalla en Windows
```

- **Instrucción en Linux/macOS:**

```
system("clear");  // Limpiar
```

Posicionamiento del Cursor: gotoxy()

Para posicionar el cursor en las coordenadas (x, y) de la consola. Esta función no es estándar en C++, pero podemos replicarla utilizando funciones de bibliotecas adicionales, como <windows.h> en sistemas operativos Windows.

Implementación de gotoxy() en Windows:

1. **Incluir la librería <windows.h>**.
2. Definir una función gotoxy() que utilice SetConsoleCursorPosition().

- **Código detallado**:

```
#include <iostream>
#include <windows.h>  // Biblioteca necesaria para gotoxy() en Windows
```

```
// Definir la función gotoxy para posicionar el cursor
void gotoxy(int x, int y) {
    COORD coord;
    coord.X = x;  // Columna (posición horizontal)
    coord.Y = y;  // Fila (posición vertical)
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), coord);
}

int main() {
    // Limpiar pantalla antes de posicionar el cursor
    system("cls");

    // Mover el cursor a la posición (10, 5)
    gotoxy(10, 5);
    std::cout << "Hola desde (10, 5)" << std::endl;

    // Mover el cursor a otra posición
    gotoxy(20, 10);
    std::cout << "Otra posición: (20, 10)" << std::endl;

    return 0;
}
```

- **Explicación**:
    - La función gotoxy(int x, int y) utiliza la estructura COORD para establecer las coordenadas X (columna) e Y (fila) del cursor en la consola.
    - SetConsoleCursorPosition() es la función de Windows que mueve el cursor a la posición especificada.
    - **Nota**: La numeración de las coordenadas en la consola empieza desde (0, 0), que es la esquina superior izquierda.

Limpieza de Pantalla:

Puedes limpiar la pantalla con la función:

```
system("cls");  // Limpiar pantalla en Windows
```

Ejemplo Combinado: Limpieza de Pantalla y Posicionamiento del Cursor

```
#include <iostream>
#include <windows.h>

void gotoxy(int x, int y) {
   COORD coord;
   coord.X = x;
   coord.Y = y;
   SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), coord);
}

int main() {
   // Limpiar la pantalla
   system("cls");

   // Mover el cursor y mostrar texto
   gotoxy(10, 5);
   std::cout << "Este texto aparece en la posición (10, 5)" << std::endl;

   gotoxy(20, 10);
   std::cout << "Este texto aparece en la posición (20, 10)" << std::endl;
```

```
    return 0;
}
```

Este código limpia la pantalla y posiciona el texto en diferentes coordenadas de la consola, utilizando la función gotoxy() que definimos.