Compendio de Ejercicios de Programación en C++

Oscar Solís - Servicio Social: Apoyo a la dociencia y asesoría académica Julio 2025

Introducción

Este compendio fue desarrollado como parte del servicio social en Apoyo a la docencia y asesoría académica en la asignatura **Programación** lenguaje C++, con el objetivo de apoyar a estudiantes en el desarrollo de habilidades lógicas, estructuración de algoritmos y buenas prácticas en la codificación.

Cada ejercicio incluye una descripción clara del problema, su solución comentada y una explicación de la lógica empleada. Los ejercicios están organizados por nivel de dificultad y temática, abordando temas fundamentales como manejo de variables, estructuras condicionales, ciclos, funciones, arreglos, punteros, clases y herencia.

Problema 1: Hola mundo

Temática: Estructura básica de un programa

Descripción: Escribe un programa en C++ que imprima en pantalla el mensaje '"Hola, mundo"'.

Este ejercicio permite familiarizarse con la sintaxis básica de C++, la estructura de un programa y cómo compilarlo y ejecutarlo.

Listing 1: HolaMundo.cpp

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

int main() {
    cout << "Hola, mundo" << endl;
    return 0;
}
```

Explicación:

- `#include <iostream>`: Permite usar `cout` para imprimir
- `using namespace std;`: Evita escribir `std::` antes de `cout`
- `int main()`: Función obligatoria donde inicia el programa

- `cout «`: Imprime el mensaje en pantalla
- `« endl;`: Agrega un salto de línea
- `return 0;`: Indica que el programa terminó correctamente

Problema 2: Conversión de Temperaturas

Temática: Uso de variables, entrada/salida, operaciones aritméticas

Descripción: Diseña un programa en C++ que reciba una temperatura en grados Fahrenheit e imprima su equivalente en grados Celsius, Kelvin y Rankine. El objetivo es aplicar fórmulas matemáticas básicas y practicar el manejo de entrada/salida y variables tipo 'float' o 'double'.

Listing 2: ConvertirTemperaturas.cpp

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

int main() {
    // Declaracion de variables
    float fahrenheit, celsius, kelvin, rankine;

    // Solicitar valor en Fahrenheit
    cout << "Ingresa la temperatura en Fahrenheit: ";
    cin >> fahrenheit;

    // Calculos
    celsius = (fahrenheit - 32) * (5.0 / 9.0);
    kelvin = celsius + 273.15;
    rankine = fahrenheit + 459.67;

    // Resultados
    cout << "Celsius: " << celsius << endl;
    cout << "Kelvin: " << kelvin << endl;
    cout << "Rankine: " << rankine << endl;

    return 0;
}
```

Explicación:

- `float fahrenheit, celsius, kelvin, rankine;` Declaración de variables tipo flotante para manejar decimales.
- `cin » fahrenheit;` Captura la temperatura que el usuario ingresa.

- `celsius = (fahrenheit - 32) * (5.0 / 9.0);`: Fórmula para convertir de Fahrenheit a Celsius.
- `kelvin = celsius + 273.15;` Convierte de Celsius a Kelvin.
- `rankine = fahrenheit + 459.67;` Fórmula para convertir directamente de Fahrenheit a Rankine.
- `cout « ...`: Muestra los resultados en pantalla.

Problema 3: Volumen de un Cilindro

Temática: Uso de constantes, entrada/salida, operaciones con potencias

Descripción: Crea un programa en C++ que calcule el volumen de un cilindro a partir del radio y la altura proporcionados por el usuario. Este ejercicio permite practicar el uso de constantes, operaciones aritméticas, entrada/salida de datos y la función de potencia 'pow'.

Listing 3: VolumenCilindro.cpp

```cpp
#include <iostream>
#include <cmath> // Para usar pow
using namespace std;

int main() {
    // Declaraci n de constante y variables
    const double PI = 3.1415926536;
    double radio, altura, volumen;

    // Solicitar datos al usuario
    cout << "Ingresa el radio del cilindro: ";
    cin >> radio;

    cout << "Ingresa la altura del cilindro: ";
    cin >> altura;

    // C lculo del volumen
    volumen = PI * pow(radio, 2) * altura;

    // Mostrar resultado
    cout << "El volumen del cilindro es: " << volumen << endl;

    return 0;
}
```

Explicación:

- `#include <cmath>` permite utilizar funciones matemáticas como pow

- `const double PI = 3.1415926536` define una constante para con mayor precisión
- `double radio, altura, volumen` declara las variables reales necesarias
- `cin » radio;` y `cin » altura;` capturan los valores ingresados por el usuario
- `volumen = PI * pow(radio, 2) * altura` aplica la fórmula del volumen del cilindro
- `cout « volumen` muestra el resultado en consola

Problema 4: Cálculo de ángulo restante

Temática: Entrada/Salida básica y operaciones aritméticas

Descripción: Escribe un programa en C++ que calcule el tercer ángulo de un triángulo cuando se conocen dos de sus ángulos. El programa debe solicitar al usuario los valores de dos ángulos y calcular el tercero usando la propiedad de que la suma de los ángulos internos de un triángulo es 180°.

Listing 4: AnguloTriangulo.cpp

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

int main() {
    // Declaracion de variables
    float A, B, C;

    // Solicitar y leer los angulos
    cout << "Ingrese el primer angulo (A): ";
    cin >> A;
    cout << "Ingrese el segundo angulo (B): ";
    cin >> B;

    // Calcular el tercer  ngulo
    C = 180 - (A + B);

    // Mostrar el resultado
    cout << "El tercer angulo mide: " << C << " grados" << endl;

    return 0;
}
```

Explicación:

- `#include <iostream>`: Necesario para las operaciones de entrada/salida

- `using namespace std;`: Permite usar `cout` y `cin` sin prefijo
- `float A, B, C;`: Declara tres variables para almacenar los ángulos
- `cin » A;`: Lee el valor del primer ángulo ingresado por el usuario
- `cin » B;`: Lee el valor del segundo ángulo ingresado por el usuario
- `C = 180 - (A + B);`: Fórmula para calcular el ángulo restante
- `cout « "El tercer angulo...`: Muestra el resultado formateado
- `return 0;`: Indica que el programa terminó correctamente

Nota: Este programa asume que los ángulos ingresados son válidos (A + B < 180°). Sería buena práctica agregar validación de entrada.

Problema 5: Cálculo del Cateto B

Temática: Operaciones con potencias y raíz cuadrada

Descripción: Escribe un programa en C++ que calcule el valor del cateto B en un triángulo rectángulo usando el teorema de Pitágoras, a partir de los valores del cateto A y la hipotenusa C. Este ejercicio fortalece el uso de operaciones matemáticas con potencias y raíces.

Listing 5: CalcularCatetoB.cpp

```cpp
#include <iostream>
#include <cmath> // Para pow y sqrt
using namespace std;

int main() {
    // Declaraci n de variables
    double a, b, c;

    // Solicitar datos al usuario
    cout << "Ingresa el valor del cateto A: ";
    cin >> a;

    cout << "Ingresa el valor de la hipotenusa C: ";
    cin >> c;

    // C lculo del cateto B
    b = sqrt(pow(c, 2) - pow(a, 2));

    // Mostrar resultado
    cout << "El valor del cateto B es: " << b << endl;

```

```
22      return 0;
23  }
```

Explicación:

- `#include <cmath>` permite el uso de funciones como pow y `sqrt`
- `double a, b, c` declara las variables reales necesarias
- `cin » a;` y `cin » c;` capturan los valores ingresados por el usuario
- `pow(c, 2)` calcula el cuadrado de la hipotenusa
- `pow(a, 2)` calcula el cuadrado del cateto A
- `sqrt(...)` obtiene la raíz cuadrada de la diferencia de cuadrados
- `cout « b` muestra el resultado en consola

Problema 6: Clasificación de Ángulos

Temática: Condicionales anidadas (‘if‘, ‘else if‘, ‘else‘)

Descripción: Crea un programa en C++ que lea dos ángulos y determine si son iguales o diferentes. Si son iguales, deberá indicar el tipo de ángulo (agudo, recto, obtuso, llano o cóncavo). Si son diferentes, el programa debe clasificar cada uno por separado. Este ejercicio permite practicar el uso de condicionales múltiples y anidadas.

Listing 6: ClasificacionAngulos.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5      // Declaraci n de variables
6      double anguloA, anguloB;
7
8      // Solicitar  ngulos  al usuario
9      cout << "Ingresa el  ngulo  A: ";
10     cin >> anguloA;
11
12     cout << "Ingresa el  ngulo  B: ";
13     cin >> anguloB;
14
15     // Comparaci n de los  ngulos
16     if (anguloA == anguloB) {
17         cout << "LOS  NGULOS  SON IGUALES" << endl;
18
```

```
19          if (anguloA < 90) {
20              cout << "SON AGUDOS" << endl;
21          }
22          else if (anguloA == 90) {
23              cout << "SON RECTOS" << endl;
24          }
25          else if (anguloA > 90 && anguloA < 180) {
26              cout << "SON OBTUSOS" << endl;
27          }
28          else if (anguloA == 180) {
29              cout << "SON LLANOS" << endl;
30          }
31          else if (anguloA > 180 && anguloA < 360) {
32              cout << "SON C NCAVOS" << endl;
33          }
34
35      } else {
36          cout << "LOS  NGULOS  SON DIFERENTES" << endl;
37
38          // Clasificaci n del  ngulo  A
39          if (anguloA < 90) {
40              cout << "EL  NGULO  A ES AGUDO" << endl;
41          }
42          else if (anguloA == 90) {
43              cout << "EL  NGULO  A ES RECTO" << endl;
44          }
45          else if (anguloA > 90 && anguloA < 180) {
46              cout << "EL  NGULO  A ES OBTUSO" << endl;
47          }
48          else if (anguloA == 180) {
49              cout << "EL  NGULO  A ES LLANO" << endl;
50          }
51          else if (anguloA > 180 && anguloA < 360) {
52              cout << "EL  NGULO  A ES C NCAVO" << endl;
53          }
54
55          // Clasificaci n del  ngulo  B
56          if (anguloB < 90) {
57              cout << "EL  NGULO  B ES AGUDO" << endl;
58          }
59          else if (anguloB == 90) {
60              cout << "EL  NGULO  B ES RECTO" << endl;
61          }
62          else if (anguloB > 90 && anguloB < 180) {
63              cout << "EL  NGULO  B ES OBTUSO" << endl;
64          }
65          else if (anguloB == 180) {
```

```
66              cout << "EL  NGULO  B ES LLANO" << endl;
67          }
68          else if (anguloB > 180 && anguloB < 360) {
69              cout << "EL  NGULO  B ES C NCAVO" << endl;
70          }
71      }
72
73      return 0;
74  }
```

Explicación:

- `double anguloA, anguloB` declara las variables para almacenar los valores de los ángulos
- `cin » anguloA` y `cin » anguloB` permiten ingresar los datos
- Se usa `if (anguloA == anguloB)` para verificar si los ángulos son iguales
- Dentro de ese bloque se emplean `if` y `else if` para determinar el tipo de ángulo
- Si los ángulos son distintos, se clasifican individualmente con condicionales anidadas
- Se usan operadores lógicos como > y < para definir los rangos de cada tipo de ángulo

Problema 7: Cálculo de funciones trigonométricas

Temática: Funciones matemáticas y condicionales

Descripción: Programa que calcula el seno, coseno y arco tangente de un ángulo, permitiendo al usuario especificar si el ángulo está en grados o radianes. Además, realiza la conversión entre estas unidades.

Listing 7: FuncionesTrigonometricas.cpp

```
1  #include <iostream>
2  #include <cmath>
3  using namespace std;
4
5  int main() {
6      const double PI = 3.1415926536;
7      double angulo, senAng, cosAng, arcTanAng, angRadianes, angGrados;
8      char resp;
9
10     cout << " EN  QUE TIENE EL TAMA O DEL ANGULO?" << endl;
11     cout << " GRADOS (G), RADIANES(R)?: ";
12     cin >> resp;
13
```

```
14      cout << "Ingrese el valor del angulo: ";
15      cin >> angulo;
16
17      if (toupper(resp) == 'G') {
18          angRadianes = angulo * (PI/180);
19          cout << "EQUIVALEN A " << angRadianes << " RADIANES" << endl;
20
21          senAng = sin(angRadianes);
22          cosAng = cos(angRadianes);
23          arcTanAng = atan(angRadianes);
24      } else {
25          angGrados = angulo * (180/PI);
26          cout << "EQUIVALEN A " << angGrados << " GRADOS" << endl;
27
28          senAng = sin(angulo);
29          cosAng = cos(angulo);
30          arcTanAng = atan(angulo);
31      }
32
33      cout << "SENO = " << senAng << endl;
34      cout << "COSENO = " << cosAng << endl;
35      cout << "ARCO TANGENTE = " << arcTanAng << endl;
36
37      return 0;
38  }
```

Explicación:

- `const double PI = 3.1415926536` define la constante PI con mayor precisión
- `double angulo, senAng, ...` declara las variables para almacenar los valores
- `char resp` almacena la elección del usuario (Grados o Radianes)
- `toupper(resp)` convierte la respuesta a mayúscula para uniformidad
- `if (toupper(resp) == 'G')` decide si trabajar con grados o radianes
- `angRadianes = angulo * (PI/180)` convierte grados a radianes
- `angGrados = angulo * (180/PI)` convierte radianes a grados
- `sin()`, `cos()` y `atan()` calculan las funciones trigonométricas

Notas:

- Se usa `toupper()` para aceptar 'g' o 'G' como entrada válida
- Las funciones trigonométricas de `cmath` trabajan en radianes
- Se podría mejorar agregando validación para respuestas no válidas

Problema 8: Equivalencias entre metros, yardas, pies y pulgadas

Temática: Condicionales múltiples, operaciones aritméticas, selección con caracteres

Descripción: Crea un programa en C++ que convierta una unidad de longitud (metros, yardas, pies o pulgadas) a sus equivalentes en las demás unidades. El usuario debe indicar qué unidad desea convertir y proporcionar la cantidad. Este ejercicio permite practicar el uso de 'if' con condiciones sobre caracteres, operaciones aritméticas y mensajes claros al usuario.

Listing 8: EquivalenciasLongitud.cpp

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

int main() {
    // Declaraci n de variables
    double metros, yardas, pies, pulgadas;
    char resp;

    // Men  de selecci n
    cout << "  QU   DESEA CONVERTIR?" << endl;
    cout << " METROS  (M), YARDAS (Y), PIES (P), PULGADAS (U)?: ";
    cin >> resp;

    if (resp == 'M' || resp == 'm') {
        cout << "Ingrese el n mero de metros: ";
        cin >> metros;

        pulgadas = (metros * 100) / 2.54;
        pies = pulgadas / 12;
        yardas = pies / 3;

        cout << metros << " METROS EQUIVALEN A: " << endl;
        cout << pulgadas << " PULGADAS" << endl;
        cout << pies << " PIES" << endl;
        cout << yardas << " YARDAS" << endl;
    }

    if (resp == 'Y' || resp == 'y') {
        cout << "Ingrese el n mero de yardas: ";
        cin >> yardas;

        pies = yardas * 3;
        pulgadas = pies * 12;
        metros = (pulgadas * 2.54) / 100;

        cout << yardas << " YARDAS EQUIVALEN A: " << endl;
```

```
37          cout << pulgadas << " PULGADAS" << endl;
38          cout << pies << " PIES" << endl;
39          cout << metros << " METROS" << endl;
40      }
41
42      if (resp == 'P' || resp == 'p') {
43          cout << "Ingrese el n mero de pies: ";
44          cin >> pies;
45
46          pulgadas = pies * 12;
47          yardas = pies / 3;
48          metros = (pulgadas * 2.54) / 100;
49
50          cout << pies << " PIES EQUIVALEN A: " << endl;
51          cout << pulgadas << " PULGADAS" << endl;
52          cout << yardas << " YARDAS" << endl;
53          cout << metros << " METROS" << endl;
54      }
55
56      if (resp == 'U' || resp == 'u') {
57          cout << "Ingrese el n mero de pulgadas: ";
58          cin >> pulgadas;
59
60          pies = pulgadas / 12;
61          yardas = pies / 3;
62          metros = (pulgadas * 2.54) / 100;
63
64          cout << pulgadas << " PULGADAS EQUIVALEN A: " << endl;
65          cout << pies << " PIES" << endl;
66          cout << yardas << " YARDAS" << endl;
67          cout << metros << " METROS" << endl;
68      }
69
70      return 0;
71  }
```

Explicación:

- `char resp` almacena la opción seleccionada por el usuario (M, Y, P, U)
- `cin » resp` lee la letra que indica qué unidad desea convertir
- Se usa una serie de `if` con comparaciones de letras mayúsculas y minúsculas
- Cada bloque convierte desde una unidad base a las otras tres
- Las constantes usadas son: 1 metro = 100 cm, 1 pulgada = 2.54 cm, 1 pie = 12 pulgadas, 1 yarda = 3 pies

- `cout « ...` se usa para mostrar el resultado en cada bloque

Problema 9: Segunda Ley de Newton

Temática: Estructuras condicionales y fórmulas físicas

Descripción: Programa que calcula fuerza, aceleración o masa según la Segunda Ley de Newton (F = m × a), permitiendo al usuario seleccionar qué variable desea calcular.

Listing 9: SegundaLeyNewton.cpp

```
#include <iostream>
#include <cctype> // Para toupper()
using namespace std;

int main() {
    double F, A, M;
    char Resp;

    cout << "  QU   DESEA CALCULAR?" << endl;
    cout << " FUERZA (F), ACELERACION(A), MASA(M)?: ";
    cin >> Resp;
    Resp = toupper(Resp); // Convertir a may scula

    if (Resp == 'F') {
        cout << "Ingrese la masa (kg): ";
        cin >> M;
        cout << "Ingrese la aceleracion (m/ s  ): ";
        cin >> A;
        F = M * A;
        cout << "F = " << F << " N (Newtons)" << endl;
    }
    else if (Resp == 'A') {
        cout << "Ingrese la masa (kg): ";
        cin >> M;
        cout << "Ingrese la fuerza (N): ";
        cin >> F;
        A = F / M;
        cout << "A = " << A << " m/ s  " << endl;
    }
    else if (Resp == 'M') {
        cout << "Ingrese la fuerza (N): ";
        cin >> F;
        cout << "Ingrese la aceleracion (m/ s  ): ";
        cin >> A;
        M = F / A;
```

```
36          cout << "M = " << M << " kg" << endl;
37      }
38      else {
39          cout << "Opcion no valida. Use F, A o M." << endl;
40      }
41
42      return 0;
43  }
```

Explicación:

- `#include <cctype>` permite usar `toupper()` para estandarizar la entrada
- `double F, A, M` almacenan fuerza (Newtons), aceleración (m/s²) y masa (kg)
- `char Resp` guarda la selección del usuario (F, A o M)
- `toupper(Resp)` convierte la entrada a mayúscula para uniformidad
- `if (Resp == 'F')` bloque para cálculo de fuerza (F = m × a)
- `else if (Resp == 'A')` bloque para cálculo de aceleración (a = F/m)
- `else if (Resp == 'M')` bloque para cálculo de masa (m = F/a)
- `else` maneja entradas no válidas

Mejoras implementadas:

- Conversión automática a mayúsculas para aceptar 'f' o 'F'
- Mensajes descriptivos para la entrada de datos
- Unidades incluidas en los resultados (N, m/s², kg)
- Manejo de opciones no válidas
- Estructura más eficiente con `else if` en lugar de IFs separados

Posibles extensiones:

- Validar que los valores ingresados sean positivos
- Permitir múltiples cálculos en una sola ejecución
- Mostrar la fórmula utilizada en cada cálculo

Problema 10: Solución de Ecuación Cuadrática

Temática: Estructuras condicionales y operaciones matemáticas

Descripción: Programa que resuelve una ecuación cuadrática de la forma $ax^2 + bx + c = 0$, determinando si tiene solución única, raíces reales diferentes o raíces complejas conjugadas.

Listing 10: EcuacionCuadratica.cpp

```cpp
#include <iostream>
#include <cmath> // Para sqrt() y fabs()
using namespace std;

int main() {
    double A, B, C;
    double RaizUnica, ParteReal, ParteImaginaria;
    double RaizReal1, RaizReal2;

    cout << "Ingrese el coeficiente A: ";
    cin >> A;
    cout << "Ingrese el coeficiente B: ";
    cin >> B;
    cout << "Ingrese el coeficiente C: ";
    cin >> C;

    if (A == 0) {
        // Caso especial: ecuaci n lineal
        RaizUnica = -C / B;
        cout << "ECUACION LINEAL - RAIZ UNICA = " << RaizUnica << endl;
    }
    else {
        double discriminante = pow(B, 2) - 4 * A * C;

        if (discriminante == 0) {
            // Ra z  nica
            RaizUnica = -B / (2 * A);
            cout << "RAIZ UNICA = " << RaizUnica << endl;
        }
        else if (discriminante < 0) {
            // Ra ces complejas
            ParteReal = -B / (2 * A);
            ParteImaginaria = sqrt(fabs(discriminante)) / (2 * A);
            cout << "RAICES COMPLEJAS:" << endl;
            cout << ParteReal << " + " << ParteImaginaria << "i" << endl;
            cout << ParteReal << " - " << ParteImaginaria << "i" << endl;
        }
        else {
```

```
            // Raices reales
            RaizReal1 = (-B + sqrt(discriminante)) / (2 * A);
            RaizReal2 = (-B - sqrt(discriminante)) / (2 * A);
            cout << "RAICES REALES:" << endl;
            cout << "Raiz 1 = " << RaizReal1 << endl;
            cout << "Raiz 2 = " << RaizReal2 << endl;
        }
    }

    return 0;
}
```

Explicación:

- `#include <cmath>` para usar funciones matemáticas como raíz cuadrada (`sqrt`) y valor absoluto (`fabs`)
- `double A, B, C` almacenan los coeficientes de la ecuación cuadrática
- Se calcula el `discriminante` = $b^2 - 4ac$ para determinar el tipo de raíces
- `if (A == 0)` maneja el caso especial cuando es ecuación lineal
- `if (discriminante == 0)` calcula la raíz única cuando el discriminante es cero
- `else if (discriminante < 0)` maneja el caso de raíces complejas conjugadas
- `else` calcula las dos raíces reales cuando el discriminante es positivo
- Se imprimen los resultados con formato claro para cada caso

Mejoras implementadas:

- Manejo del caso especial cuando A = 0 (ecuación lineal)
- Uso de `fabs()` para valor absoluto de números reales
- Mensajes de salida más descriptivos
- Cálculo eficiente del discriminante
- Presentación clara de números complejos (formato a ± bi)

Posibles extensiones:

- Validar entrada de datos (evitar divisiones por cero)
- Mostrar la ecuación formateada (ej: "2x² + 3x - 5 = 0")
- Calcular y mostrar el vértice de la parábola
- Opción para resolver múltiples ecuaciones sin salir del programa

Nota Matemática: La naturaleza de las raíces depende del discriminante ($D = b^2 - 4ac$):

- $D > 0$: Dos raíces reales distintas
- $D = 0$: Una raíz real doble
- $D < 0$: Dos raíces complejas conjugadas

Problema 11: Tipo de triángulo

Temática: Estructuras condicionales anidadas y operadores lógicos

Descripción: Programa que determina si un triángulo es equilátero, isósceles o escaleno, basándose en la longitud de sus tres lados.

Listing 11: TipoTriangulo.cpp

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

int main() {
    double LadoA, LadoB, LadoC;

    cout << "Ingrese la longitud del primer lado: ";
    cin >> LadoA;
    cout << "Ingrese la longitud del segundo lado: ";
    cin >> LadoB;
    cout << "Ingrese la longitud del tercer lado: ";
    cin >> LadoC;

    if (LadoA <= 0 || LadoB <= 0 || LadoC <= 0) {
        cout << "Error: Los lados deben ser valores positivos." << endl;
    }
    else if (LadoA + LadoB <= LadoC || LadoA + LadoC <= LadoB || LadoB + ↩
       LadoC <= LadoA) {
        cout << "Error: No cumple la desigualdad triangular." << endl;
    }
    else {
        if (LadoA != LadoB && LadoA != LadoC && LadoB != LadoC) {
            cout << "ES ESCALENO (todos los lados diferentes)" << endl;
        }
        else {
            if (LadoA == LadoB && LadoA == LadoC && LadoB == LadoC) {
                cout << "ES EQUILATERO (todos los lados iguales)" << endl;
            }
            else {
                cout << "ES ISOSCELES (dos lados iguales)" << endl;
```

```
            }
        }
    }

    return 0;
}
```

Explicación:

- `double LadoA, LadoB, LadoC` almacenan las longitudes de los lados
- `cin » LadoA` y similares capturan las medidas ingresadas por el usuario
- Primera validación: `LadoA <= 0` verifica lados positivos
- Segunda validación: `LadoA + LadoB <= LadoC` comprueba desigualdad triangular
- `if (LadoA != LadoB && LadoA != LadoC)` identifica triángulo escaleno
- `if (LadoA == LadoB && LadoA == LadoC)` identifica triángulo equilátero
- `else` caso restante identifica triángulo isósceles

Mejoras implementadas:

- Validación de datos de entrada (valores positivos)
- Comprobación de la desigualdad triangular (suma de dos lados > tercer lado)
- Mensajes descriptivos que incluyen la definición de cada tipo
- Estructura anidada mejor organizada

Posibles extensiones:

- Calcular y mostrar el perímetro del triángulo
- Determinar si es triángulo rectángulo usando el teorema de Pitágoras
- Mostrar el tipo de ángulos (acutángulo, obtusángulo)

Nota: Un triángulo válido debe cumplir:

1. Todos los lados > 0
2. La suma de cualquier dos lados > tercer lado

Problema 12: Cálculo de descuento para cliente

Temática: Estructuras de control CASE y cálculos comerciales

Descripción: Programa que calcula el total a pagar por un cliente según su tipo (1-4) y la cantidad de hojas de hielo seco adquiridas, aplicando diferentes porcentajes de descuento según categoría.

Listing 12: CalculoDescuento.cpp

```cpp
#include <iostream>
#include <iomanip> // Para formateo de salida
#include <string>
using namespace std;

int main() {
    string NombreClie;
    int TipoClie, Cantidad;
    float PrecioUni, SubTotal, Descuento, TotalPagar;

    // Entrada de datos
    cout << "SISTEMA DE VENTAS - HOJAS DE HIELO SECO" << endl;
    cout << "Nombre del cliente: ";
    getline(cin, NombreClie);
    cout << "Tipo de cliente (1-4): ";
    cin >> TipoClie;
    cout << "Cantidad de hojas: ";
    cin >> Cantidad;
    cout << "Precio unitario: ";
    cin >> PrecioUni;

    // C lculos
    SubTotal = Cantidad * PrecioUni;

    switch(TipoClie) {
        case 1: Descuento = SubTotal * 0.05f; break;
        case 2: Descuento = SubTotal * 0.08f; break;
        case 3: Descuento = SubTotal * 0.12f; break;
        case 4: Descuento = SubTotal * 0.15f; break;
        default: Descuento = 0; // Por si ingresa tipo no v lido
    }

    TotalPagar = SubTotal - Descuento;

    // Salida formateada
    cout << fixed << setprecision(2);
    cout << "\n--- RECIBO ---" << endl;
```

```
38      cout << "Cliente: " << NombreClie << endl;
39      cout << "Subtotal: $" << SubTotal << endl;
40      cout << "Descuento (" << (Descuento/SubTotal)*100 << "%): $" << ↩
           Descuento << endl;
41      cout << "TOTAL A PAGAR: $" << TotalPagar << endl;
42
43      return 0;
44  }
```

Explicación:

- `string NombreClie` almacena el nombre del cliente (hasta 30 caracteres)
- `int TipoClie` guarda la categoría del cliente (1 a 4)
- `getline(cin, NombreClie)` permite capturar nombres con espacios
- `switch(TipoClie)` estructura CASE para determinar el descuento
- `fixed « setprecision(2)` formatea la salida a 2 decimales
- Se calcula automáticamente el porcentaje de descuento mostrado

Mejoras implementadas:

- Validación implícita para tipos de cliente no válidos (default case)
- Formato profesional de salida monetaria
- Cálculo y visualización del porcentaje de descuento aplicado
- Recepción correcta de nombres con espacios
- Diseño de recibo claro y organizado

Posibles extensiones:

- Validar que TipoClie esté entre 1-4
- Validar que Cantidad y PrecioUni sean positivos
- Almacenar historial de ventas en archivo
- Mostrar catálogo de tipos de cliente con sus beneficios
- Calcular IVA u otros impuestos

Tabla de descuentos:

Tipo Cliente	Descuento
1	5%
2	8%
3	12%
4	15%

Nota: Los descuentos son acumulables con otras promociones según políticas de la empresa.

Problema 13: Clientes con Artículos

Temática: Ciclos anidados, acumuladores y entrada/salida de datos

Descripción: Escribe un programa en C++ que permita registrar varios clientes y los artículos que compran, calculando el total a pagar por cliente y el total general. Se debe preguntar si hay más artículos y más clientes tras cada iteración.

Listing 13: ClientesConArticulos.cpp

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

int main() {
char NombreClie[30], Articulo[30], Otro, Hay;
int TotClientes = 0, Cantidad;
float Precio, TotPagar, TotPagarClie, TotPagarGral = 0;

kotlin
Copiar
Editar
cout << "LISTADO DE CLIENTES Y ARTICULOS" << endl;

do {
    cout << "\nIngrese el NOMBRE DEL CLIENTE: ";
    cin >> NombreClie;
    cout << "Cliente: " << NombreClie << endl;

    TotPagarClie = 0;

    do {
        cout << "Ingrese el ARTICULO: ";
        cin >> Articulo;
        cout << "Ingrese la CANTIDAD: ";
        cin >> Cantidad;
        cout << "Ingrese el PRECIO: ";
        cin >> Precio;

        TotPagar = Cantidad * Precio;
        cout << "Articulo: " << Articulo << ", Total a pagar: $" << ↩
           TotPagar << endl;

        TotPagarClie += TotPagar;

        cout << " HAY  OTRO ARTICULO (S/N)? ";
        cin >> Hay;
```

```
    } while (Hay != 'N' && Hay != 'n');

    cout << "Total a pagar por cliente: $" << TotPagarClie << endl;

    TotClientes++;
    TotPagarGral += TotPagarClie;

    cout << " HAY  OTRO CLIENTE (S/N)? ";
    cin >> Otro;

} while (Otro != 'N' && Otro != 'n');

cout << "\nTOTAL CLIENTES: " << TotClientes << endl;
cout << "TOTAL GENERAL A PAGAR: $" << TotPagarGral << endl;

return 0;
}
```

Explicación:

- `char NombreClie[30], Articulo[30]`: Cadenas para almacenar el nombre del cliente y el artículo.
- `char Otro, Hay`: Variables de control para continuar con otro cliente o artículo.
- `int TotClientes, Cantidad`: Contadores para total de clientes y cantidad comprada.
- `float Precio, TotPagar, TotPagarClie, TotPagarGral`: Variables para cálculos monetarios.
- El ciclo `do-while` externo permite registrar varios clientes.
- El ciclo `do-while` interno permite registrar múltiples artículos por cliente.
- Se acumula el total a pagar por cliente (`TotPagarClie`) y el total general (`TotPagarGral`).

Problema 14: Sistema de Vendedores (15 máximo)

Temática: Estructuras de control anidadas con límite fijo

Descripción: Programa que registra ventas e incentivos para hasta 15 vendedores, calculando totales individuales y generales, con diferentes porcentajes de incentivo según tipo de artículo.

Listing 14: SistemaVendedoresLimitado.cpp

```
#include <iostream>
```

```cpp
#include <iomanip>
#include <string>
using namespace std;

int main() {
    const int MAX_VENDEDORES = 15;
    string NombreVend, Articulo;
    char Hay;
    int TotVend = 0, Cantidad, Clave;
    float Precio, TotVendido, Incentivo, TotIncentivo;
    float Venta, TotGralIncentivo = 0, TotGralVenta = 0;

    cout << "SISTEMA DE VENTAS (MAX " << MAX_VENDEDORES << " VENDEDORES)" << endl;
    cout << "==================================" << endl << endl;

    do {
        cout << "\nVendedor " << (TotVend + 1) << " de " << MAX_VENDEDORES << endl;
        cout << "Nombre del vendedor: ";
        cin.ignore();
        getline(cin, NombreVend);

        TotVendido = 0;
        TotIncentivo = 0;

        do {
            cout << "\nRegistro de art culo:" << endl;
            cout << "Nombre del art culo: ";
            getline(cin, Articulo);

            cout << "Clave (1-4): ";
            while(!(cin >> Clave) || Clave < 1 || Clave > 4) {
                cout << "Inv lido. Ingrese clave (1-4): ";
                cin.clear();
                cin.ignore(1000, '\n');
            }

            cout << "Cantidad: ";
            while(!(cin >> Cantidad) || Cantidad <= 0) {
                cout << "Inv lido. Ingrese cantidad positiva: ";
                cin.clear();
                cin.ignore(1000, '\n');
            }

            cout << "Precio unitario: $";
            while(!(cin >> Precio) || Precio <= 0) {
```

```
                cout << "Inv lido. Ingrese precio positivo: $";
                cin.clear();
                cin.ignore(1000, '\n');
            }

            Venta = Cantidad * Precio;

            switch(Clave) {
                case 1: Incentivo = Venta * 0.15f; break;
                case 2: Incentivo = Venta * 0.10f; break;
                case 3: Incentivo = Venta * 0.05f; break;
                case 4: Incentivo = Venta * 0.03f; break;
            }

            TotVendido += Venta;
            TotIncentivo += Incentivo;

            cout << "\n Agregar otro art culo? (S/N): ";
            cin >> Hay;
            cin.ignore();
        } while(toupper(Hay) == 'S');

        // Reporte por vendedor
        cout << fixed << setprecision(2);
        cout << "\n--- RESUMEN VENDEDOR ---" << endl;
        cout << "Vendedor: " << NombreVend << endl;
        cout << "Total vendido: $" << setw(10) << TotVendido << endl;
        cout << "Incentivo:     $" << setw(10) << TotIncentivo << endl;

        TotVend++;
        TotGralVenta += TotVendido;
        TotGralIncentivo += TotIncentivo;

    } while(TotVend < MAX_VENDEDORES);

    // Reporte final
    cout << "\n--- REPORTE FINAL ---" << endl;
    cout << "Total de vendedores procesados: " << TotVend << endl;
    cout << "Venta global total: $" << setw(12) << TotGralVenta << endl;
    cout << "Incentivo global:   $" << setw(12) << TotGralIncentivo << ↩
       endl;

    return 0;
}
```

Explicación:

- `const int MAX_VENDEDORES = 15` define el límite de vendedores
- Estructura `do-while` externa controla el límite de 15 vendedores
- Estructura `do-while` interna maneja múltiples artículos por vendedor
- Validaciones robustas para claves (1-4), cantidades y precios positivos
- `setw()` para alinear columnas en los reportes
- Contador `TotVend` muestra progreso (1/15, 2/15...)

Mejoras implementadas:

- Validación de entrada para evitar datos incorrectos
- Visualización del progreso (vendedor X de 15)
- Formato profesional con alineación de columnas
- Manejo seguro de entradas con `cin.clear()` y `ignore()`
- Eliminación de variable `Otro` ya que el límite es fijo

Tabla de Incentivos:

Clave	Tipo Artículo	Incentivo
1	Electrónicos	15%
2	Muebles	10%
3	Ropa	5%
4	Accesorios	3%

Diagrama de Flujo:

1. Inicializar contadores y totales
2. Para cada vendedor (hasta 15):
    - Registrar datos personales
    - Para cada artículo:
        - Capturar detalles
        - Calcular venta e incentivo
        - Acumular totales
    - Mostrar resumen individual
3. Mostrar reporte consolidado

Notas Técnicas:

- Usa `float` para cálculos monetarios (precisión suficiente)
- `toupper()` normaliza respuestas S/N
- `setprecision(2)` asegura 2 decimales en valores monetarios
- El límite de 15 vendedores evita sobrecarga de memoria

Problema 15: Secuencia Fibonacci

Temática: Bucles y secuencias matemáticas

Descripción: Programa que genera los primeros 20 términos de la secuencia Fibonacci, donde cada número es la suma de los dos anteriores, comenzando con 0 y 1.

Listing 15: Fibonacci.cpp

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

int main() {
    int penultimo = 0, ultimo = 1;
    int numero;

    cout << "Secuencia Fibonacci (primeros 20 terminos):" << endl;

    // Imprimir los dos primeros t rminos
    cout << penultimo << ", " << ultimo;

    for(int i = 3; i <= 20; i++) {  // Comenzamos desde 3 porque ya tenemos 2 t rminos
        numero = penultimo + ultimo;
        cout << ", " << numero;

        // Actualizar valores para siguiente iteraci n
        penultimo = ultimo;
        ultimo = numero;
    }

    cout << endl;
    return 0;
}
```

Explicación:

- `int penultimo = 0, ultimo = 1` inicializa los primeros dos términos
- El bucle `for` itera desde 3 hasta 20 (genera 18 términos adicionales)
- `numero = penultimo + ultimo` calcula el siguiente término Fibonacci
- Se actualizan `penultimo` y `ultimo` para la siguiente iteración
- Los términos se imprimen en formato de lista separada por comas

Mejoras implementadas:

- Muestra los primeros 20 términos correctamente (incluyendo los iniciales 0 y 1)

- Formato de salida mejorado con separadores
- Comentarios explicativos en el código
- Nombre de variables más descriptivos

Salida esperada:

```
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 41
```

Notas matemáticas:

- La secuencia Fibonacci se define como:

$$F(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ F(n-1) + F(n-2) & \text{si } n > 1 \end{cases}$$

- Relación áurea: $\lim_{n\to\infty} \frac{F(n+1)}{F(n)} = \phi \approx 1.61803$
- Aplicaciones: algoritmos, teoría de números, patrones en naturaleza

Variantes para explorar:

- Generar términos hasta que se alcance un valor máximo
- Implementación recursiva
- Calcular la razón entre términos consecutivos
- Almacenar la secuencia en un arreglo

Problema 16: Cálculo de Potencias

Temática: Bucles anidados y operaciones matemáticas

Descripción: Programa que calcula y muestra las potencias n^n para los números del 1 al 8, donde cada número se eleva a sí mismo.

Listing 16: Potencias.cpp

```
#include <iostream>
#include <iomanip>
#include <cmath> // Para pow() aunque usaremos multiplicaci n iterativa
using namespace std;

int main() {
    cout << "POTENCIAS n^n DEL 1 AL 8" << endl;
    cout << "========================" << endl;
    cout << left << setw(10) << "Numero" << "Potencia" << endl;
```

```
    cout << "----------------" << endl;

    for(int numero = 1; numero <= 8; numero++) {
        double potencia = numero; // Inicializamos con el valor base

        // Calculamos n^n mediante multiplicaci n iterativa
        for(int i = 1; i < numero; i++) {
            potencia *= numero;
        }

        cout << left << setw(10) << numero << potencia << endl;
    }

    return 0;
}
```

Explicación:

- Bucle externo `for(int numero = 1; numero <= 8; numero++)` itera del 1 al 8
- `double potencia = numero` inicializa la potencia con el valor base
- Bucle interno `for(int i = 1; i < numero; i++)` realiza la multiplicación iterativa
- `potencia *= numero` acumula el producto para calcular n^n
- `setw(10)` formatea la salida en columnas alineadas

Salida esperada:

```
POTENCIAS n^n DEL 1 AL 8
========================
Numero    Potencia
----------------
1         1
2         4
3         27
4         256
5         3125
6         46656
7         823543
8         16777216
```

Mejoras implementadas:

- Encabezado descriptivo y formato tabular
- Uso de `double` para manejar valores grandes

- Implementación didáctica con multiplicación iterativa
- Alineación profesional de columnas

Notas matemáticas:

- La operación n^n crece muy rápidamente (crecimiento exponencial)
- Para $n = 8$, $8^8 = 16,777,216$
- Complejidad algorítmica: $O(n^2)$ para este método

Variantes para explorar:

- Usar la función `pow()` de cmath
- Calcular potencias n^m con ambos variables
- Implementar recursividad
- Mostrar notación científica para números grandes
- Calcular sumatorias de potencias

Comparación de métodos:

Método	Ventaja	Desventaja
Multiplicación iterativa	Didáctico	Ineficiente para n grandes
Función `pow()`	Óptimo	Menos transparente
Recursivo	Elegante	Consumo de memoria

Problema 17: Ecuaciones Cuadráticas para A=1-5

Temática: Bucles con estructuras condicionales anidadas

Descripción: Programa que resuelve ecuaciones cuadráticas para valores de A desde 1 hasta 5, con B y C calculados automáticamente, mostrando todos los casos posibles (raíz única, reales diferentes y complejas).

Listing 17: EcuacionesCuadraticas.cpp

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

int main() {
    double A, B, C = 6;
    double discriminante, raizUnica, parteReal, parteImaginaria;
    double raizReal1, raizReal2;
```

```
10
11      cout << "SOLUCION DE ECUACIONES CUADRATICAS PARA A=1-5" << endl;
12      cout << "=============================================" << endl;
13      cout << fixed << setprecision(3);
14
15      for(int i = 1; i <= 5; i++) {
16          A = i;
17          C = C - A;
18          B = A - C;
19
20          cout << "\n--- Caso A = " << A << " ---" << endl;
21          cout << "Coeficientes: A=" << A << ", B=" << B << ", C=" << C << ↩
                endl;
22
23          if(A == 1) {
24              raizUnica = -C / A;
25              cout << "RAIZ UNICA: " << raizUnica << endl;
26          }
27          else {
28              discriminante = pow(B, 2) - 4 * A * C;
29
30              if(discriminante < 0) {
31                  parteReal = -B / (2 * A);
32                  parteImaginaria = sqrt(fabs(discriminante)) / (2 * A);
33                  cout << "RAICES COMPLEJAS:" << endl;
34                  cout << parteReal << " + " << parteImaginaria << "i" << ↩
                        endl;
35                  cout << parteReal << " - " << parteImaginaria << "i" << ↩
                        endl;
36              }
37              else {
38                  raizReal1 = (-B + sqrt(discriminante)) / (2 * A);
39                  raizReal2 = (-B - sqrt(discriminante)) / (2 * A);
40                  cout << "RAICES REALES:" << endl;
41                  cout << "Raiz 1 = " << raizReal1 << endl;
42                  cout << "Raiz 2 = " << raizReal2 << endl;
43              }
44          }
45      }
46
47      return 0;
48  }
```

Explicación:

- Bucle `for(int i = 1; i <= 5; i++)` itera para A=1 a A=5
- `C = C - A` y `B = A - C` calculan los coeficientes según el algoritmo

- `if(A == 1)` maneja el caso especial de ecuación lineal
- `discriminante = pow(B,2) - 4*A*C` determina el tipo de raíces
- Se muestran resultados formateados con 3 decimales

Salida esperada:

```
SOLUCION DE ECUACIONES CUADRATICAS PARA A=1-5
==========================================

--- Caso A = 1 ---
Coeficientes: A=1, B=-4, C=5
RAIZ UNICA: -5.000

--- Caso A = 2 ---
Coeficientes: A=2, B=-1, C=3
RAICES COMPLEJAS:
0.250 + 1.199i
0.250 - 1.199i

--- Caso A = 3 ---
Coeficientes: A=3, B=3, C=0
RAICES REALES:
Raiz 1 = 0.000
Raiz 2 = -1.000

--- Caso A = 4 ---
Coeficientes: A=4, B=8, C=-4
RAICES REALES:
Raiz 1 = 0.414
Raiz 2 = -2.414

--- Caso A = 5 ---
Coeficientes: A=5, B=14, C=-9
RAICES REALES:
Raiz 1 = 0.562
Raiz 2 = -3.162
```

Análisis Matemático:

- Para A=1: Ecuación lineal (B=0) con solución única
- Para A=2: Discriminante negativo (raíces complejas conjugadas)
- Para A=3,4,5: Discriminante positivo (dos raíces reales distintas)

- Relación entre coeficientes: $C_n = C_{n-1} - A$, $B = A - C$

Mejoras implementadas:

- Precisión decimal consistente (3 decimales)
- Encabezados claros para cada caso
- Cálculo eficiente del discriminante
- Formato profesional para números complejos

Posibles extensiones:

- Mostrar la ecuación completa formateada (ax² + bx + c = 0)
- Calcular suma y producto de raíces
- Graficar las parábolas correspondientes
- Mostrar el valor del discriminante

Nota Pedagógica: Este ejercicio demuestra:

- Todos los casos posibles de ecuaciones cuadráticas
- Evolución de las soluciones al variar los coeficientes
- Transición entre diferentes tipos de raíces

Problema 18: Simulación de Inversión a 24 Meses

Temática: Interés compuesto y acumulación financiera

Descripción: Programa que simula una inversión con capitalización mensual durante 24 meses, mostrando el crecimiento del capital con una tasa de interés anual del 36%.

Listing 18: Inversion24Meses.cpp

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

int main() {
    const double TASA_ANUAL = 0.36; // 36%
    const int MESES = 24;
    int mes;
    double capital = 10000.00;
    double interes, saldo, totalInteres = 0;

    cout << "SIMULACION DE INVERSION A 24 MESES" << endl;
```

```
13      cout << "Capital inicial: $" << fixed << setprecision(2) << capital <<↩
            endl;
14      cout << "Tasa anual: " << TASA_ANUAL*100 << "%" << endl;
15      cout << "========================================" << endl;
16      cout << setw(5) << "Mes" << setw(12) << "Capital"
17           << setw(12) << "Interes" << setw(12) << "Saldo" << endl;
18      cout << "----------------------------------------" << endl;
19
20      for(mes = 1; mes <= MESES; mes++) {
21          interes = capital * (TASA_ANUAL/12); // Inter s mensual
22          saldo = capital + interes;
23          totalInteres += interes;
24
25          cout << setw(5) << mes << setw(12) << capital
26               << setw(12) << interes << setw(12) << saldo << endl;
27
28          capital = saldo; // Capitalizaci n mensual
29      }
30
31      cout << "========================================" << endl;
32      cout << "Total intereses ganados: $" << totalInteres << endl;
33      cout << "Saldo final: $" << capital << endl;
34
35      return 0;
36  }
```

Explicación:

- `TASA_ANUAL = 0.36` representa el 36% de interés anual
- `capital = 10000.00` es la inversión inicial
- El bucle `for` itera por cada mes (1 a 24)
- `interes = capital * (TASA_ANUAL/12)` calcula el interés mensual
- `saldo = capital + interes` actualiza el saldo mensual
- `totalInteres` acumula la suma de todos los intereses
- Se usa capitalización compuesta (interés sobre interés)

Salida esperada (primeros y últimos meses):

```
SIMULACION DE INVERSION A 24 MESES
Capital inicial: $10000.00
Tasa anual: 36%
========================================
  Mes     Capital     Interes       Saldo
```

```
----------------------------------------
    1   10000.00     300.00   10300.00
    2   10300.00     309.00   10609.00
    3   10609.00     318.27   10927.27
    ...
   23   19502.94     585.09   20088.03
   24   20088.03     602.64   20690.67
========================================
Total intereses ganados: $10690.67
Saldo final: $20690.67
```

Fórmulas clave:

- Interés mensual: $I_m = C \times \left(\frac{r}{12}\right)$
- Saldo mensual: $S = C + I_m$
- Interés acumulado: $I_{total} = \sum_{i=1}^{24} I_i$
- Tasa mensual equivalente: $r_m = \frac{0.36}{12} = 3\%$

Mejoras implementadas:

- Constantes para valores fijos (tasa, meses)
- Formato tabular profesional con `setw()`
- Precisión de 2 decimales para valores monetarios
- Encabezados y resúmenes claros
- Cálculo del saldo final y total de intereses

Análisis Financiero:

- La inversión crece de $10,000 a $20,690.67 en 2 años
- El interés total representa el 106.9% del capital inicial
- Rendimiento efectivo anual: 43.9% (por capitalización mensual)
- Cada mes el capital aumenta aproximadamente un 3%

Posibles extensiones:

- Permitir entrada de capital inicial y tasa por el usuario
- Mostrar gráfica de crecimiento
- Comparar con interés simple
- Calcular impuestos sobre ganancias
- Exportar resultados a archivo CSV

Problema 19: Cálculo de Producción de Trabajadores

Temática: Procesamiento de datos con bucles anidados

Descripción: Programa que registra la producción semanal (6 días) de 15 trabajadores, calculando la producción total por trabajador y la producción general de toda la plantilla.

Listing 19: ProduccionTrabajadores.cpp

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

int main() {
    const int NUM_TRABAJADORES = 15;
    const int DIAS_SEMANA = 6;
    string NombreTra;
    int TotTrab = 0, TotProdGral = 0;

    cout << "SISTEMA DE REGISTRO DE PRODUCCION" << endl;
    cout << "==================================" << endl << endl;

    for(int T = 1; T <= NUM_TRABAJADORES; T++) {
        cout << "\nTrabajador " << T << " de " << NUM_TRABAJADORES << endl;
        cout << "Nombre del trabajador: ";
        cin.ignore();
        getline(cin, NombreTra);

        int TotProd = 0;

        for(int D = 1; D <= DIAS_SEMANA; D++) {
            int ProDia;
            cout << "Produccion del dia " << D << ": ";
            while(!(cin >> ProDia) || ProDia < 0) {
                cout << "Valor invalido. Ingrese produccion (>=0): ";
                cin.clear();
                cin.ignore(1000, '\n');
            }
            TotProd += ProDia;
        }

        cout << "\nRESUMEN TRABAJADOR" << endl;
        cout << "Nombre: " << NombreTra << endl;
        cout << "Produccion total: " << TotProd << " unidades" << endl;

```

```
38          TotProdGral += TotProd;
39          TotTrab++;
40      }
41
42      cout << "\n=== REPORTE FINAL ===" << endl;
43      cout << "Total de trabajadores procesados: " << TotTrab << endl;
44      cout << "Produccion general total: " << TotProdGral << " unidades" << ↩
            endl;
45      cout << "Promedio por trabajador: " << fixed << setprecision(1)
46           << (double)TotProdGral/TotTrab << " unidades" << endl;
47
48      return 0;
49  }
```

Explicación:

- `const int NUM_TRABAJADORES = 15` y `DIAS_SEMANA = 6` definen los límites
- Bucle externo `for` recorre cada trabajador (1 a 15)
- Bucle interno `for` recoge la producción de cada día (1 a 6)
- `getline(cin, NombreTra)` permite nombres con espacios
- Validación de entrada para producción no negativa
- `TotProd` acumula la producción individual
- `TotProdGral` suma la producción de todos los trabajadores

Salida esperada:

```
SISTEMA DE REGISTRO DE PRODUCCION
================================

Trabajador 1 de 15
Nombre del trabajador: Juan Pérez
Produccion del dia 1: 50
...
Produccion del dia 6: 45

RESUMEN TRABAJADOR
Nombre: Juan Pérez
Produccion total: 320 unidades

...

=== REPORTE FINAL ===
```

```
Total de trabajadores procesados: 15
Produccion general total: 4875 unidades
Promedio por trabajador: 325.0 unidades
```

Mejoras implementadas:

- Constantes para valores fijos (15 trabajadores, 6 días)
- Validación de entrada para valores no negativos
- Cálculo automático del promedio por trabajador
- Mensajes descriptivos durante la ejecución
- Formato profesional de salida

Análisis de Datos:

- Sistema ideal para control de productividad laboral
- Permite identificar trabajadores de alto/bajo rendimiento
- Base para cálculos de bonos por productividad
- Fácilmente adaptable a diferentes períodos (meses, trimestres)

Posibles extensiones:

- Almacenar datos en archivos para persistencia
- Mostrar ranking de trabajadores por productividad
- Calcular variación día a día
- Generar reportes gráficos
- Agregar meta de producción y calcular cumplimiento

Estructura del programa:

1. Inicialización de variables y constantes
2. Encabezado del sistema
3. Bucle principal por cada trabajador:
    - Captura de nombre
    - Bucle de captura diaria con validación
    - Cálculo de totales individuales
4. Cálculo y muestra de resultados generales

Problema 20: Cálculo de Factoriales para N Números

Temática: Bucles anidados y matemáticas discretas

Descripción: Programa que calcula el factorial para una serie de N números ingresados por el usuario, manejando correctamente el caso especial del 0! = 1.

Listing 20: Factoriales.cpp

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

int main() {
    int N, I, J = 0, Fact, Num;

    cout << "CALCULADORA DE FACTORIALES" << endl;
    cout << "Ingrese la cantidad de numeros a procesar: ";
    cin >> N;

    // Validar entrada positiva
    while(N <= 0) {
        cout << "Error: Debe ser positivo. Reingrese: ";
        cin >> N;
    }

    do {
        J++;
        cout << "\nIngrese el numero " << J << ": ";
        cin >> Num;

        // Validar n mero no negativo
        while(Num < 0) {
            cout << "Error: Factorial no definido para negativos. ↩
               Reingrese: ";
            cin >> Num;
        }

        if(Num == 0) {
            Fact = 1; // Caso especial 0! = 1
        } else {
            Fact = 1;
            for(I = Num; I >= 1; I--) {
                Fact *= I; // Equivalente a Fact = Fact * I
            }
        }

        cout << "El factorial de " << Num << " es: " << Fact << endl;
```

```
38
39      } while(J < N);
40
41      cout << "\nSe calcularon " << N << " factoriales." << endl;
42      return 0;
43  }
```

Explicación:

- `int N` almacena la cantidad de números a procesar
- Bucle `do-while` principal controla la cantidad de números (J <= N)
- `if(Num == 0)` maneja el caso especial del factorial de 0
- Bucle `for` interno calcula el factorial mediante multiplicación sucesiva
- `Fact *= I` acumula el producto factorial
- Validaciones para entradas no negativas

Ejemplo de ejecución:

```
CALCULADORA DE FACTORIALES
Ingrese la cantidad de numeros a procesar: 3

Ingrese el numero 1: 5
El factorial de 5 es: 120

Ingrese el numero 2: 0
El factorial de 0 es: 1

Ingrese el numero 3: 7
El factorial de 7 es: 5040

Se calcularon 3 factoriales.
```

Mejoras implementadas:

- Validación de entradas para números no negativos
- Mensajes descriptivos durante la ejecución
- Manejo explícito del caso 0! = 1
- Contador de números procesados
- Estructura clara con comentarios explicativos

Análisis Matemático:

- Definición factorial: $n! = \prod_{k=1}^{n} k = 1 \times 2 \times \cdots \times n$
- Caso especial: $0! = 1$ por definición
- Crecimiento factorial es más rápido que exponencial
- Para $n > 20$, los resultados exceden la capacidad de `int` (usar `long long`)

Posibles extensiones:

- Manejar números grandes con bibliotecas de precisión arbitraria
- Calcular factoriales dobles (n!!)
- Mostrar gráfica de crecimiento factorial
- Implementar versión recursiva
- Calcular coeficientes binomiales $\binom{n}{k} = \frac{n!}{k!(n-k)!}$

Nota sobre eficiencia:

- Complejidad algorítmica: O(N × M) donde M es el valor numérico máximo
- Para optimización, podrían almacenarse factoriales precalculados
- Versión recursiva es menos eficiente en tiempo y espacio

Problema 21: Cálculo de Materiales Requeridos

Temática: Procesamiento de pedidos y cálculos industriales

Descripción: Programa que calcula los materiales necesarios para la producción basado en pedidos acumulados, aplicando factores de conversión específicos para cada tipo de material.

Listing 21: MaterialesRequeridos.cpp

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

int main() {
    char Resp;
    int Cantidad, TotProd = 0;
    int Material1, Material2, Material3, Material4, Material5, Material6;

    cout << "SISTEMA DE CALCULO DE MATERIALES" << endl;
    cout << " Hay  pedido (S/N)? ";
    cin >> Resp;
```

```
    Resp = toupper(Resp); // Normalizar a may scula

    while(Resp == 'S') {
        cout << "Ingrese la cantidad pedida: ";
        while(!(cin >> Cantidad) || Cantidad <= 0) {
            cout << "Error: Ingrese cantidad positiva: ";
            cin.clear();
            cin.ignore(1000, '\n');
        }

        TotProd += Cantidad;

        cout << " Hay  otro pedido (S/N)? ";
        cin >> Resp;
        Resp = toupper(Resp);
    }

    // C lculo de materiales requeridos
    Material1 = TotProd * 3;
    Material2 = TotProd * 4;
    Material3 = TotProd * 1;
    Material4 = TotProd * 2;
    Material5 = TotProd * 3;
    Material6 = TotProd * 2;

    // Mostrar resultados
    cout << "\nMATERIALES REQUERIDOS PARA " << TotProd << " UNIDADES" << ↩
       endl;
    cout << "====================================" << endl;
    cout << setw(10) << "Material" << setw(15) << "Cantidad" << endl;
    cout << "------------------------------------" << endl;
    cout << setw(10) << "Tipo 1" << setw(15) << Material1 << endl;
    cout << setw(10) << "Tipo 2" << setw(15) << Material2 << endl;
    cout << setw(10) << "Tipo 3" << setw(15) << Material3 << endl;
    cout << setw(10) << "Tipo 4" << setw(15) << Material4 << endl;
    cout << setw(10) << "Tipo 5" << setw(15) << Material5 << endl;
    cout << setw(10) << "Tipo 6" << setw(15) << Material6 << endl;
    cout << "====================================" << endl;

    return 0;
}
```

Explicación:

- `while(Resp == 'S')` procesa múltiples pedidos hasta que se ingrese 'N'
- `toupper(Resp)` normaliza la entrada a mayúscula

- Validación para cantidades positivas
- `TotProd` acumula el total de unidades pedidas
- Cada material se calcula multiplicando el total por un factor específico
- Salida formateada en tabla con `setw()`

Ejemplo de ejecución:

```
SISTEMA DE CALCULO DE MATERIALES
¿Hay pedido (S/N)? S
Ingrese la cantidad pedida: 50
¿Hay otro pedido (S/N)? S
Ingrese la cantidad pedida: 30
¿Hay otro pedido (S/N)? N

MATERIALES REQUERIDOS PARA 80 UNIDADES
====================================
Material      Cantidad
------------------------------------
Tipo 1             240
Tipo 2             320
Tipo 3              80
Tipo 4             160
Tipo 5             240
Tipo 6             160
====================================
```

Mejoras implementadas:

- Validación robusta para cantidades positivas
- Normalización de respuestas S/N más amigable con mensajes claros
- Formato tabular profesional para resultados
- Cálculo automático de todos los materiales

Análisis Industrial:

- Sistema ideal para gestión de inventario
- Factores de conversión configurables (3:1, 4:1, etc.)
- Fácilmente adaptable a diferentes productos
- Base para sistemas MRP (Material Requirements Planning)

Posibles extensiones:

- Almacenar factores de conversión en arreglos
- Agregar nombres descriptivos para cada material
- Calcular costos basados en precios unitarios
- Generar órdenes de compra automáticas
- Exportar resultados a hojas de cálculo

Tabla de Factores:

Material	Factor	Ejemplo para 100 unidades
1	3	300
2	4	400
3	1	100
4	2	200
5	3	300
6	2	200

Problema 22: Registro de Precipitaciones por Población

Temática: Acumulación de datos meteorológicos con estructuras anidadas

Descripción: Programa que registra acumulados de lluvia para múltiples poblaciones, calculando totales por población y generales, e identificando poblaciones sin precipitación.

Listing 22: RegistroLluvias.cpp

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

int main() {
    string Poblacion;
    char Hay, Otro;
    int TotPobla = 0, TotPobNoLluvia = 0;
    double Lluvia, TotLluviaPob, TotLluviaGral = 0;

    cout << "SISTEMA DE REGISTRO DE PRECIPITACIONES" << endl;
    cout << "======================================" << endl;

    cout << "\n Hay  poblaci n para registrar (S/N)? ";
    cin >> Hay;
    Hay = toupper(Hay);

    while(Hay == 'S') {
```

```
20          cout << "\nNombre de la poblaci n: ";
21          cin.ignore();
22          getline(cin, Poblacion);
23
24          TotLluviaPob = 0;
25          cout << " Hubo  lluvia en " << Poblacion << " (S/N)? ";
26          cin >> Otro;
27          Otro = toupper(Otro);
28
29          while(Otro == 'S') {
30              cout << "Mil metros  c bicos de lluvia: ";
31              while(!(cin >> Lluvia) || Lluvia < 0) {
32                  cout << "Error: Ingrese valor positivo: ";
33                  cin.clear();
34                  cin.ignore(1000, '\n');
35              }
36
37              TotLluviaPob += Lluvia;
38
39              cout << " Registrar  otra lluvia para " << Poblacion << " (S/N↩
                    )? ";
40              cin >> Otro;
41              Otro = toupper(Otro);
42          }
43
44          // Resultados por poblaci n
45          cout << fixed << setprecision(2);
46          cout << "\n--- RESUMEN " << Poblacion << " ---" << endl;
47          cout << "Total lluvia: " << TotLluviaPob << " mm " << endl;
48
49          TotPobla++;
50          TotLluviaGral += TotLluviaPob;
51
52          if(TotLluviaPob == 0) {
53              TotPobNoLluvia++;
54              cout << "Esta poblaci n no registr  lluvia" << endl;
55          }
56
57          cout << "\n Registrar otra poblaci n (S/N)? ";
58          cin >> Hay;
59          Hay = toupper(Hay);
60      }
61
62      // Reporte final
63      cout << "\n=== ESTADISTICAS GENERALES ===" << endl;
64      cout << "Total poblaciones registradas: " << TotPobla << endl;
65      cout << "Acumulado total de lluvia: " << TotLluviaGral << " mm " << ↩
```

```
        endl;
66      cout << "Poblaciones sin lluvia: " << TotPobNoLluvia << endl;
67      cout << "Promedio de lluvia por poblaci n: "
68           << (TotPobla > 0 ? TotLluviaGral/TotPobla : 0) << " mm " << endl↩
                ;
69
70      return 0;
71  }
```

Explicación:

- `while(Hay == 'S')` controla el registro de múltiples poblaciones
- `while(Otro == 'S')` maneja múltiples registros de lluvia por población
- `toupper()` normaliza respuestas S/N
- Validación para valores de lluvia no negativos
- `TotLluviaPob` acumula la lluvia por población
- `TotLluviaGral` suma todos los registros
- `TotPobNoLluvia` cuenta poblaciones sin precipitación

Ejemplo de ejecución:

```
SISTEMA DE REGISTRO DE PRECIPITACIONES
======================================

¿Hay población para registrar (S/N)? S

Nombre de la población: Valle Verde
¿Hubo lluvia en Valle Verde (S/N)? S
Milímetros cúbicos de lluvia: 12.5
¿Registrar otra lluvia para Valle Verde (S/N)? S
Milímetros cúbicos de lluvia: 8.3
¿Registrar otra lluvia para Valle Verde (S/N)? N

--- RESUMEN VALLE VERDE ---
Total lluvia: 20.80 mm³

¿Registrar otra población (S/N)? S

Nombre de la población: Loma Seca
¿Hubo lluvia en Loma Seca (S/N)? N
```

```
--- RESUMEN LOMA SECA ---
Total lluvia: 0.00 mm³
Esta población no registró lluvia

=== ESTADISTICAS GENERALES ===
Total poblaciones registradas: 2
Acumulado total de lluvia: 20.80 mm³
Poblaciones sin lluvia: 1
Promedio de lluvia por población: 10.40 mm³
```

Mejoras implementadas:

- Validación de entradas para valores positivos
- Normalización de respuestas S/N
- Cálculo automático del promedio de lluvia
- Identificación clara de poblaciones sin lluvia
- Formato profesional con precisión decimal

Análisis Meteorológico:

- Sistema ideal para estudios de distribución pluvial
- Permite identificar patrones espaciales de precipitación
- Base para alertas tempranas de sequía
- Datos útiles para agricultura de precisión

Posibles extensiones:

- Almacenar fechas de cada registro de lluvia
- Georreferenciar las poblaciones
- Generar mapas de calor de precipitación
- Exportar datos a formatos científicos (CSV, JSON)
- Calcular percentiles históricos

Estructura del programa:

1. Inicialización de variables acumuladoras
2. Bucle principal de registro por población
3. Bucle anidado para múltiples mediciones de lluvia
4. Cálculo de estadísticas por población
5. Actualización de acumuladores globales
6. Generación de reporte consolidado

Problema 23: Sistema de Registro de Mantenimiento de Máquinas

Temática: Gestión industrial y clasificación de mantenimientos

Descripción: Programa que registra los mantenimientos realizados en 10 máquinas, clasificándolos por tipo y generando reportes individuales y consolidados.

Listing 23: MantenimientoMaquinas.cpp

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

int main() {
    const int TOTAL_MAQUINAS = 10;
    const int TOTAL_TIPOS = 7;
    char Hay;
    int TotManten = 0, CantManten, Tipo, Maquina;
    int contadorTipos[TOTAL_TIPOS + 1] = {0}; //  ndices  1-7

    cout << "SISTEMA DE REGISTRO DE MANTENIMIENTO" << endl;
    cout << "====================================" << endl << endl;

    for(Maquina = 1; Maquina <= TOTAL_MAQUINAS; Maquina++) {
        CantManten = 0;
        cout << "\nMAQUINA #" << Maquina << endl;
        cout << " Requiere  mantenimiento (S/N)? ";
        cin >> Hay;
        Hay = toupper(Hay);

        while(Hay == 'S') {
            cout << "Tipo de mantenimiento (1-7): ";
            while(!(cin >> Tipo) || Tipo < 1 || Tipo > 7) {
                cout << "Error: Ingrese tipo v lido (1-7): ";
                cin.clear();
                cin.ignore(1000, '\n');
            }

            CantManten++;
            contadorTipos[Tipo]++; // Incrementa el contador del tipo ↩
               correspondiente

            cout << " Registrar  otro mantenimiento para esta m quina (S/↩
               N)? ";
            cin >> Hay;
            Hay = toupper(Hay);
        }
```

```
37
38          cout << "M quina #" << Maquina << ": " << CantManten << " ↩
                mantenimientos" << endl;
39          TotManten += CantManten;
40      }
41
42      // Reporte final
43      cout << "\n=== REPORTE CONSOLIDADO ===" << endl;
44      cout << "Total de mantenimientos realizados: " << TotManten << endl <<↩
            endl;
45
46      cout << "DESGLOSE POR TIPO:" << endl;
47      cout << setw(15) << "Tipo" << setw(20) << "Cantidad" << endl;
48      cout << "--------------------------------" << endl;
49      for(int i = 1; i <= TOTAL_TIPOS; i++) {
50          cout << setw(15) << i << setw(20) << contadorTipos[i] << endl;
51      }
52
53      return 0;
54  }
```

Explicación:

- `const int TOTAL_MAQUINAS = 10` y `TOTAL_TIPOS = 7` definen constantes
- Arreglo `contadorTipos` almacena acumuladores por tipo (1-7)
- Bucle `for` recorre las 10 máquinas
- Bucle `while` interno registra múltiples mantenimientos por máquina
- Validación robusta para tipos de mantenimiento (1-7)
- `toupper()` normaliza respuestas S/N

Ejemplo de ejecución:

```
SISTEMA DE REGISTRO DE MANTENIMIENTO
====================================

MAQUINA #1
¿Requiere mantenimiento (S/N)? S
Tipo de mantenimiento (1-7): 2
¿Registrar otro mantenimiento para esta máquina (S/N)? S
Tipo de mantenimiento (1-7): 5
¿Registrar otro mantenimiento para esta máquina (S/N)? N
Máquina #1: 2 mantenimientos
```

```
...

=== REPORTE CONSOLIDADO ===
Total de mantenimientos realizados: 15

DESGLOSE POR TIPO:
           Tipo            Cantidad
--------------------------------
              1                   3
              2                   4
              3                   1
              4                   2
              5                   3
              6                   1
              7                   1
```

Mejoras implementadas:

- Uso de arreglo para contadores de tipo (más escalable)
- Validación de entrada para tipos de mantenimiento
- Normalización de respuestas S/N
- Formato tabular profesional para reportes
- Constantes configurables para fácil adaptación

Análisis Industrial:

- Sistema ideal para programas de mantenimiento preventivo
- Permite identificar máquinas/problemáticas (alta frecuencia de mantenimiento)
- Detecta tipos de mantenimiento más frecuentes
- Base para análisis de costos por tipo de mantenimiento

Posibles extensiones:

- Registrar fecha y técnico responsable
- Asociar costos a cada tipo de mantenimiento
- Generar alertas por exceso de mantenimientos
- Exportar datos a sistemas CMMS (Computerized Maintenance Management Systems)
- Calcular MTBF (Mean Time Between Failures)

Estructura del programa:

1. Inicialización de contadores
2. Bucle principal por cada máquina (1-10)
3. Bucle anidado para múltiples mantenimientos
4. Validación y clasificación por tipo
5. Acumulación de estadísticas
6. Generación de reportes consolidados

Problema 24: Sistema de Monitoreo de Estaciones de Trabajo

Temática: Control de productividad industrial

Descripción: Programa que evalúa el desempeño semanal de 10 estaciones de trabajo comparando su producción contra un nivel de productividad establecido, clasificándolas como DEFICIENTE, BUENO o EXCELENTE.

Listing 24: EstacionesTrabajo.cpp

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

int main() {
    const int NUM_ESTACIONES = 10;
    string Observacion;
    char Hay;
    int Estacion, ProDia, ProdSem, NivProductividad, TotProd = 0;

    cout << "SISTEMA DE MONITOREO DE PRODUCTIVIDAD" << endl;
    cout << "=====================================" << endl << endl;

    // Entrada del nivel de productividad esperado
    cout << "Ingrese el nivel de productividad esperado por estacion: ";
    while(!(cin >> NivProductividad) || NivProductividad <= 0) {
        cout << "Error: Ingrese un valor positivo: ";
        cin.clear();
        cin.ignore(1000, '\n');
    }

    for(Estacion = 1; Estacion <= NUM_ESTACIONES; Estacion++) {
        ProdSem = 0;
        cout << "\nESTACION #" << Estacion << endl;
        cout << " Hubo  produccion hoy (S/N)? ";
```

```
27          cin >> Hay;
28          Hay = toupper(Hay);
29
30          while(Hay == 'S') {
31              cout << "Produccion del dia: ";
32              while(!(cin >> ProDia) || ProDia < 0) {
33                  cout << "Error: Ingrese valor positivo: ";
34                  cin.clear();
35                  cin.ignore(1000, '\n');
36              }
37
38              ProdSem += ProDia;
39
40              cout << " Registrar  otro dia de produccion (S/N)? ";
41              cin >> Hay;
42              Hay = toupper(Hay);
43          }
44
45          // Evaluacion de productividad
46          if(ProdSem < NivProductividad) {
47              Observacion = "DEFICIENTE";
48          } else if(ProdSem == NivProductividad) {
49              Observacion = "BUENO";
50          } else {
51              Observacion = "EXCELENTE";
52          }
53
54          // Reporte por estacion
55          cout << "\n--- RESUMEN ESTACION #" << Estacion << " ---" << endl;
56          cout << "Produccion semanal: " << ProdSem << endl;
57          cout << "Nivel esperado: " << NivProductividad << endl;
58          cout << "Evaluacion: " << Observacion << endl;
59
60          TotProd += ProdSem;
61      }
62
63      // Reporte final
64      cout << "\n=== REPORTE GENERAL ===" << endl;
65      cout << "Total producido por todas las estaciones: " << TotProd << ↩
            endl;
66      cout << "Promedio por estacion: " << fixed << setprecision(2)
67           << static_cast<double>(TotProd)/NUM_ESTACIONES << endl;
68
69      return 0;
70  }
```

Explicación:

- `const int NUM_ESTACIONES = 10` define el número de estaciones
- Bucle `for` recorre cada estación de trabajo
- Bucle `while` interno registra producción diaria
- Estructura `if-else` clasifica el rendimiento
- `toupper()` normaliza respuestas S/N
- Validación para valores positivos en producción

Ejemplo de ejecución:

```
SISTEMA DE MONITOREO DE PRODUCTIVIDAD
=====================================

Ingrese el nivel de productividad esperado por estacion: 100

ESTACION #1
¿Hubo produccion hoy (S/N)? S
Produccion del dia: 25
¿Registrar otro dia de produccion (S/N)? S
Produccion del dia: 30
¿Registrar otro dia de produccion (S/N)? N

--- RESUMEN ESTACION #1 ---
Produccion semanal: 55
Nivel esperado: 100
Evaluacion: DEFICIENTE

...

=== REPORTE GENERAL ===
Total producido por todas las estaciones: 1250
Promedio por estacion: 125.00
```

Mejoras implementadas:

- Validación de entradas para valores positivos más amigable con mensajes descriptivos
- Cálculo automático del promedio de producción
- Sistema de clasificación de rendimiento claro
- Formato profesional de salida

Análisis Industrial:

- Herramienta para gestión de eficiencia operativa
- Permite identificar estaciones problemáticas
- Base para programas de mejora continua
- Útil para balanceo de líneas de producción

Posibles extensiones:

- Registrar causas de bajo rendimiento
- Asignar metas personalizadas por estación
- Generar alertas automáticas para rendimiento deficiente
- Integrar con sistemas de incentivos laborales
- Exportar datos a herramientas de Business Intelligence

Escala de Evaluación:

Evaluación	Rango de Producción
DEFICIENTE	$Produccion < Meta$
BUENO	$Produccion = Meta$
EXCELENTE	$Produccion > Meta$

Recomendaciones:

- Analizar causas raíz para estaciones DEFICIENTES
- Replicar mejores prácticas de estaciones EXCELENTES
- Revisar periodicidad de los ciclos de evaluación
- Considerar factores externos que afecten productividad