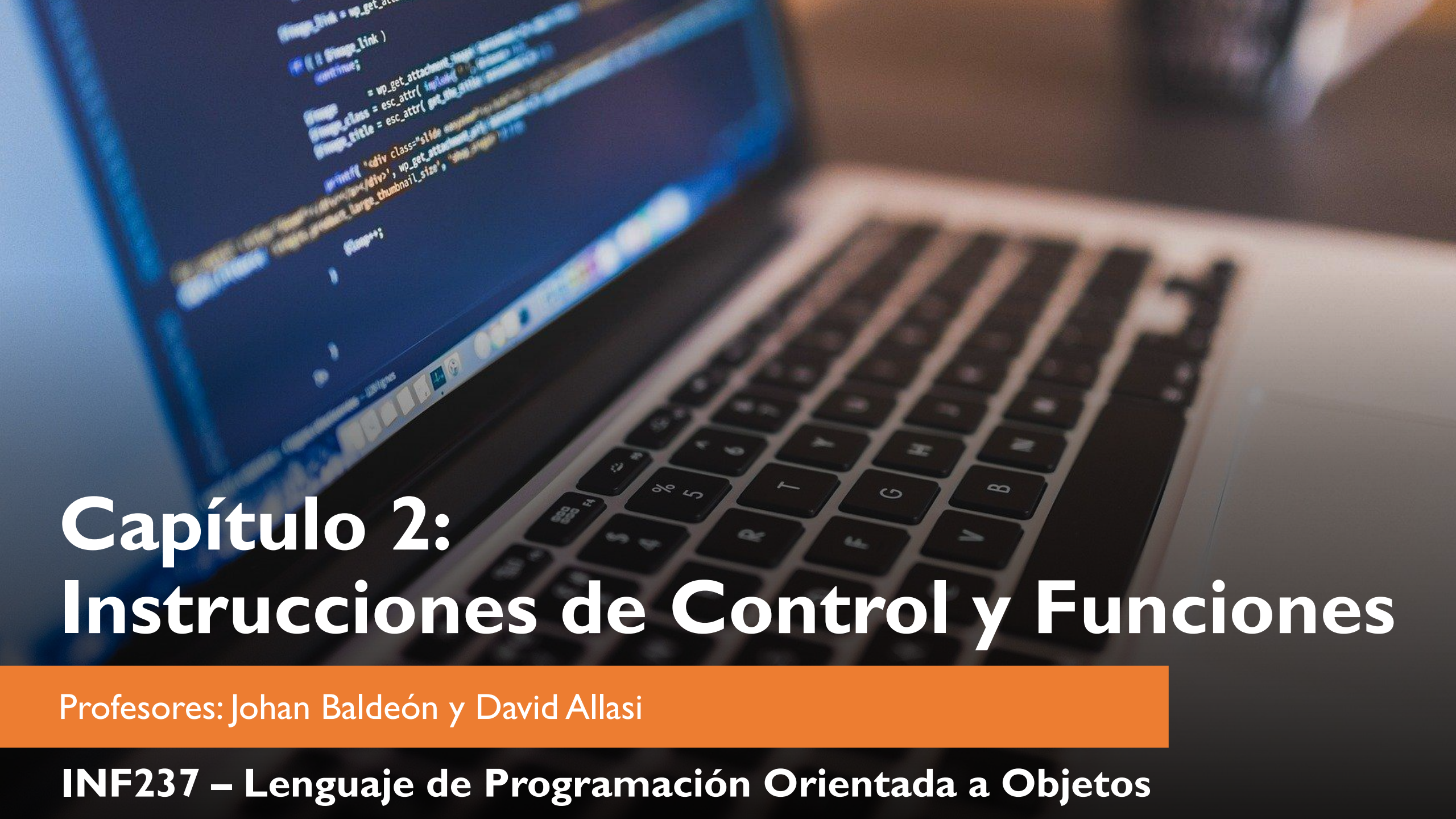


# Capítulo 2: Instrucciones de Control y Funciones

Profesores: Johan Baldeón y David Allasi

**INF237 – Lenguaje de Programación Orientada a Objetos**

# Agenda

- Introducción
  - Cómo se resuelve un problema
  - Definición de algoritmo
- Instrucciones de control:
  - Instrucciones selectivas:  if, switch.
  - Instrucciones iterativas: do/while, while, for, break, continue.
- Funciones

# Introducción

# ¿Cómo se resuelve un problema?

- Tener una completa comprensión del problema
- Nos debemos hacer tres preguntas:
  - ¿Qué salidas debemos generar?
  - ¿Qué entradas debemos ingresar para obtener las salidas solicitadas?
  - ¿Cuál es el procedimiento que debemos usar para transformar las entradas en las salidas?
- Un planteamiento correcto nos evitará perder tiempo en la implementación de algoritmos que posteriormente nos demos cuenta que son incorrectos.

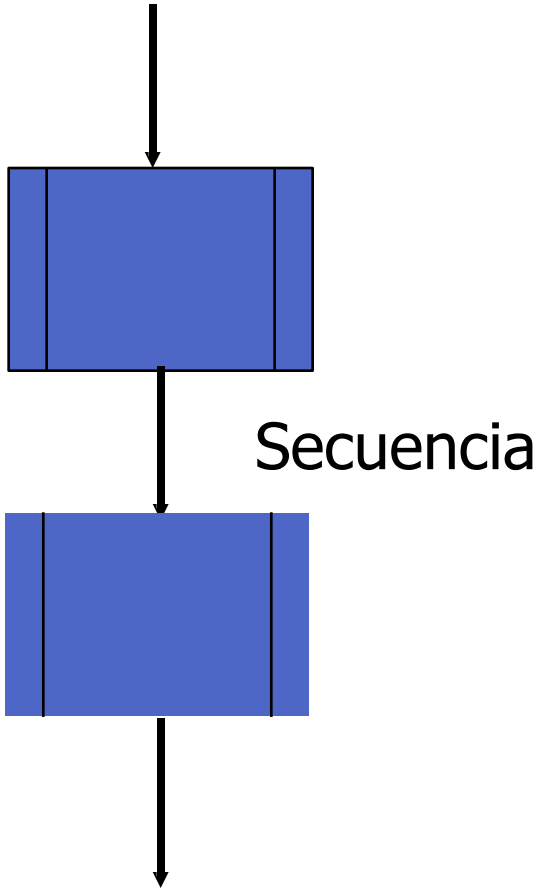
# ¿Qué es un algoritmo?

- Algoritmo: procedimiento dado en términos de:
  - Acciones que deben realizarse.
  - Secuencia de estas acciones.
- Primero se determina **qué hace** el programa.
- Luego se determina **cómo lo hace**.
- Se divide el problema en varios subproblemas que se solucionan de forma independiente (divide y vencerás). Esto se denomina diseño modular.
- Control del programa:
  - Especificar secuencia en la que los enunciados serán ejecutados.

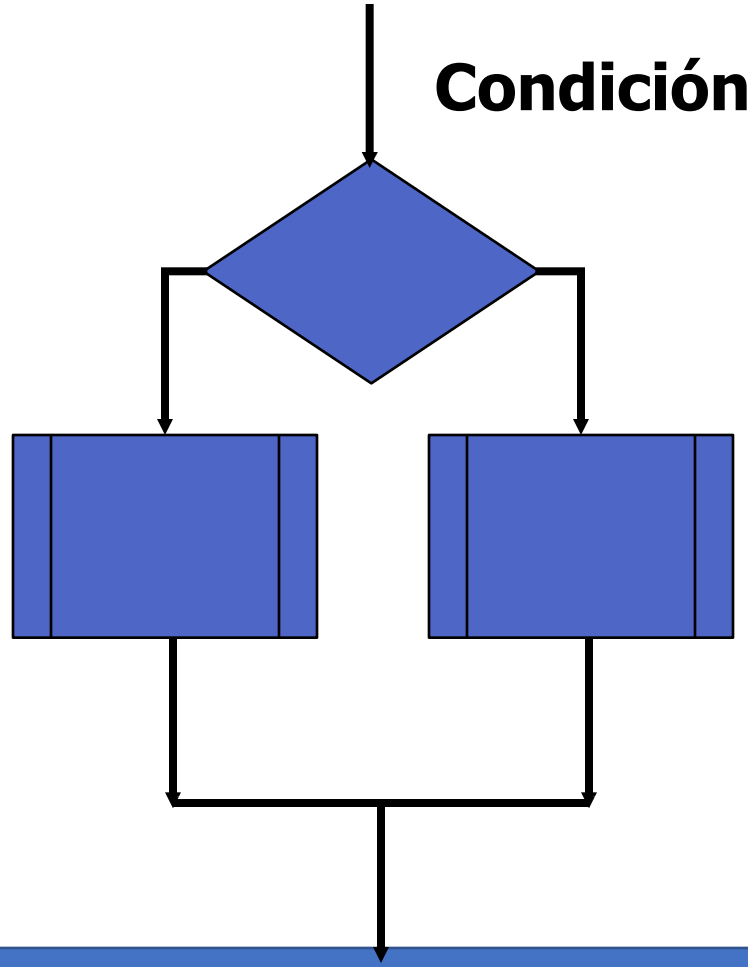
# Estructuras lógicas

- Todo algoritmo puede expresarse mediante 3 estructuras lógicas:
  - La secuencia.
  - La condición.
  - La repetición.

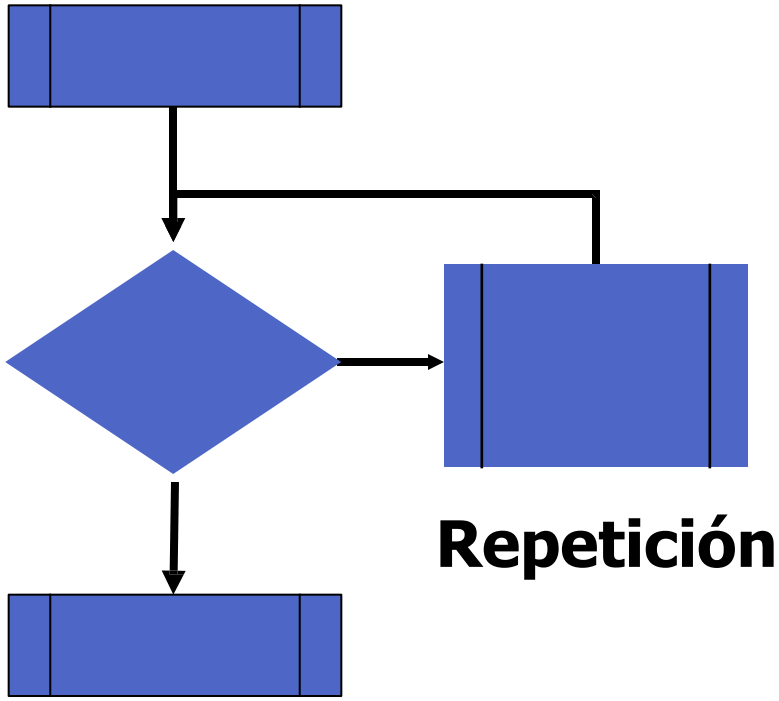
# Notación gráfica (I)



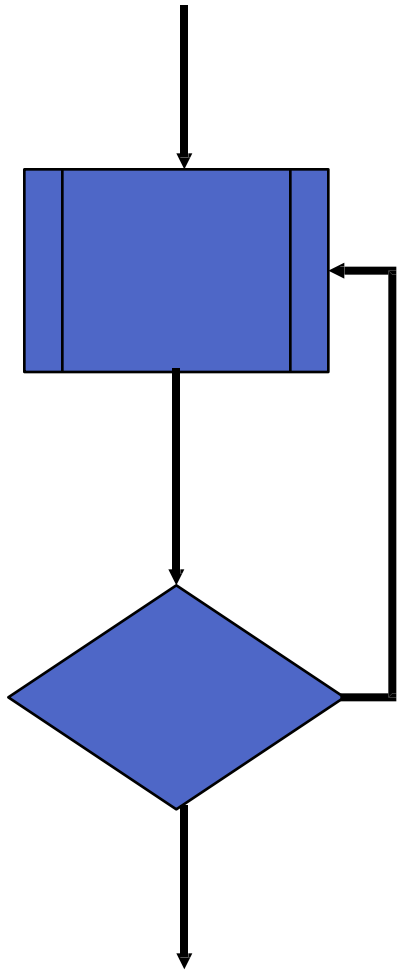
Secuencia



# Notación gráfica (2)



Repetición



# Instrucciones en C++

- En C++ existen cuatro tipos de instrucciones:
  - Selectivas
  - Iterativas
  - Salto
  - Expresiones

# Instrucciones Selectivas

# Instrucción selectiva if

- La estructura de selección ***if*** es utilizada para realizar una de dos acciones distintas en base a una condición.
- En el siguiente ejemplo se ve que el texto se imprimirá si la variable i es mayor que 0.

```
if (i > 0)
{
        cout << "i es mayor que 0";
}
```

# Instrucción selectiva if

- Estructura selectiva if - else
- Sintaxis: versión simple

```
if (expresión)
   sentencia 1;
else
   sentencia 2;
```

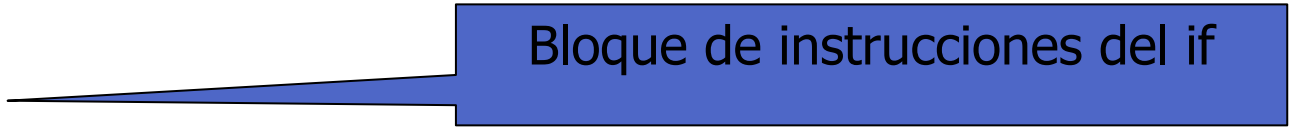
Si solo hay una sentencia luego del if, entonces no es necesario usar { }

No es necesario el uso de {} si solo se tiene una sentencia.

# Instrucción selectiva if

- Sintaxis: versión bloques

```
if (expresión)
{
  …
}
else
{
  …
}
```




# Instrucción selectiva if / else

- if
  - Sólo realiza una acción si la condición es verdadera
- if/else
  - Especifica una acción a realizarse si la condición es verdadera y otra acción si la condición es falsa

# Instrucción selectiva if / else

- Entre la parte del if y else podemos insertar una o más instrucciones if/else
- Por ejemplo:

```
if (i > 0) {
        cout << "i es mayor a cero";
}
else if (i == 0) {
        cout << "i es igual a cero";
}
else {
        cout << "i es menor a cero";
}
```

# Anidación de instrucciones: selectiva if / else

- La sentencia que se ejecuta cuando la condición en una sentencia ***if*** es verdadera también puede ser un ***if*** . Esto se llama una sentencia ***if*** anidada.
- La condición para el interior ***if*** está comprobado solamente si la condición para el exterior ***if*** es verdad. Un caso que está anidado dentro de otra también puede contener un ***if*** anidado.
- Usted generalmente puede seguir anidando ***if*** uno dentro del otro como este por todo el tiempo que lo crea pertinente.

# Anidación de instrucciones: selectiva if / else

- Por ejemplo:

```
if(letter >= 'A') { // Primera Validación
    if(letter <= 'Z') { // Segunda Validación
       cout << endl << “Ud. Ingreso una letra Mayúscula” << endl;
    }
    else {
       if(letter >= 'a') { // Primera Validación del caso contrario
              if(letter <= 'z') { // Segunda Validación
                     cout << endl << “Ud. Ingreso una letra minúscula.” << endl;
              }
       }
    }
}
```

# Ejemplos básicos de if e if / else

- Por ejemplo, el siguiente pseudocódigo:

  *Si la nota del alumno es mayor o igual a 11  Imprimir "Aprobado"*

- La condición *"la nota del alumno es mayor  o igual a 11"* tiene dos posibilidades: es  verdadera o es falsa.
- Si la condición es **verdadera**
  - Se ejecuta la acción de **imprimir** y el programa continúa al siguiente enunciado.
- Si la condición es **falsa:**
  - El enunciado de impresión es ignorado y el programa continúa al siguiente enunciado.
- Si lo traducimos al lenguaje C++:

```
if ( nota_alumno >= 11 )
  cout << "Aprobado" << endl;
```

# Ejemplos básicos de if e if / else

- Otro ejemplo en pseudocódigo:

*Si la nota del alumno es mayor o igual a 11*
    *Imprime "Aprobado"*
*de lo contrario*
    *Imprime "Jalado"*

  - Notar las convenciones de espaciado y tabulado

- Traducido a C++ :

```
if ( nota >= 11 )
    cout << "Aprobado" << endl;
else
    cout << "Desaprobado" << endl;
```

# Notación abreviada del if / else

- Operador condicional ternario (***cond* ? *X* : *Y***)
  - Usa tres argumentos (condición, valor para condición **verdadera**, valor para condición **falsa**)
  - Por ejemplo:

```
if(nota >= 11)
  aprobados = 1;
else
  aprobados = 0;
```

  - Lo anterior se puede escribir de la siguiente manera:

```
aprobados = (nota>=11)? 1:0;
```

# Notación abreviada del if / else

- También se puede combinar con otras funciones como por ejemplo cout.
- Por ejemplo:

```
(i>10) ? cout << "mayor que 10" : cout << "menor que 10";
```

# Ejercicio

Dos jóvenes, Neo y Trinity, caminan cada uno con sus respectivas velocidades en m/s. Determine la distancia que los separa luego de *t* segundos partiendo de un mismo punto, y se puede dar cualquiera de las siguientes situaciones:

*a) Se mueven en el mismo sentido.*
*b) Se mueven en sentidos contrarios.*
*c) Se mueven en forma perpendicular.*

Elabore un programa en C++ que determine la distancia que los separa luego de un determinado tiempo (t) y cualquiera de los tres posibles ángulos (ang) que forman las direcciones de sus movimientos, si las velocidades de cada uno son v1 y v2.

*Datos de entrada*: `v1, v2, t, ang`
*Salida*: `distancia`

# Ejercicio

Si parten en el mismo sentido, es decir, el ángulo es 0º:

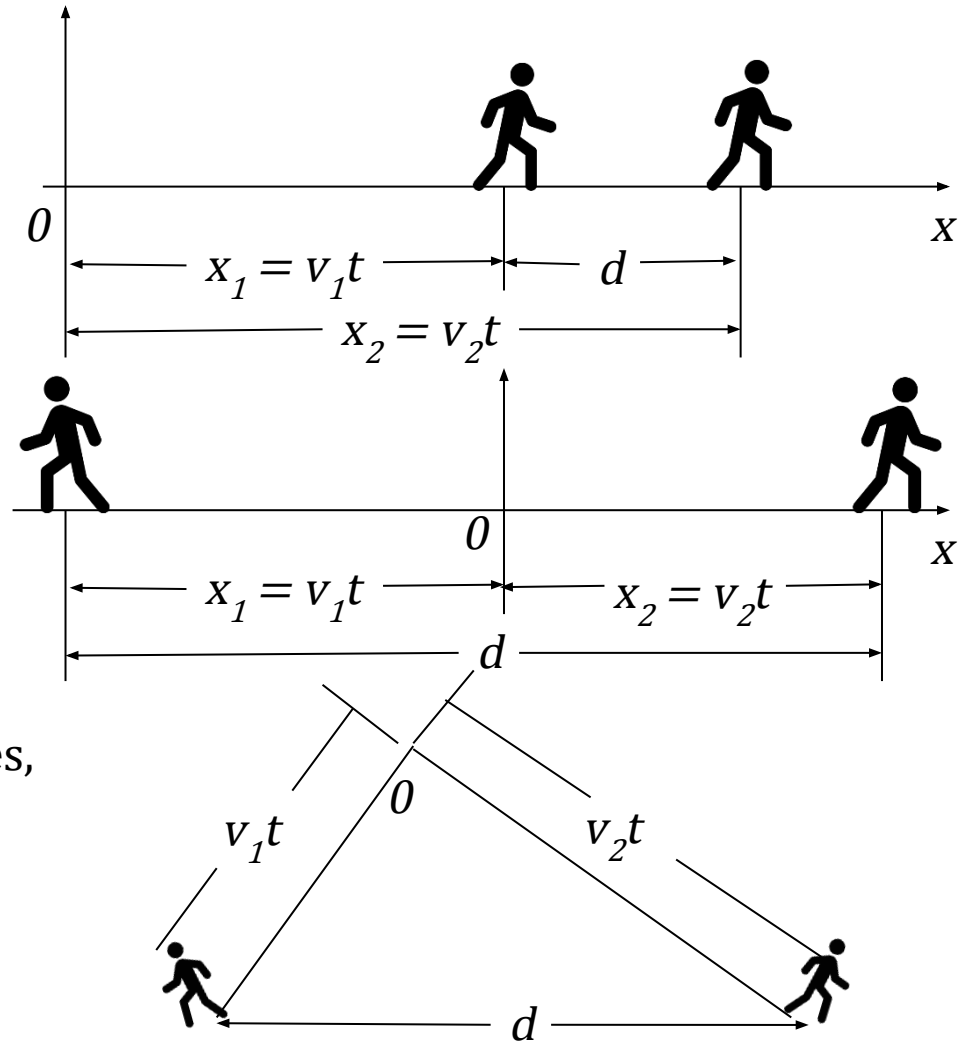
$$d = x_2 - x_1 = v_2 t - v_1 t = t|v_2 - v_1|$$

Si parten en sentidos contrarios, es decir, el ángulo es 180º:

$$d = x_1 + x_2 = v_1 t + v_2 t$$

Si parten en sentidos perpendiculares, es decir, el ángulo es 90º:

$$d = \sqrt{(v_1 t)^2 + (v_2 t)^2}$$



# Ejercicio

```
Ingrese los valores de las velocidades de Neo y Trini
ty en m/s:1.2 0.9
Ingrese el valor del tiempo en segundos:20
Escribir 'Ingrese el valor del angulo [0, 90, 180] qu
e forman las direcciones de sus movimientos:90
La separacion entre Neo y Trinity es de 30.00 metros.
```

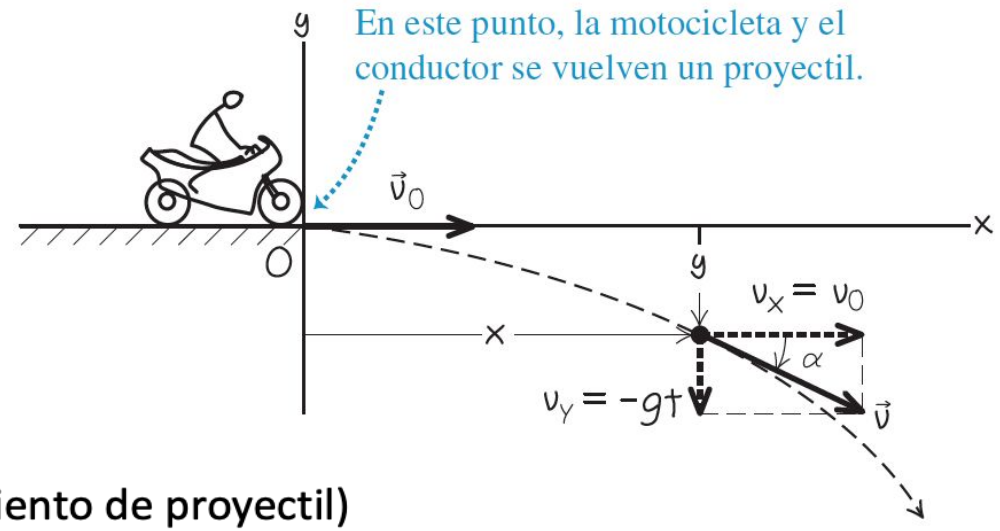
# Ejercicio

Un acróbata en motocicleta se lanza del borde de un risco. Si justo en el borde, su velocidad es horizontal con magnitud de $v_x$ m/s y la base del otro lado del precipicio está a una distancia x con una diferencia de altura y, menor a la altura inicial del borde del risco. Averigue el tiempo que le tomaría recorrer los x metros y, si el motociclista logra o no la hazaña de llegar al otro lado.

Elabore un programa en C++ que determine si logra la hazaña de llegar a la base del otro lado del precipicio.

*Datos de entrada*: `v0, x, y`

*Salidas*: `El tiempo para recorrer la distancia (x) horizontal, y si logra o no logra la hazaña.`




*Fórmulas*:

F1: $x = (v_0 \cos \alpha_0)\, t$ (movimiento de proyectil)

F2: $y = (v_0 \sin \alpha_0)\, t - \frac{1}{2} g\, t^2$ (movimiento de proyectil)

# Ejercicio

```
Ingrese la velocidad inicial (v0) en m/s:9
Ingrese la distancia (x) a saltar en m.:4.5
Ingrese la diferencia de altura por debajo del risco del otro lado (m):1.225
El tiempo para recorrer la distancia(x) horizontal es 0.50 segundos.
El motociclista logra la hazana.
```

# Instrucción de Selección Múltiple

# La instrucción de Selección Múltiple switch

- Se evalúa el valor de variable y salta a la instrucción ***"case"*** con el mismo valor. Si ningún valor coincide, salta a la sentencia default, si es que esta sentencia default existe.
- Es importante recordar la sentencia break dado que si no la colocamos, la ejecución sólo continuaría con el código adjunto a la siguiente sentencia case.
- La sentencia break se utiliza para saltar de una sentencia switch o iteración.

# La instrucción de Selección Múltiple switch

- Dentro de la instrucción switch, dos declaraciones de casos no pueden tener el mismo valor.
- La declaración por defecto ("***default***") puede ser omitido, y sólo podemos tener una alternativa por defecto.

# La instrucción de Selección Múltiple switch

- Formato
  - Series de etiquetas **`case`** y el caso opcional **`default`**

```
switch ( variable ){
  case '1':
    acciones
  case '2':
    acciones
  default:
    acciones
  }
```

  - "**`break;`**" permite salir de la estructura

# La instrucción de Selección Múltiple switch

- Por ejemplo:

```
switch (i) {
    case 1:
    case 2:
        cout << "i es igual a 1 o 2" << endl;  break;
    case 3:
        cout << "i es igual a 3" << endl;  int j = i + 1;
        cout << "j = " << j;
        break;
    default:
        cout << "i no es igual a 1, 2, o 3." << endl;  break;
}
```

# Instrucciones Iterativas

# Instrucciones de repetición controlada

- Con control al inicio del bloque:

  while()

- Con control al final del bloque:

  *do-while()*

- Con control al inicio y con contadores:

  *for()*

- Instrucciones de control:

  *break*
  *continue*

# Instrucciones de repetición controlada

Con entrada controlada

Con salida controlada


Fuente: https://www.reddit.com/r/ProgrammerHumor/comments/a5mghb/the_importance_of_knowing_how_to_correctly_use/

# La instrucción while()

- while = mientras
- Estructura
  - El programador especifica que se ejecute una sentencia o un bloque de instrucciones **mientras** cierta condición se mantenga como **verdadera.**
  - Pseudocódigo:

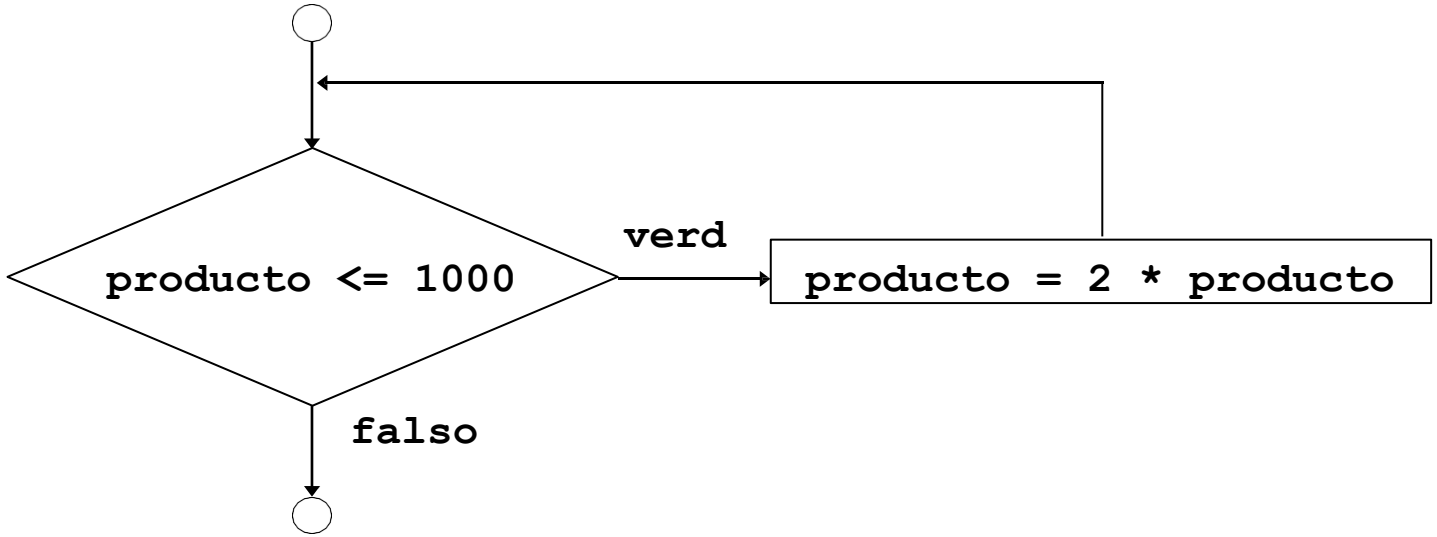
*Mientras haya más cosas en mi lista de compras*
*Comprar el siguiente ítem y sacarlo de mi lista*

  - El bucle **while** se repite hasta que la condición sea **falsa.**

# La instrucción while()

- Ejemplo:

```
int producto = 2;
while ( producto <= 1000 )
        producto = 2 * producto;
```



# Algoritmos típicos de repetición

- Validación de ingreso de datos.
- Repetición controlada por contador.
- Repetición controlada por centinela.

# Ingreso de dato con validación

- Se pide leer del teclado una variable entera mayor que cero:

```
int dato=0;
while(1){
    cout << “Ingrese dato mayor que cero: ”;
    cin >> dato;
    if(dato > 0)
        break;
}
```

# Ingreso de dato con validación (2)

- Se pide leer del teclado una variable entera mayor que cero y menor que 20:

```
int dato=0;
while(1) {
        cout << “Ingrese dato entre 0 y 20:”;
        cin >> dato;
        if(dato > 0 && dato < 20)
            break;
}
```

# Algoritmo de repetición controlada por contador

- El bucle se repite hasta que el contador alcanza cierto valor.
- Se usa cuando el número de repeticiones es conocido con anticipación.

# Ejercicio

Elabore un programa en C++ que solicite al usuario un número natural mayor o igual a 10 y menor o igual a 20. Mientras el valor ingresado sea menor que 10 o mayor que 20, se deberá mostrar el mensaje que el valor ingresado es incorrecto y deberá volver a solicitar al usuario el número natural mayor o igual a 10 y menor o igual a 20. Si el valor ingresado es correcto y es n, luego deberá mostrar en consola lo siguiente:

a. Los primeros n números naturales.

b. Los primeros n números pares.

c. Los primeros n números múltiplos de n.

```
Ingrese un numero mayor o igual a 10 y menor o igual a 20:
10
Los primeros 10 numeros naturales son:
0 1 2 3 4 5 6 7 8 9
Los primeros 10 numeros pares son:
0 2 4 6 8 10 12 14 16 18
Los primeros 10 numeros multiplos de 10 son:
0 10 20 30 40 50 60 70 80 90
```

# Algoritmo de repetición controlada por contador

- Ejemplo: Una clase de diez estudiantes toma un test. Las notas (enteros del 0 al 20) están disponibles. Determinar el promedio de las notas de la clase
- Pseudocódigo:

```
Poner total en cero
Poner contador de notas en uno
Mientras el contador de notas es menor o igual a 10  ingrese la siguiente nota.
   Sumar la nota al total
   Sumar uno al contador
Calcular el promedio de la clase como el total dividido entre diez
Imprimir el promedio de la clase
```

```cpp
// Uso de while para calcular el promedio de notas de una clase
// de 10 alumnos aplicando un algoritmo de repetición controlado
// por contador.
#include <iostream>
using namespace std;

int main(){
    int i, grade, total, average;
    /* Se inicializan las variables */
    total = 0;
    i = 1;

    while(i <= 10) { // Instrucción iterativa controlada por contador
        cout << "Ingrese nota: ";
        cin >> grade; // Se lee la nota
        total += grade;
        ++i; // Se incrementa el contador
    }
    average = total / 10;
    cout << endl << "El promedio de la clase es " << average << endl;
    return 0;
}
```

1. Se inicializan las variables

2. Se ejecuta un lazo o bucle

3. Mostrar resultados

# Ejecución del Programa

```
Ingrese nota: 19
Ingrese nota: 18
Ingrese nota: 20
Ingrese nota: 15
Ingrese nota: 14
Ingrese nota: 17
Ingrese nota: 20
Ingrese nota: 11
Ingrese nota: 8
Ingrese nota: 16

El promedio de la clase es 15
```

# Formulando algoritmos con refinamiento descendente paso a paso

- El problema se convierte en :

  Desarrolle un programa de promedios de la clase que pueda procesar un número arbitrario de notas, cada vez que se ejecute el programa.

  - Número de alumnos desconocido

  - ¿Cómo se culmina el programa?

# Formulando algoritmos con refinamiento descendente paso a paso

- Debe usarse un valor centinela
  - También conocido como valor señal, un valor sustituto o valor bandera.
  - Nos ayuda a indicar el “fin del ingreso de datos”.
  - El bucle termina cuando el usuario ingresa el valor escogido como centinela.
  - El valor centinela se escoge de tal forma que no se confunda con un valor posible (en nuestro ejemplo conviene el valor ‘n’ )

# Formulando algoritmos con refinamiento descendente paso a paso

- Refinamiento descendente paso a paso
  - Empezamos con una representación general de pseudocódigo:

    **Determinar el promedio de la clase correspondiente al examen**

  - Dividimos el algoritmo general en tareas pequeñas y las enlistamos en orden:

    **Inicializar variables**
    **Ingreso, suma y cuenta de las notas del examen**
    **Calcular e imprimir el promedio**

# Formulando algoritmos con refinamiento descendente paso a paso

- La mayoría de los programas se dividen en tres fases:
  - Inicialización: se inicializan las variables
  - Proceso: Se ingresan los valores de datos y se ajustan las variables de programa correspondientes
  - Terminación: cálculo e impresión de los resultados finales

# Formulando algoritmos con refinamiento descendente paso a paso

- Se refina la fase de inicialización

  ***Inicialización de variables***:

  Se inicializa total a cero

  Se inicializa i (contador de notas) a cero

- Refinar

  ***nota, total y contador de notas*** en

  **Ingrese la primera nota**
  **(While) mientras el usuario no haya ingresado el centinela**
  **Sumar la nota a total**
  **Sumar uno al contador de notas**
  **Ingrese la siguiente nota (puede ser el centinela)**

# Formulando algoritmos con refinamiento descendente paso a paso

- Refinar Calcular e imprimir el promedio en:

  (If ) si el contador no es igual a cero

  Poner el promedio como el total dividido entre el contador

  Imprimir el promedio

  (else) de lo contrario

  Imprimir “No se ingresaron notas”

```cpp
#include <iostream>
using namespace std;
int main(){
    char indicator('s');
    int grade, i{0}, total(0);

    while ('s' == indicator) {
        cout << "Ingrese una nota: ";
        cin >> grade;
        i++;
        total += grade;
        cout << "Va a ingresar una nueva nota (s o n)? ";
        cin >> indicator;
    }
    cout << endl << "El promedio de la clase es " << total / i << endl;
    return 0;
}
```

1. Se inicializan las variables
2. Se realiza bucle mientras el indicador sea ‘s’:
    1. Se solicita una nota
    2. Se incrementa el contador i.
    3. Se suma la nota ingresada al total.
    4. Se consulta si se va a ingresar una nueva nota o no. Es decir, indicador puede tener el valor de ‘s’ o ‘n’.
3. Se hace el cálculo de promedio y se imprime resultados.

# Estructuras de control anidadas

- Problema
  - El colegio tiene una lista de notas para 10 alumnos.
  - Escribir un programa que analice los resultados.
  - Si más de 8 alumnos aprobaron, imprimir el mensaje ”Aumentar el nivel"

# Estructuras de control anidadas

- Tener en cuenta:
  - El programa debe procesar 10 resultados de  prueba
    - Debe usarse un lazo controlado por contador.
  - Deben usarse dos contadores
    - Uno para el número de aprobados, otro para los jalados.

# Estructuras de control anidadas

- Representación general
  - Analice los resultados del examen y decida si debe aumentarse el nivel.
- Primer refinamiento
  - Inicializar variables.
  - Ingrese las diez notas y cuente los aprobados y jalados.
  - Imprimir un resumen de los resultados de exámenes y decidir si se debe aumentar el nivel.

# Estructuras de control anidadas

- Refine Inicializar variables
  - Inicializar aprobados con cero
  - Inicializar jalados con cero
  - Inicializar contador de notas con uno
- Refinar Ingrese las diez notas y cuente los  aprobados y jalados

(while) mientras el contador de alumnos es menor o igual a 10

Ingrese el siguiente resultado de examen
( if ) si el alumno aprobó incremente en uno aprobados
(else) de lo contrario, incremente en uno jalados

Incremente en uno el contador de alumnos

# Estructuras de control anidadas

- Refinar Imprimir un resumen de los resultados del examen y decidir si se debe aumentar el nivel
  - Imprimir el número de aprobados
  - Imprimir el número de jalados
  - (if) Si más de 8 alumnos aprueban, Imprimir “Levantar el nivel”.

# Ejercicio

- Desarrolle la codificación del programa aplicando el algoritmo descrito previamente.

# Más instrucciones iterativas

- Estructura de repetición do-while()
- Repetición controlada por contador
- Estructura de repetición for()
- Estructura de repetición for(): Notas y  Observaciones
- Ejemplos usando la estructura for()
- Enunciados break y continue
- Resumen de programación estructurada

# La instrucción de repetición do / while

- Estructura de repetición do/while
  - Similar a while
  - La condición para la repetición se evalúa después que el  lazo es ejecutado
    - Todas las acciones se ejecutan al menos una vez
  - Formato:

```
do {
        enunciado;
} while ( condición );
```

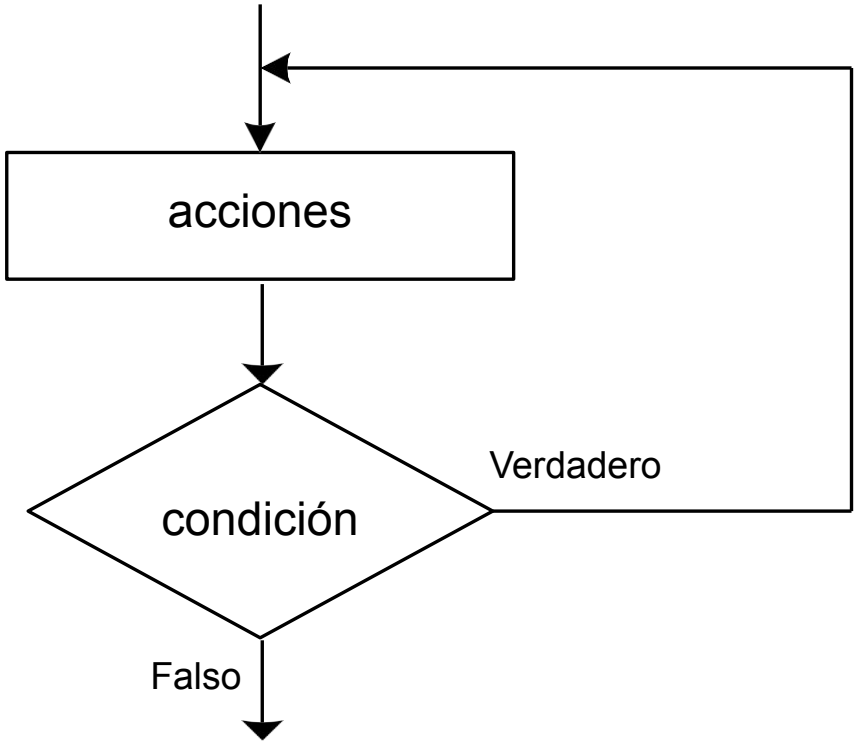
# La instrucción de repetición do / while

- Ejemplo

```
do {
        printf( "%d ", contador );
} while (++contador <= 10);
```

Imprime los enteros del 1 al 10

# Do / while: Diagrama de Flujo



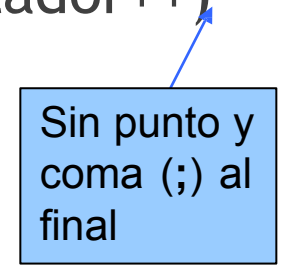
# Instrucción iterativa for

- Formato a usarse con bucles for

```
for ( inicialización; CondiciondeContinuaciondeCiclo; incremento )
        enunciado
```

- Ejemplo:

```
for( int contador = 1; contador<=10; contador++)
                    cout << contador << endl ;
```




- Imprime los enteros del uno al diez.

# Instrucción iterativa for

- La sentencia for reemplaza a bucles del tipo while del siguiente tipo:

```
inicialización;
while (CondiciondeContinuaciondeCiclo)
{
        enunciado o acción;
        incremento;
}
```

- Inicialización e incremento
  - Se separan por comas
  - Ejemplo:

```
for (i = 0, j = 0;  j + i <= 10; j++, i++)
        cout << j + i  << endl;
```

# Instrucción iterativa for (obs.)

- Expresiones Aritméticas
  - Las partes de Inicialización, Condición de continuación de bucle e Incremento pueden contener expresiones aritméticas.
  - Si x es igual a 2 e y es igual a 10

    ```
    for ( j = x; j <= 4 * x * y; j += y / x )
    ```

  - es equivalente a

    ```
    for ( j = 2; j <= 80; j += 5 )
    ```

# Instrucción iterativa for (notas)

- "Incremento" puede ser negativo (decremento):  x--.
- Si la Condición de continuación de bucle es inicialmente falsa, el enunciado no se ejecuta y el Control procede con el siguiente enunciado después de la estructura for.

# Los enunciados break y continue

- break
  - Causa una salida inmediata de una estructura tipo while, for, do/while o switch
  - La ejecución del programa continúa con la primera sentencia después de la estructura.
  - Usos comunes de break
    - Salida temprana de un lazo
    - Saltar el resto de una estructura switch

# Los enunciados break y continue

- continue
  - Salta los enunciados restantes del cuerpo de una estructura while, for o do/while.
    - Continúa con la siguiente iteración del lazo.
  - while y do/while
    - Se realiza la evaluación de la continuación del lazo inmediatamente después que continue es ejecutada.
  - for
    - La expresión de incremento es ejecutada, y luego es evaluada la condición de continuación del lazo.

# Resuelva

Se pueden clasificar a los triángulos de acuerdo a la magnitud de sus lados tal como se muestra a continuación:

| Tipos de triángulos según sus lados | | | |
|---|---|---|---|
| **Triángulo equilátero** | Tiene los tres lados iguales | $Área = \frac{\sqrt{3}}{4} \cdot a^2$ | a: lado del triángulo |
| **Triángulo isósceles** | Tiene dos lados iguales y el tercero distinto | $Área = \frac{b \cdot \sqrt{a^2 - \frac{b^2}{4}}}{2}$ | a: uno de los lados iguales,<br>b: el otro lado |
| **Triángulo escaleno** | Tiene sus tres lados distintos | $Área = \sqrt{s(s-a)(s-b)(s-c)}$<br>con $a$, $b$, $c$ los tres lados y $s$ el semiperímetro $s = \frac{a+b+c}{2}$ | |

Elabore un programa en C++ que permita ingresar la magnitud de los lados del triángulo. Si todos los datos ingresados son válidos, debe determinar el tipo de triángulo según el criterio de clasificación y calcular el área de acuerdo a la fórmula. Si los datos ingresados son inválidos, deberá avisar al usuario y volver a solicitar los datos.

# Resuelva

- Elabore un programa en C++ que muestre los primeros "n" números primos. El valor de "n" deberá ser ingresado por el usuario.
- Un número capicúa o palíndromo se refiere a cualquier número que se lee igual de izquierda a derecha que derecha a izquierda. Elabore un programa en C++ que permita identificar si un número es capicúa o no.

  Por ejemplo: 1367631 es capicúa

  1234239 no es capicúa

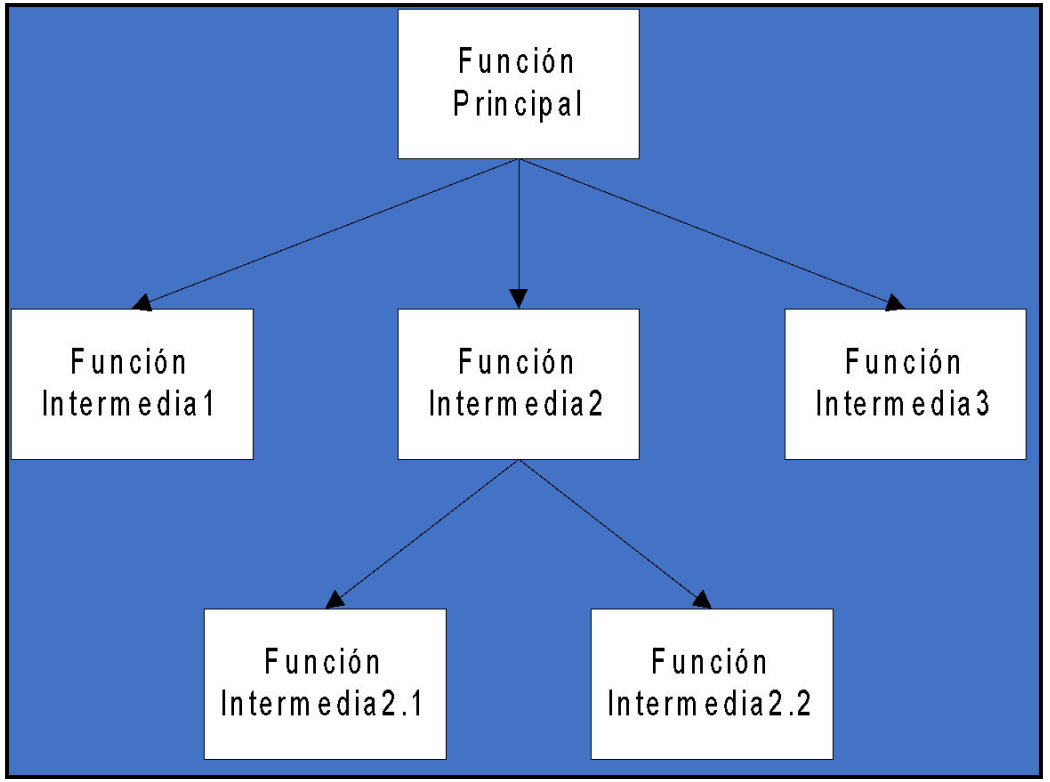
# Averigüe y resuelva

- Elabore un programa que solicite la cantidad de estudiantes de una clase, por cada estudiante solicite su código, nombre, apellido paterno, y nota del laboratorio 1, para finalmente mostrar en pantalla lo siguiente:
  - Datos del estudiante que obtuvo la menor nota y su nota.
  - Datos del estudiante que obtuvo la mayor nota y su nota.
  - Promedio de notas del laboratorio 1.
  - Cantidad de alumnos que aprobaron el laboratorio 1.
  - Porcentaje de alumnos que desaprobaron el laboratorio 1.

# Funciones

# Introducción a Funciones

- Técnica del diseño descendente
- Dividir la complejidad de un problema en módulos menos complejos.
- Cada parte es más fácil de implementar (codificar) que el programa completo.
- El programador controla ***lo que el módulo hace***.



# Componentes de los programas en C++

- Los programas en C++ están compuestos por pequeños módulos definidos por el programador para realizar determinadas acciones: ***funciones***.

- Las funciones realizan tareas específicas:
  - Operaciones matemáticas particulares
  - Manipulación de datos

- Las funciones retornan resultados calculados.

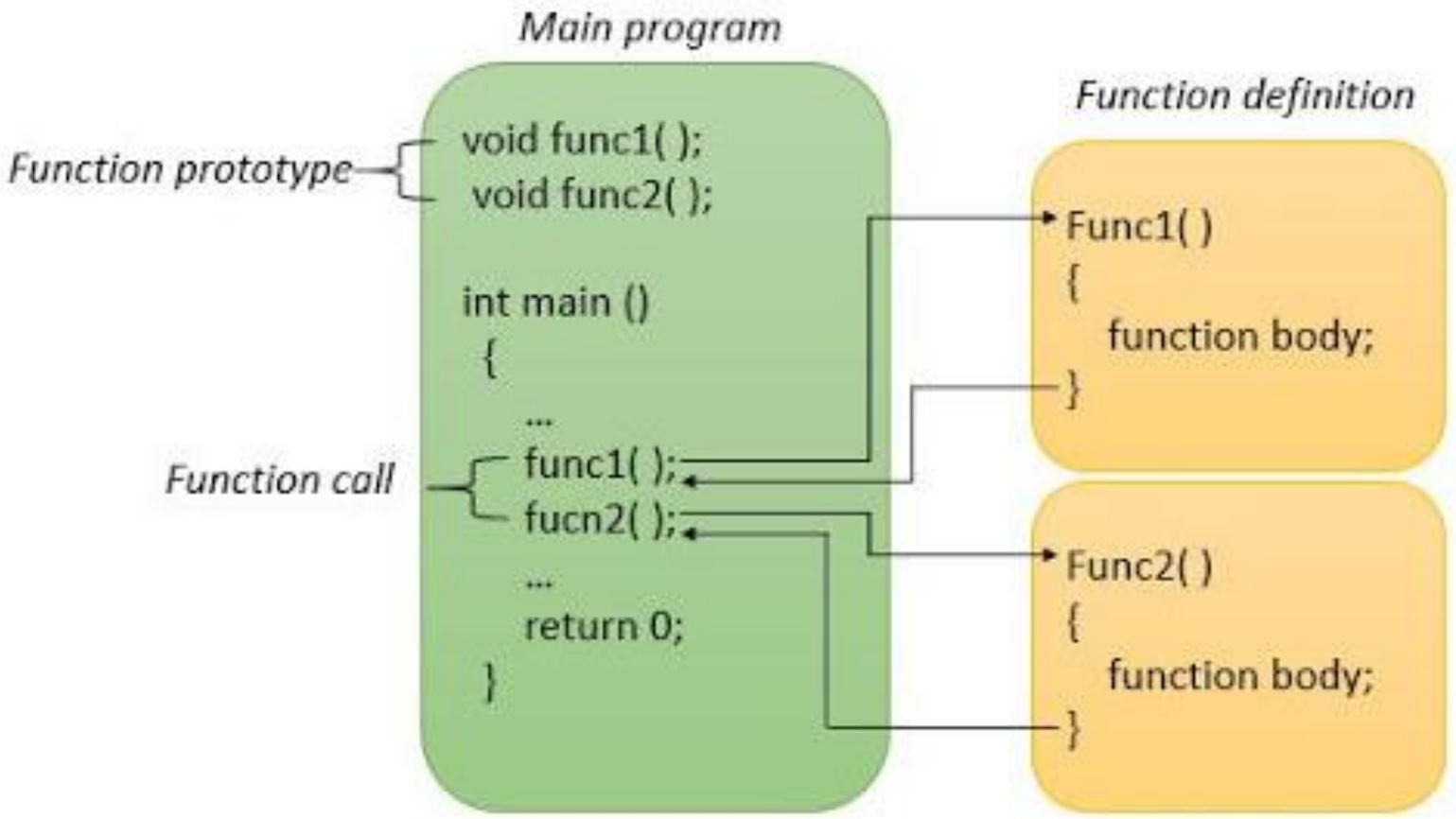
- Estas funciones se pueden ejecutar tantas veces como sea necesario y desde diversos puntos del programa.

# Cómo hacer funciones propias en C++

- Paso 1: Identificar las características de la función:
  - Qué se quiere calcular: ***Retorno*** de la función
  - Qué datos necesita para efectuar el cálculo: ***Parámetros*** de la función
  - Cómo hará el cálculo

- Paso 2:
  - Escribir la ***declaración*** de la función
  - Escribir el ***prototipo*** de la función

# Cómo hacer funciones propias en C++



# Ejemplo de función en C++

- Función Factorial()

Nombre de la función

Parámetro de la función

Tipo de Retorno

```
long Factorial(int n)
{
   long fact = 1;
   for (int i = 1; i <= dato; i++)
   {
      fact = fact * i;
   }
   return fact;
}
```

Resultado calculado por la función

# Funciones: Elementos

- Todas las funciones deben tener un nombre
  - Debe cumplir con las reglas de formación de los identificadores
- Las funciones pueden tener uno o varios parámetros
  - No confundir parámetros con variables auxiliares
- Las funciones pueden devolver un valor de algún determinado tipo
  - Se debe especificar si la función devuelve o no valor, de forma explícita

# Funciones: Declaración

- Formato de definición de una función:

```
Tipo_dato_retorno Nombre_Funcion(lista_parámetros)
{
  cuerpo_de_la_funcion
}
```

- El nombre de la función.
- El tipo de dato de retorno:
  - Si no se quiere devolver valor alguno, usar ***void***.
  - Si no se indica, se asume que es tipo ***int***.
- La lista de parámetros.

# Funciones: Uso de instrucción return

- La palabra reservada ***return*** indica el valor que la función retorna o devuelve al finalizar.
- Normalmente, es la última instrucción del cuerpo de la función.
- Cuando el compilador encuentra la palabra return, termina la ejecución de la función, ignorando las instrucciones que siguen.

# Funciones: Invocación y Parámetros

- Las funciones pueden ser usadas tanto en el programa principal (main) como en otras funciones definidas por el programador.

- Debe respetarse cantidad y tipo de parámetros que maneja la función, en todos los casos.

# Funciones: Prototipos

- Declaración anticipada al uso de las de las funciones.
- Solamente se indica el tipo de dato que devuelve, el nombre y los parámetros.
- Después se define en su totalidad.

# Usos generales de las funciones

- Uso 1: para cálculo de algún tipo
  - Se realizan operaciones (matemáticas) con sus parámetros.
- Uso 2: para determinar si se cumple o no cierta condición
  - Simular funciones boolean de VB.
- Uso 3: funciones que no retornan valor (void)
  - Hacen las veces de procedimientos de VBA.
  - Solo cuando se va a mostrar resultados.

# Llamada a Funciones: Por Valor

- Copia del parámetro se pasa a la función.
- Los cambios en la función no afectan al original.
- Se usa cuando la función no necesita modificar el parámetro. Se evitan cambios accidentales.

# Llamada a Funciones: Por Referencia

- Se pasa a la función el parámetro original.
- Los cambios en la función afectan al original.
- Requiere del uso de punteros.

# Llamada a Funciones

## Por Valor

- Copia del parámetro se pasa a la función
- Los cambios en la función no afectan al original
- Se usa cuando la función no necesita modificar el parámetro. Se evitan cambios accidentales.

## Por Referencia

- Se pasa a la función el parámetro original
- Los cambios en la función afectan al original
- Requiere del uso de punteros

# Resumen de lo aprendido

- Un algoritmo es un conjunto de instrucciones que realizadas en orden conducen a obtener la solución de un problema.
- Un algoritmo puede expresarse mediante 3 estructuras lógicas: la secuencia, la condición y la repetición.
- Una instrucción selectiva (if) es utilizada para realizar una de dos acciones distintas en base a una condición.
- En una instrucción de selección múltiple (switch) se evalúa el valor de una variable y salta a la instrucción "case" que tiene el mismo valor. Si ningún valor coincide, salta a la sentencia "default", si es que esta sentencia "default" existe.

# Resumen de lo aprendido

- Una estructura iterativa permite la repetición de una secuencia de instrucciones, también se le denomina ciclo, lazo o bucle.
- Una estructura iterativa puede ser:
  - Con control al inicio del bloque:
    `while( condición ) { … }`
  - Con control al final del bloque:
    `do { … } while( condición)`
  - Con control al inicio y con contadores:
    `for (inicialización; condición; expresión a ejecutar al término de una iteración)`
- `break` causa una salida inmediata de una estructura tipo `while, for, do/while` o `switch`.
- `continue` salta los enunciados restantes del cuerpo de una estructura `while, for` o `do/while`.

# Resumen de lo aprendido

- Las unidades de programas definidas por el usuario se conocen generalmente por el término de subprogramas para representar los módulos correspondientes; sin embargo, se denominan con nombres diferentes en los distintos lenguajes de programación.
- Así en los lenguajes C y C++ los subprogramas se denominan funciones.
- Las funciones se pueden utilizar para romper un programa en módulos de menor complejidad. De esta forma un trabajo complejo se puede descomponer en otras unidades más pequeñas que interactúan unas con otras de un modo controlado.
- Estos módulos tienen las siguientes propiedades:

  a) El propósito de cada función debe estar claro y ser simple.

  b) Una función debe ser lo bastante corta como para ser comprendida en toda su entidad.

  c) Todas sus acciones deben estar interconectadas y trabajar al mismo nivel de detalle.

  d) El tamaño y la complejidad de un subprograma se pueden reducir llamando a otros subprogramas para que hagan subtareas.

# Resumen de lo aprendido

- Las funciones definidas por el usuario son subrutinas que realizan una operación y devuelven un valor al entorno o módulo que le llamó.
- Los argumentos pasados a las funciones se manipulan por la rutina para producir un valor de retorno. Algunas funciones calculan y devuelven valores, otras funciones no.
- Una llamada a una función que devuelve un valor, se encuentra normalmente en una sentencia de asignación, una expresión o una sentencia de salida.
- Los componentes básicos de una función son la cabecera de la función y el cuerpo de la función.
- Los argumentos son el medio por el cual un programa llamador comunica o envía los datos a una función. Los parámetros son el medio por el cual una función recibe los datos enviados o comunicados.
- Cuando una función se llama, los argumentos reales en la llamada a la función se pasan a dicha función y sus valores se sustituyen en los parámetros formales de la misma.

# Resumen de lo aprendido

- Después de pasar los valores de los parámetros, el control se pasa a la función. El cálculo comienza en la parte superior de la función y prosigue hasta que se termina, en cuyo momento el resultado se devuelve al programa llamador
- Cada variable utilizada en un programa tiene un ámbito (rango o alcance) que determina en qué parte del programa se puede utilizar. El ámbito de una variable es local o global y se determina por la posición donde se sitúa la variable.
- Una variable local se define dentro de una función y sólo se puede utilizar dentro de la definición de dicha función o bloque.
- Una variable global está definida fuera de una función y se puede utilizar en cualquier función a continuación de la definición de la variable. Todas las variables globales que no son inicializadas por el usuario, normalmente se inicializan a cero por la computadora.

# Bibliografía


- P. Deitel and H. Deitel. *C how to program : with an introduction to C++*. Pearson, Boston, 2016.
- Kernighan, B., & Ritchie, D. (1988). *The C programming language* (2nd ed.). Prentice Hall
- Stroustrup, B. (2014). *A tour of C++*. Pearson Education.
- Stroustrup, B. (2014). *Programming: Principles and Practice Using C++ (2nd ed.)*. Addison-Wesley.
- Stroustrup, B. (2018). *The C++ programming language* (4th ed.). Addison-Wesley.
- Stroustrup, B. (1994). *The design and evolution of C++*. Reading, Mass: Addison-Wesley.