



# Unidad II Fundamentos de C++

M.C. Juan Carlos Olivares Rojas



# Agenda

- 2.1 Familiarización con el ambiente de programación
- 2.2 Análisis de la estructura de un programa
- 2.3 Envío de mensajes a la salida estándar
- 2.4 Concepto y manejo de variables
- 2.5 Operadores aritméticos
- 2.6 Recepción de mensajes desde la salida estándar
- 2.7 Estructuras de decisión
- 2.8 Estructuras de repetición

## 2.1 Familiarización con el ambiente de programación

- La forma de programar en C/C++ depende en la mayoría de los casos del entorno de programación (IDE) en el cual se trabaje. A estas versiones se les denomina dialectos de C/C++ y pueden ser Visual C++, Borland C++ Builder, Turbo C, Turbo C++
- En este curso se manejarán algunos entornos de programación pero la idea no es depender de ninguno de ellos

## 2.2 Análisis de la estructura de un programa

- En general un programa en C/C++ sigue la secuencia ordenada de pasos que realiza un algoritmo.
- La primeras líneas de cualquier programa en C/C++ corresponden a la inclusión de bibliotecas de funciones, seguida de definición de macros y otras directivas de preprocesamiento.

# Programa en C

```
/*Primer programa en C*/  
#include <stdio.h>  
#DEFINE PI 3.14159  
  
int void main()  
{  
    printf("El valor de PI es: %d", PI);  
    return 0;  
}
```

# Estructura de un programa en C

- Todo programa en C debe tener una función main la cual indica el punto de partida de la aplicación
- Antes de esa función se pueden declarar variables, las cuales serán visibles por todo el programa, también se pueden hacer prototipos de función

# Estructura de un programa en C

- Toda función en C debe devolver algún valor (incluso hasta vacío) y puede recibir un número determinado de parámetros.
- Todo bloque de instrucciones está delimitado por las llaves { }
- Las instrucciones siempre terminan en ;

# Estructura de un programa en C

- C es un lenguaje sensible a las mayúsculas, todas las palabras claves reservadas para el lenguaje van en minúsculas
- En general todas las constantes se ponen en mayúsculas.
- Algunos compiladores de C permiten el uso de instrucciones en ensamblador en el lenguaje.

# Características deseables de un programa

- Integridad
- Claridad
- Sencillez
- Eficiencia
- Modularidad
- Generalidad

## 2.3 Envío de mensajes a la salida estándar

- La forma de enviar información a la salida estándar (propriadamente el monitor) es a través de la función `printf` definida en la biblioteca `stdlib.h`
- `printf` recibe como argumento la cadena a mostrar en la salida estándar junto con una serie de parámetros que indican opciones auxiliares de formato e inclusión de resultados de variables

# printf

- Sintaxis:
  - printf(“cadena a mostrar”, argumentos);
- %c caracter
- %d entero decimal
- %e, %f, %g flotante
- %h entero corto
- %i entero decimal, hexadecimal u octal
- %o entero octal
- %s cadena
- %u decimal sin signo
- %x hexadecimal

# printf

- Prefijos:
  - h dato corto
  - l dato largo
  - L dato largo real
- - alineación a la izquierda
- + pone signo de + o – a los números
- 0 pone 0s en números justificados a la derecha
- # indica que los datos octales y hexadecimales sean precedidos por 0 y 0x respectivamente

# Secuencias de escape

- `\b` representa el retroceso de un espacio
- `\n` representa nueva línea
- `\r` representa retorno de carro
- `\t` representa un tabulador

## 2.4 Concepto y manejo de variables

- Variable: es una localidad de memoria en la computadora cuyo contenido cambia durante la ejecución del tiempo
- C/C++ es un lenguaje fuertemente tipado por lo que habrá que declarar primero las variables para poder utilizarlas

# Variables

- Las variables se declaran al inicio del programa o bloques de instrucción, aunque en algunos compiladores se pueden inicializar en cualquier momento.
- Debido a la gran variedad de dialectos de C/C++ se recomienda trabajar con las especificaciones ANSI, lo cual permite que nuestros programas sean portables a una gran diversidad de plataformas

# Tipos de datos

- Los tipos de datos básicos en C/C++ se muestran a continuación:
- int enteros (16 bits) //depende de la palabra del micro
- float flotantes (4 bytes)
- Char carácter (1 byte)

# Tipos de datos

- Otros tipos de datos básicos son:
- short que representa un entero corto (1 byte)
- long que representa un entero largo (4 bytes)
- double que representa un flotante con doble precisión (8 bytes)
- unsigned indica un entero sin signo
- signed suele omitirse ya que es tomado por default y representa un tipo de datos con signo

# Tipos de datos

- Una de las problemáticas que se presentan en C, es que los tipos de datos varían dependiendo de la arquitectura de computadoras, en micros de 32 bits los enteros ocupan 4 bytes, los flotantes 8.
- Se recomienda hacer uso del operador `sizeof()` que devuelve el tamaño de un tipo de datos

# Tipos de datos

- Los caracteres se representan dependiendo del código usando, generalmente es ASCII por lo que ocupan un byte, en caso contrario ocupan 2 bytes si son Unicode
- Se pueden combinar tipos de datos para crear tipos como entero corto sin signo: unsigned short

# Variables

- Las variables se declaran indicando primeramente el tipo de datos, seguido del identificador o identificadores separados con coma
- Los tipos de datos nos indican como deben ser almacenados los datos en memoria. Por ejemplo si se declara una variable:
- `int a;`

# Variables

- Esa variable a sólo podrá almacenar valores como: 0, 543, etc.
- No puede almacenar valores como: 12,234, 12.35, 10 20 30, etc.
- Las constantes son localidades de memoria cuyo contenido no cambia, se declaran con la palabra reservada const

# Variables

- El ámbito de una variable incluye el lugar desde donde se declare y estructuras anidadas.
- Una variable declarada como local a una función sólo es visible dentro de esa función. Las variables globales pueden ser accedidas desde donde sea

# Identificador

- Es el nombre que se le da una variable, por convención deben iniciar con una letra seguida por una combinación de letras, dígitos o símbolos especiales como el guión bajo (\_).
- No se pueden utilizar palabras reservadas del lenguaje en el nombre de un identificador. Los identificadores también se asocian a otros elementos como funciones.

# Palabras reservadas del lenguaje C

- auto
- break
- case
- const
- continue
- default
- do
- double
- else
- enum
- extern
- float
- for
- goto
- if
- int
- long
- register
- return
- short
- signed
- sizeof
- static
- struct
- switch
- typedef
- union
- unsigned
- void
- volatile
- while
- otros:
- asm
- entry
- far
- huge
- near
- pascal

# Palabras reservadas C++

- new
- delete
- typeid
- dynamic\_cast
- static\_cast
- reinterpret\_cast
- const\_cast
- inline
- private
- virtual
- public
- try
- wchar\_t
- bool
- mutable
- protected
- template
- typeid
- catch
- explicit
- namespace
- this
- typename
- class
- false
- throw
- using
- friend
- operator
- true

# Identificadores

- En general los identificadores deben tener un tamaño menor de 32 caracteres
- Ejemplos de identificadores válidos: suma, var1, x\_10.
- Ejemplos de identificadores inválidos: 1a1, 1918, if, \_paz, fmol@es

## 2.5 Operadores aritméticos

- + Suma
  - - Resta
  - \* Multiplicación
  - / División
  - % Módulo (resto de la división)
- 
- Al realizar operaciones aritméticas los valores numéricos se convierten al de mayor valor: flotante, entero con signo, etc.

# Operadores

- Si se antepone un tipo de datos entre paréntesis lo que se realiza es una conversión o cast implícito
- `(int)(f/53)`
- - Es un operador unario que cambia de signo a su operando

# Operadores

- ++ Es el operador de incremento. C++ es similar a:  
 $C = C + 1;$
- -- Es el operador unario de decremento. C- es equivalente a:  $C = C - 1;$
- El operador de asignación se representa con el símbolo =.  $C = 5$ . Indica que en la localidad de memoria donde está C se asigna el valor de 5

# Operadores lógicos y relacionales

- $>$  Mayor que
- $<$  Menor que
- $==$  Igual que
- $!=$  diferente de
- $>=$  mayor igual
- $<=$  menor igual
- $!$  Negación

# Operadores lógicos

- && AND lógico
- || OR lógico
- & AND a nivel de bits
- | OR a nivel de bits
  
- Otros operadores:
- ? Ternario similar a un if. Condición ?  
Acciones\_verdadero: acciones\_falso

## 2.6 Recepción de mensajes desde la salida estándar

- La forma de obtener datos para los programas es a través del teclado, el dispositivo predeterminado de entrada.
- En C, los datos se leen con la función `scanf` que tiene una sintaxis similar a la de `printf`, sólo que se ponen los modificadores:
- `scanf(“%d %f %s”, entero, flotante, cadena);`

## 2.7 Estructuras de decisión

- Las estructuras de decisión permiten bifurcar la secuencia lineal que sigue un programa. Son estructuras que limitan el código espagueti de los goto.
- Las secuencias de decisión en C son dos if, para tomas de decisiones simples y switch; para la toma de decisiones con diversas opciones.

if

if(condición lógica)

Instrucción si es verdadera la condición

if(condición lógica)

{

Conjunto de instrucción si la condición es verdadera

}

# if

if(condición lógica)

    Instrucción si la condición es verdadera

else

    Instrucción si la condición es falsa

if(condición lógica)

{

    Conjunto de instrucciones si la condición es verdadera

}

else

{

    Conjunto de instrucciones si la condición es falsa

}

# switch

```
switch(variable)
```

```
{
```

```
    case 1: conjunto de instrucciones
```

```
        break;
```

```
    ...
```

```
    case n: conjunto de instrucciones
```

```
        break;
```

```
    default: conjunto de instrucciones predeterminadas
```

```
}
```

## 2.8 Estructuras de repetición

- Las estructuras de repetición pueden ser precondicionales o postcondicionales dependiendo de si la condición se evalúa antes o después de entrar al ciclo
- Las estructuras de repetición precondicionales son: while y for; mientras que las postcondicionales es: do-while

# while

La sintaxis de while es la siguiente:

while(condición)

Instrucción a repetir si la condición es verdadera

while(condición)

{

Conjunto de instrucciones a repetir si la condición es verdadera

}

# for

for(inicialización; condición; incremento)

Instrucción a repetirse mientras condición sea verdadera

for(inicialización; condición; incremento)

{

Instrucción a repetirse mientras condición sea verdadera

}

# Do-while

do

{

Conjunto de instrucciones que se repiten al menos una vez hasta que la condición sea verdadera

}while(condición);

# ¿Preguntas?

