# Curso C++ - Clase 1

Juan Antonio Zubimendi
azubimendi@lifia.info.unlp.edu.ar

LIFIA

8 de noviembre de 2010

# Conceptos básicos

Asumimos estos conocimientos

- Programación Orientada a Objetos
- Conocimientos básicos de Java o lenguaje similar

# Introducción

En en curso vamos a

- Programar en C++
- Utilizar el entorno GNU/Linux

# Preprocesador

Como primer paso de la compilación, se invoca al preprocesador.
El preprocesador transforma el archivo de entrada en una salida a través de directivas que se encuentran en la entrada.
Estas directivas se reconocen por el # inicial.
Estas directivas no son especificas del lenguaje C++.

- #include
- #define
- #if

# #include

Esta directiva incluye el archivo especificado entre comillas o $<$ y $>$.

- *entre comillas*: El archivo lo busco en el directorio actual
- *entre* $<$ *y* $>$: El archivo se busca en los directorios de librerias del sistema

Al archivo incluido también se le aplica el preprocesador.

# #define

Nos permite definir constantes y macros.
En este curso, solo vamos a usar las constantes.
Es costumbre que las constantes definidas de esta manera se escriban todas en mayuscula.

```
#define PI 3.14159
```

# #ifdef, #endif, #if

Con estas directivas podemos hacer ciertas partes del código fuente se compilen condicionalmente.
Ejemplo:

```
#ifdef PI
   perimetro = 2 * PI * radio;
#else
   perimetro = 6.28 * radio;
#endif
```

# Hola mundo

```
#include <iostream>

int main(int argc, char *argv[]) {
    std::cout << "Hola Mundo" << std::endl;
    return 0;
}
```

# función main

Todo programa de C++ se inicia invocando a la función main, cuya forma es:

```
int main(int argc, char *argv[])
```

# Entrada Salida

La entrada salida básica provista por el lenguaje C++ la hacemos utilizando librerías estándar de C++.

- Para leer datos desde el teclado utilizamos *std::cin*.
- Para mostrar datos a la pantalla utilizamos *std::cout*.

Más adelante veremos como utilizarlo.

# Archivos Encabezados

- Contienen información sobre las diferentes librerias, funciones y constantes que usamos en nuestro programa.
- Los utilizamos con la directiva de preprocesador #include.
- Pueden formar parte de librerías estandares de C++, librerías de terceros o parte de nuestro proyecto.

# Línea de Comandos

Para compilar el ejemplo podemos escribir:

```
g++ -o holaMundo holaMundo.cpp
```

o también:

```
g++ -c -o holaMundo.o holaMundo.cpp
g++ -o holaMundo holaMundo.o
```

# Tipos de Datos básicos

- *bool*: *true* y *false*
- *char*: 8 bits, -128 a 127.
- *short*: Enteros, dependende de la plataforma.
- *long*: Enteros de 32 bits.
- *long long*: Enteros de 64 bits.
- *float*: Números racionales de 32 bits.
- *double*: Números racionales de 64 bits.
- *long double*: Números racionales de 80 bits.

También tenemos las variantes *unsigned* para la mayoria de los tipos.

# Declaracion

- Toda variable debe ser definida antes de ser usada.
- En el momento de definir una variable, se le puede asignar un valor.
- Salvo las variables globales, hay que inicializar todas las variables.

```
int entero1;
char c = 'H';
bool cond1, cond2, cond3;
```

# Asignación

- Para asignarle un valor: <variable> = <valor>;

```
int entero1;
entero1 = 25;
float x = entero1 * 2.0;
```

# Operadores

- $+$, $-$, $*$
- $/$, división entera si los tipos son enteros.
- %: modulo
- Mascaras de Bits: $\hat{}$, &, $|$,
- $==$, $!=$, &&, $||$, !
- $<$, $<=$, $>$, $>=$
- $+=$, $*=$, ...

# Funciones

Las funciones en C++ se definen de la siguiente manera:

```
<tipo retorno> <nombre_función>(<parámetros>) {
    ..
    <instrucciones>
    ..
}
```

Si no necesitamos devolver parametros usamos el tipo *void* como tipo de valor de retorno. La instrucción para devolver un valor y terminar la función es *return*.

# Funciones - Parámetros

Si la función no toma parámetros, puede usarse *()* o *(void)*. Los parámetros son una lista de parametros de la siguiente manera:

```
tipo_variable parametro1 parametro1,
tipo_variable_parametro2 parametro2...
```

**IMPORTANTE**

Todas los parámetros se pasan por valor.

# Funciones

Las funciones deben declararse antes de ser usadas.
La definición de la función sirve como declaración.
La declaración puede encontrarse en un archivo encabezado.
La declaración de una función se hace de esta manera.

```
<valor retorno> <nombre_función>(<parámetros>);
```

A esto llamaremos prototipo de la función.

# Sobrecarga de Funciones

Distintas implementaciones de la misma función se pueden definir. C++ llamará a la función que mejor coincida respecto de los parámetros. A esto se llama sobrecargar la función.
La sobrecarga de funciones debe cumplir estas reglas:

- Las diferentes implementaciones deben recibir diferentes tipos de parámetros o cantidad de los mismos.
- El tipo de retorno no es relevante para diferenciar 2 implementaciones de la misma función.

# Valores por omisión de parámetros

Se puede asignar valores predeterminados a un parámetro de una función y obviarlo en el momento de invocar a dicha función. Luego de un parámetro con valor predeterminado, no puede haber un parámetro sin valor predeterminado.

```
int potencia(int n, int pot = 2) {
  ...
}

...
int res = potencia(x + 1);
```

- Cada instrucción deberá terminar con ;
- Los bloques pueden contener 0, 1 o más instrucciones.
- Los bloques definen un nuevo alcance para las variables que definamos dentro de ese bloque, como veremos más adelante.
- En cualquier lugar donde es legal poner una instrucción, se puede insertar un bloque de instrucciones.

# if

```
if (condicion)
  <instrucción o bloque si cond. verd>
else
  <instrucción o bloque si cond. falsa>
```

# while

```
while (condición)
  <instrucción o bloque>
```

```
do
  <instruccion o bloque>
while (condición);
```

# switch

```
switch(escalar) {
case <opción1>:
case <opción2>:
    break;

case <opción3>:
    break;
default:
    break;
}
```

## N

o olvidar el *break*!

# Operador Ternario

El operador ternario nos permite escribir una expresión que puede devolver 1 de 2 valores posibles, dependendiendo de una condición.

```
condición?<valor_verdadero>:<valor_falso>;
```

# for

```
for (<inicio>; <condición fin>; <incremento>)
        <instrucción o bloque si cond. verd>
```

En <inicio> podemos definir una variable y su alcance será solamente la instrucción *for*.

# break

El comando break sale del ciclo instantáneamente. Si hay ciclos anidados, sale del ciclo de mas interno.

# continue

Corta la iteración actual del ciclo y arranca el ciclo nuevamente.

# Alcance

Llamamos alcance a los lugares donde podemos referenciar una variable.
El alcance de las variables depende donde fue definida.

- Si fue definida dentro de una función: Desde el punto donde fue definida, hasta el bloque o fin de la función.
- Si fue definida fuera de una función: Desde el punto donde fue definida, hasta el fin del bloque donde fue definida, o el final de la función

# Declaración vs. definición

- La declaración hace conocida la variable al compilador, sabe como "tratarla".
- La definición hace la reserva de memoria. Una definición puede servir como declaración.
- Toda variable o función puede estar declarada más de una vez, pero solo puede existir una sola defición de la misma.

Como hago una declaración:

- *función*: Escribo el prototipo de la función, seguida de ;.
- *variable*: antepongo la palabra clave *extern* a la definición de la variable (sin inicializarla).

# Concepto

La entrada salida de un programa en una terminal (consola) tiene tres flujos asociados:

- *stdout*: Llamada salida estándar, es donde vemos la salida de nuestro programa. Suele ser la terminal donde ejecutamos los programas.
- *stdin*: Llamada entrada estándar, es de donde se obtienen datos para nuestro programa. Suele asociarse a la entrada que usamos a través del teclado.
- *stderr*: Llamada error estándar: es donde los programas notifican errores al usuario. Es diferente a la salida estándar aunque ambas suelen escribir en la terminal.

En C++ existe una librería que nos provee este tipo de entrada salida.

# std::cout

C++ cuenta con un objeto (provisto por una librería estándar) para realizar la salida a la terminal. Este objeto es *std::cout*. Puede imprimir tanto cadenas de caracteres como variables. Para utilizarlo debemos incluir el archivo de encabezado .
Modo de uso:

```
int i = 5;
std::cout << "Hola Mundo - i = " << i << std::endl;
```

Con *std::endl* tenemos un salto de línea.

# std::cin

C++ además nos provee *std::cin* que nos permite leer desde la entrada estàndar. Para utilizarlo debemos incluir el archivo de encabezado .
Modo de uso:

```
int lado;
std::cout << "Medida del lado: ";
std::cin << lado;
std::cout << "El perimetro del cuadrado de lado  " << lad
std::cout << " es " << (lado * 4) << std::endl;
```

# std::cin - continuación

Si queremos leer cadenas de caracteres (strings) podemos usar un tipo de datos especial provisto como librería estándar. El tipo es *std::string*.

```
std::string nombre;
std::cout << "Como te llamas?";
std::cin << nombre;
std::cout << "Hola " << nombre << std::endl;
```

Un problema es que *std::cin* solo lee una palabra (se detiene ante el primer espacio). Si queremos leer una línea completa, podemos utilizar *std::getline*.

```
std::string nombre;
```

# Introducción

CMake es una herramienta para la construcción de software.

- Es Software Libre
- Es multiplataforma
- Puede construir proyectos basados en diferentes lenguajes
- Las distribuciones de GNU/Linux suelen proveerlo como paquete oficial de la distribución.
- Algunos entornos de desarrollo se integran bien con CMake

# CMake

En el transcurso del Curso usaremos CMake como herramienta de construcción. Vamos a proveer plantillas para usar y poder crear sus proyectos fácilmente.

# Como usarlo

- El código fuente de nuestro programas va en el directorio *src*
- Para compilar, preparamos haciendo:

```
mkdir build
cd build
cmake ..
```

- Ahora podemos compilar haciendo:

```
make
```