PARADIGMAS DE PROGRAMACIÓN
Licenciatura en Informática
Programador Universitario

PARADIGMAS DE PROGRAMACIÓN
Lenguaje de Programación C++

Lenguaje de Programación C++

C++

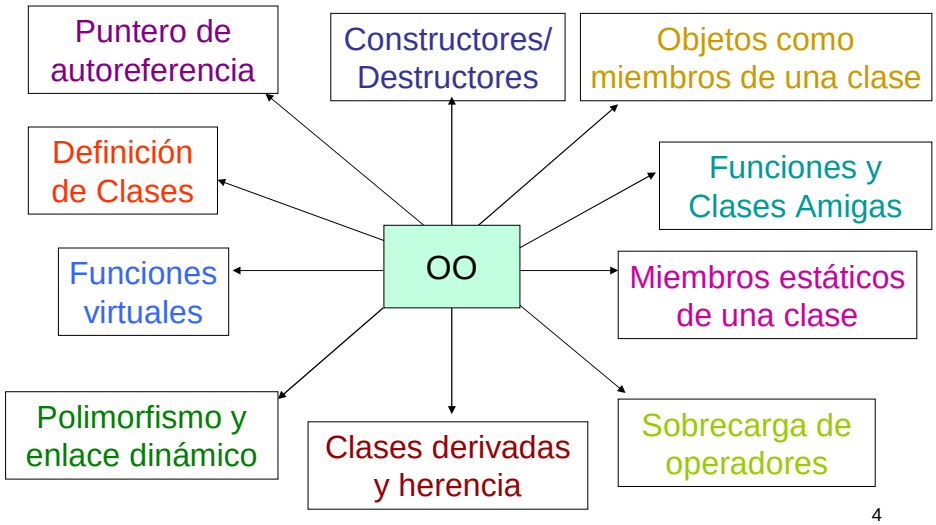
diseñado para

- **Mejorar C, manteniendo la eficiencia de C**
- **Apoyar la abstracción de datos**
- **Apoyar la POO**

Novedades de C++

3

Novedades de C++



4

Novedades de C++ no OO

COMENTARIOS EN UNA SOLA LÍNEA

- C++ permite empezar un comentario con doble barra // y lo que resta de la línea es un comentario.
- No se pueden anidar comentarios.

```
main()   /* Comentario en C */

main()  // Comentario en C++
```

5

Novedades de C++ no OO

DECLARACIONES EN CUALQUIER LUGAR DEL PROGRAMA

- Las declaraciones de variables pueden ser colocadas en cualquier parte de un enunciado ejecutable, siempre y cuando las declaraciones antecedan al uso de lo que se está declarando.

```
printf( “Ingrese dos enteros”);
int x,y;
scanf(“%d %d”,&x,&y);

for (int k=0; k<=10; k++)
     printf(“%d \n”,k);
```

6

Novedades de C++ no OO

FUNCIONES EN LINEA

- Reducen la sobrecarga por llamadas en pequeñas funciones y aumentan la velocidad del programa.
- Conveniente cuando la función se utiliza con mucha frecuencia en el programa y su código es pequeño.

Forma general:

El calificador **inline** aconseja al compilador que genere una copia del código de la función in-situ

```
inline tipo nombre (lista de parámetros con tipos){
     sentencias; //1 o mas líneas
}
```

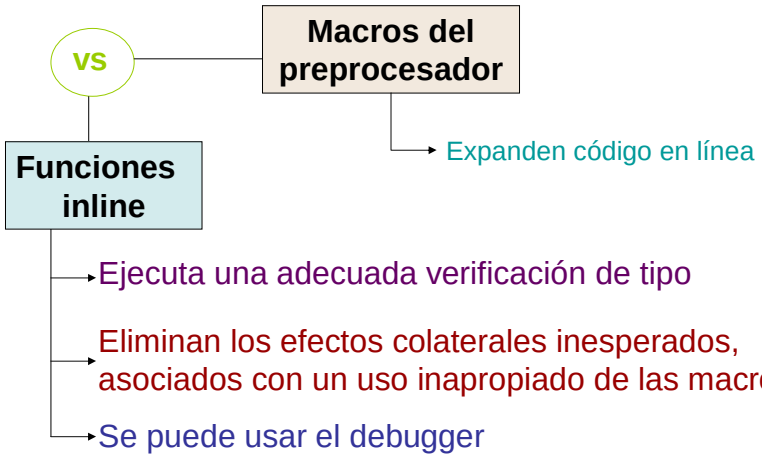
El cuerpo de la función aparece inmediatamente después de la declaración

Novedades de C++ no OO

```
//Para calcular el volumen de un cono v = π*r^2*h/3
const float PI=3.141592 ;

inline float cuad (float x) {return x*x;}

inline float volumenCono(float radio , float altura){
    return (PI * cuad(radio) * altura )/ 3.0;
}

main (){
   …
    z = volumenCono(r, h);
   …
}
```

8

Novedades de C++ no OO



9

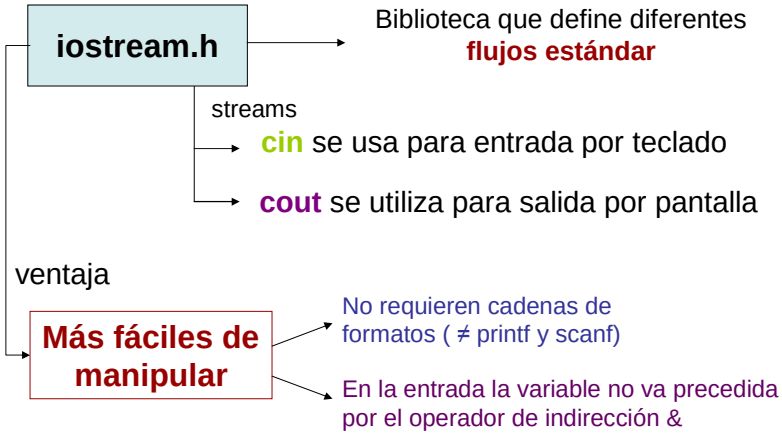
Novedades de C++ no OO

```
//Ejemplo usando MACROS:
# define cuad1(x) (x)*(x) // ok
# define cuad2(y) y*y     // cuidado!
inline int cuad3(int z) {return z*z;}

main (){
   cout << cuad1(2+3); // (2+3)*(2+3)=25
   cout << cuad2(2+3); // 2+3*2+3 = 11 ERROR!!
   cout << cuad3(2+3); // 2+3=5, 5*5=25
   return 0;
}
```

10

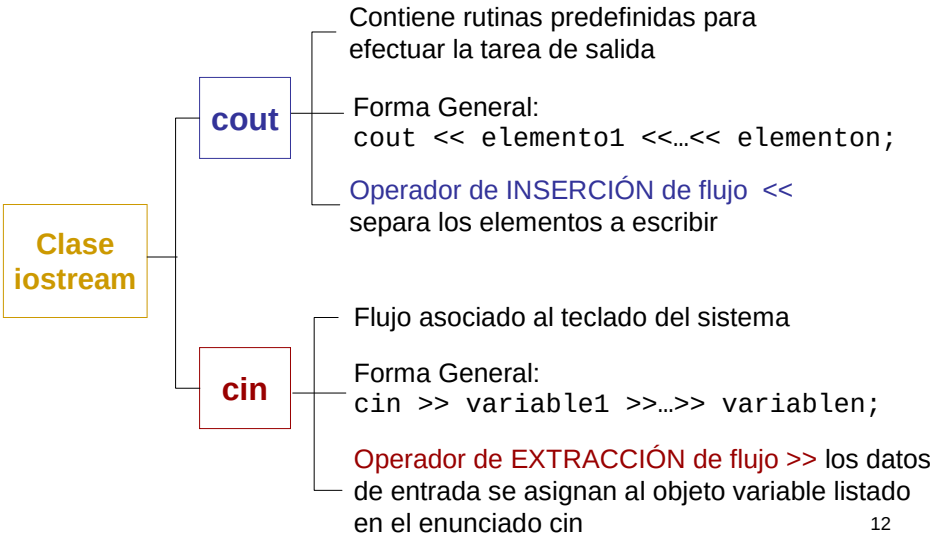
Novedades de C++ no OO

ENTRADA / SALIDA INTERACTIVA



11

Novedades de C++ no OO



12

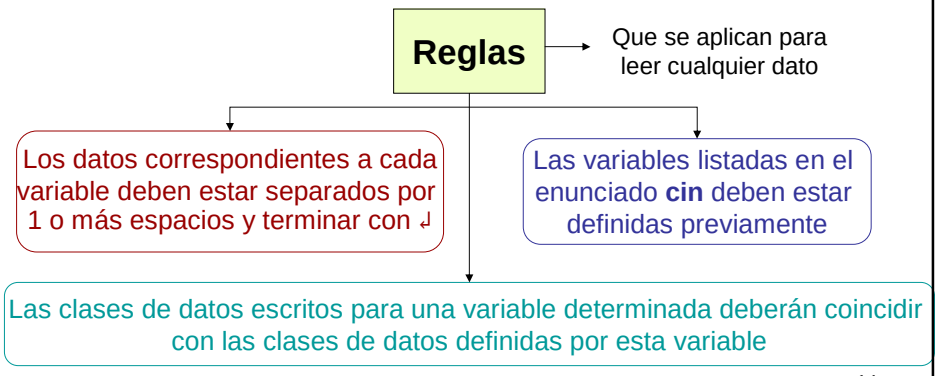
Novedades de C++ no OO

Sentencia	Salida
`cout << “Ingrese un dato:”;`	`Ingrese un dato`
`cout << ‘a’;`	`a`
`cout << 250;`	`250`
`cout << 2.89;`	`2.89`
`cout << -456;`	`-456`
`cout << 1 << 2 << 3 << 4;`	`1234`
`cout << “esta” << “palabra”;`	`estapalabra`
`cout << “esta\n” << “palabra”;`	`esta` `palabra`
`cout << “esta” << endl << “palabra”;`	`esta` `palabra`
`cout << “Edad: ” << edad ;`	`Edad: 56`

13

Novedades de C++ no OO

```
int entero;
float real;
string cadena;
cin >> entero >> real >> cadena;
```



14

Novedades de C++ no OO

SOBRECARGA DE FUNCIONES → *POLIMORFISMO ESTÁTICO*

- Se relaciona con el concepto de ***polimorfismo***, que es una de las piedras angulares de la POO.
- Permite declarar y definir varias funciones distintas que tienen el **mismo nombre pero distinta signatura**.
- En C declarar dos funciones con el mismo nombre en el mismo programa es un error de sintaxis.

15

Novedades de C++ no OO

La **signatura** de la función es diferente si tiene:
-un argumento con un tipo de dato distinto
-un número diferente de argumentos
-o ambos

```
int  funcion(string, int);
int  funcion(string, float);
int  funcion(float, string);
int  funcion(string, int, char);
char funcion(string, float); //Error
char funcion(string, float&); //Error
```

NO admite funciones que difieran sólo en el tipo del valor de retorno

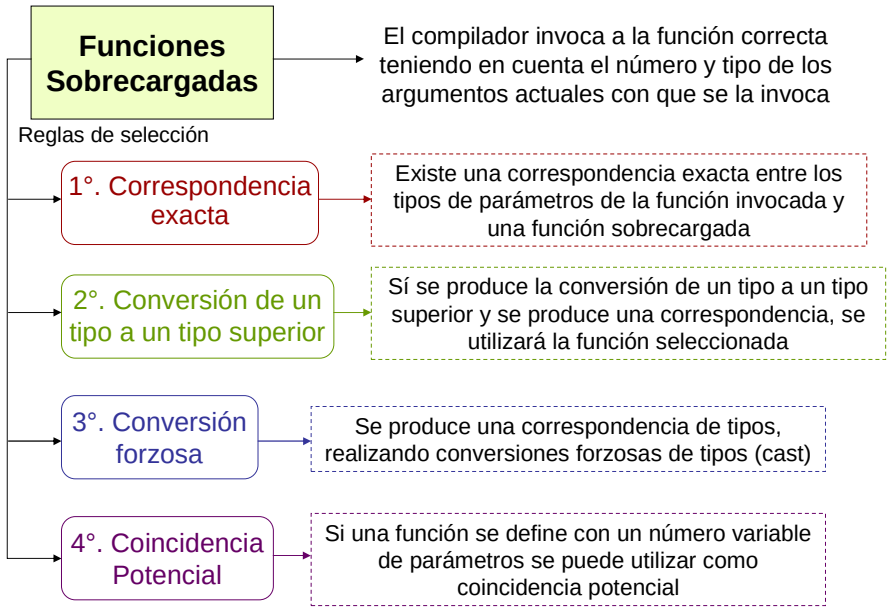
NO admite que la diferencia sea el que en una función un argumento se pasa por valor y en otra función ese argumento se pasa por referencia

Novedades de C++ no OO

La ***Homonimia de Funciones*** se utiliza para crear funciones del mismo nombre, que ejecutan tareas similares sobre tipos de datos diferentes

```
#define PI 3.1415926
// prototipos
int area (int);     // area del cuadrado
int area (int,int); // area del rectangulo
float area (float); // area del circulo

// definiciones:
int area (int lado){ return lado * lado ;}
int area (int largo, int ancho) { return largo * ancho ;}
float area (float radio){ return PI * radio * radio;}
```

17



18

Novedades de C++ no OO

Sobrecarga de OPERADORES → Para utilizar los operadores predefinidos en el lenguaje con nuevos tipos de datos

Sintaxis

```
tipoRetorno operatorXX (listaDeArgumentos);
```

Sobrecarga múltiple → Las funciones para sobrecargar un operador deben tener **argumentos diferentes**

Operadores que se pueden sobrecargar

+ - * / & ^ = ~ ¡ , > < <= >= ++ -- << >> == ¡= && += -= *= /= %= ^= &= = <<= >>= [] () -> ->* new delete

Operadores que NO se sobrecargan

?: . .* :: sizeof

19

Novedades de C++ no OO

Sobrecarga de Operadores

```
struct Fecha{
    int dia, mes, anio;
};

istream& operator>>(istream& entrada, Fecha& f)
{
    entrada>>f.dia>>f.mes>>f.anio;
    return entrada;
}
ostream& operator<<(ostream& salida, Fecha f)
{
    salida<<f.dia<<'/'<<f.mes<<'/'<<f.anio<<endl;
    return salida;
}
```

20

Novedades de C++ no OO

Sobrecarga de Operadores

```
int main() {

    Fecha f;
    cout << "Ingrese una fecha" << endl;
    cin>>f;
    cout<<f;

    ifstream archivo;
    archivo.open("d:/fechas.txt");
    archivo>>f;
    while(!archivo.eof())
    {
        cout<<f;
        archivo>>f;
    }
    archivo.close();
    return 0;
}
```

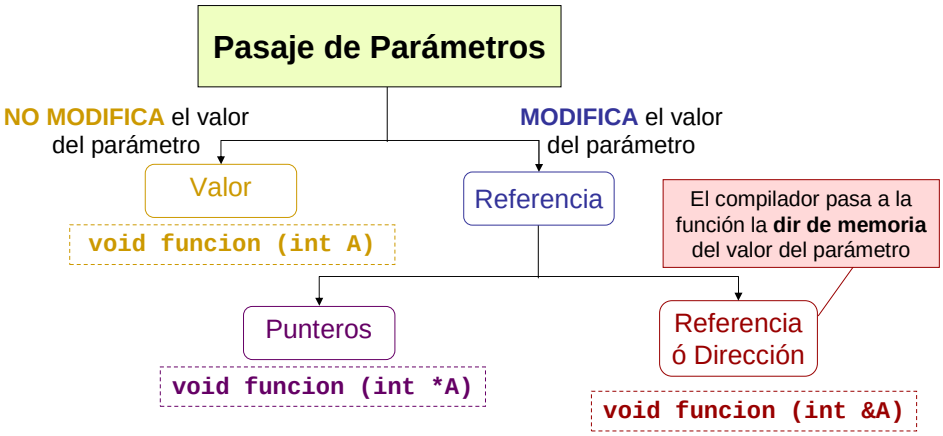

Novedades de C++ no OO

Restricciones a la sobrecarga de operadores

- No se puede:
  - Crear nuevos operadores
  - Cambiar la aridad de un operador
  - Cambiar la precedencia de los operadores
  - Cambiar la asociatividad de los operadores
  - Cambiar el modo de funcionamiento para tipos predefinidos
- Cuando se sobrecarga un operador simple, solo se sobrecarga ese operador.

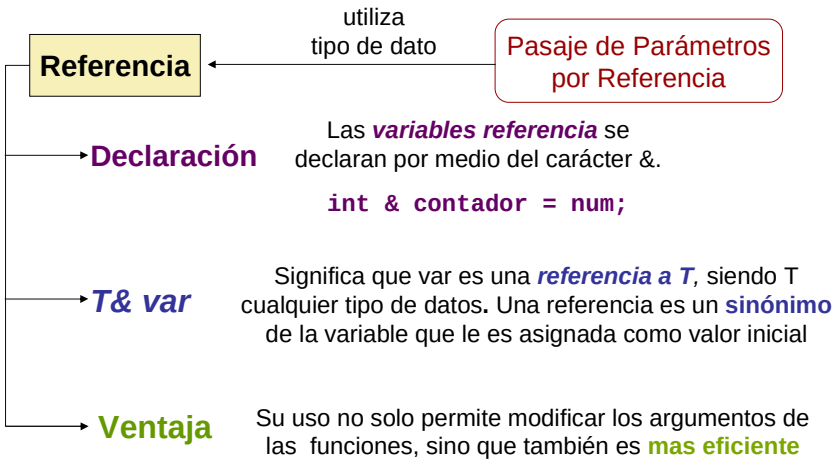
Novedades de C++ no OO

PARAMETROS POR REFERENCIA



23

Novedades de C++ no OO



24

Novedades de C++ no OO

```
//Parámetro por valor
void Fcn1(int i, int j){
        int aux = i;
        i = j;
        j = aux;
}
```

```
//Parámetro puntero
void Fcn2 (int *i, int *j){
        int aux = *i;
        *i = *j;
        *j = aux;
}
```

```
//Parámetro por referencia
void Fcn3 (int &i, int &j){
        int aux = i;
        i = j;
        j = aux;
}
```

```
main () {
   int entero1,entero2;
   …
   Fcn1(entero1,entero2);
   Fcn2(&entero1,&entero2);
   Fcn3(entero1,entero2);
   …
}
```

25

Novedades de C++ no OO

Observaciones:

- Una función puede tener una variable tipo referencia como **valor de retorno**.

```
int & maximo(int & x, int
& y){
     if (x>=y)
          return x;
      else
          return y;
}
```

- Esta función se puede invocar por ejemplo del siguiente modo:

```
maximo(x,y)=0;
```

26

Novedades de C++ no OO

ARGUMENTOS POR OMISION

Valores por defecto → Valores por omisión o ausencia o por defecto para los parámetros de las funciones

Restricciones

- **Ubicación**: Deben ser los argumentos que aparecen **más a la derecha** en la lista de parámetros de la función
- **Valor**: El valor por defecto debe ser una **expresión constante**
- **Especificación**: Los argumentos por omisión deben **especificarse en la primera ocurrencia** del nombre de la función

27

Novedades de C++ no OO

```
#include <iostream.h>

int volumen(int largo=1,int ancho=2,int altura=3){
      return largo*ancho*altura ;
}

main (){
   // pruebas
   cout<<volumen();          //resultado → 1*2*3=6

   cout<<volumen(10);        //resultado → 10*2*3=60

   cout<<volumen(10,20);     //resultado → 10*20*3=600

   cout<<volumen(10,20,30); //resultado → 10*20*30=6000

   return 0;
}
```

28

Novedades de C++ no OO

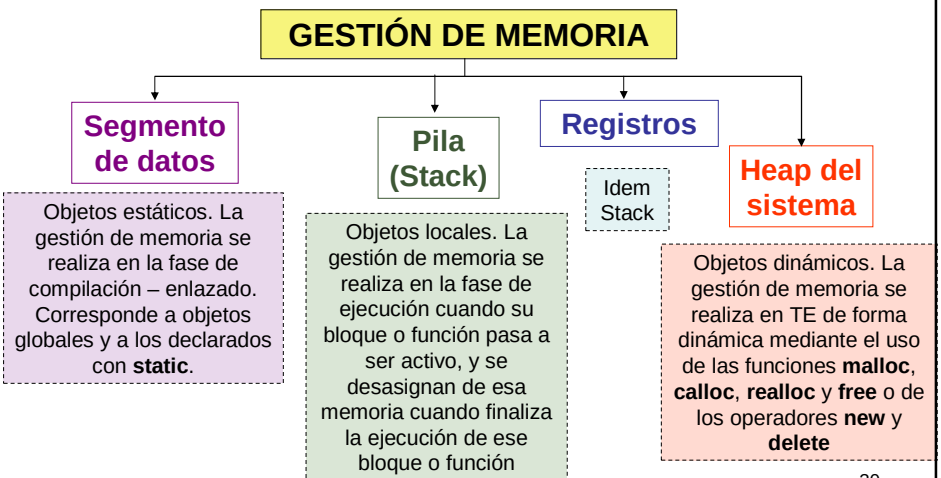
IMPORTANTE:

- Los argumentos por defecto:
  - Se **deben pasar por valor**.
  - Pueden contener valores literales o definiciones const. **No pueden ser variables.**
  - Deben colocarse al **final** del prototipo de la función.
- Después del primer argumento por defecto todos los argumentos posteriores deben incluir también valores por defecto.

29

Novedades de C++ no OO

OPERADORES NEW Y DELETE



30

Novedades de C++ no OO

Operador new

El operador **new** crea en forma automática un objeto del tamaño apropiado, si existe un constructor disponible, lo llama y retorna la dirección de la memoria asignada al objeto ó NULL si no hay espacio suficiente

Sintaxis

```
<objeto-puntero> = new <tipo-objeto>[<inicializador>];
```

```
int *pi = new int;  //asigna un entero de 2 bytes
char *pc = new char[80];//asigna un vector de 80 caract.
```

Inicializador

Se puede **inicializar el objeto asignado**.
Si no se inicializa el objeto se crea con valor indefinido

```
float *p = new float(10.4);//asigna e inicializa *p = 10.4
```

new vs malloc

```
float *p = (float *) malloc(sizeof(float));// C
float *p = new float(7);                   // C++
```

31

Novedades de C++ no OO

Operador delete

La memoria asignada desde el heap por el operador **new** es liberada por el operador **delete**.
El resultado tiene tipo **void**

Sintaxis

```
delete <objeto-puntero>
delete [] <objeto-puntero>; // caso de arrays
```

```
int *pt = new int (4);  // *pt == 4
delete pt;    //libera la memoria gestionada por new
```

Importante `delete [] nombreArrayObjetos;`

Libera la memoria asignada con new a un array de objetos de clase. El operador [] indica que debe invocar el **destructor** de la clase para cada uno de los objetos del arreglo

```
class X {/*...*/};
X *px = new X[10];  // asigna un vector de 10 objetos de X
delete[] px;
/*libera la memoria asignada al vector de la clase X e invoca al
destructor de la clase X para destruir los objetos creados*/
```