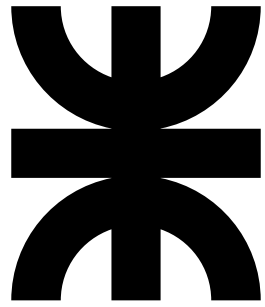
Informática II
Introducción a C++

Gonzalo F. Perez Paina



Universidad Tecnológica Nacional
Facultad Regional Córdoba
UTN-FRC

– 2019 –

Introducción

- C++ presenta mejoras en muchas de las características de C
- Brinda la posibilidad de aplicar el paradigma de programación orientado a objetos (OOP, Object Oriented Programming)
- Los programas en C++ se construyen mediante: *funciones* y tipos de datos definidos por el usuario llamados *clases*
- En el nombre C++ se incluye el operador de incremento (`++`) indicando una mejora respecto a C
- C++ fue desarrollado por Bjarne Stroustrup en los Laboratorios Bell, originalmente llamado "C con clases" (http://www.stroustrup.com/)
- En la biblioteca estándar contiene una gran colección de clases y funciones

Introducción – Estándares

- Han sido publicadas cinco revisiones del estándar C++
- Actualmente se está trabajando en la siguiente revisión C++20

Introducción – Estándares

- Han sido publicadas cinco revisiones del estándar C++
- Actualmente se está trabajando en la siguiente revisión C++20

1. En 1998 surgió el primer estándar ISO, el ISO/IEC 14882:1998, conocido informalmente como C++98
2. En 2003 se publicó una nueva versión del estándar de C++ llamado ISO/IEC 14882:2003 que corrigió problemas detectados en C++98
3. La siguiente revisión mayor del estándar se conocía informalmente como "C++0x", el cual no se liberó hasta 2001, C++11 (14882:2011). Esta versión incluye muchos agregados tanto al núcleo del lenguaje como a la biblioteca estándar
4. En 2014 se liberó el estándar C++14 como una pequeña extensión de C++11, corrigiendo algunos bugs y agregando pequeñas mejoras
5. La siguiente revisión importante fue en 2017, publicada en diciembre

Ejemplo: suma de dos enteros en C

```c
/* Programa de suma */
#include <stdio.h>

/* La función main inicia la ejecución del programa */
int main(void)
{
  int entero1; /* primer número a introducir por el usuario */
  int entero2; /* segundo número a introducir por el usuario */
  int suma; /* variable en la que se almacenará la suma */









  return 0; /* indica que el programa terminó con éxito */
} /* fin de la función main */
```

Ejemplo: suma de dos enteros en C

```c
/* Programa de suma */
#include <stdio.h>

/* La función main inicia la ejecución del programa */
int main(void)
{
  int entero1; /* primer número a introducir por el usuario */
  int entero2; /* segundo número a introducir por el usuario */
  int suma; /* variable en la que se almacenará la suma */

  printf("Introduzca el primer entero\n"); /* indicador */
  scanf("%d", &entero1); /* lee un entero */

  printf("Introduzca el segundo entero\n"); /* indicador */
  scanf("%d", &entero2); /* lee un entero */




  return 0; /* indica que el programa terminó con éxito */
} /* fin de la función main */
```

Ejemplo: suma de dos enteros en C

```c
/* Programa de suma */
#include <stdio.h>

/* La función main inicia la ejecución del programa */
int main(void)
{
  int entero1; /* primer número a introducir por el usuario */
  int entero2; /* segundo número a introducir por el usuario */
  int suma; /* variable en la que se almacenará la suma */

  printf("Introduzca el primer entero\n"); /* indicador */
  scanf("%d", &entero1); /* lee un entero */

  printf("Introduzca el segundo entero\n"); /* indicador */
  scanf("%d", &entero2); /* lee un entero */

  suma = entero1 + entero2; /* asigna el resultado a suma */

  printf("La suma es %d\n", suma); /* imprime la suma */

  return 0; /* indica que el programa terminó con éxito */
} /* fin de la función main */
```

Ejemplo: suma de dos enteros en C++

```
1  // Programa de suma
2  #include <iostream>
3
4  using namespace std;
5
6  int main()
7  {
8
9
10
11
12
13
14
15
16
17
18
19   return 0; // indica que el programa terminó de manera exitosa
20 } // fin de la función main
```

Ejemplo: suma de dos enteros en C++

```
1  // Programa de suma
2  #include <iostream>
3
4  using namespace std;
5
6  int main()
7  {
8    int entero1;
9    cout << "Introduzca el primer entero\n";
10   cin >> entero1;
11
12   int entero2; // declaración
13   cout << "Introduzca el segundo entero\n";
14   cin >> entero2;
15
16
17
18
19   return 0; // indica que el programa terminó de manera exitosa
20 } // fin de la función main
```

Ejemplo: suma de dos enteros en C++

```
1  // Programa de suma
2  #include <iostream>
3
4  using namespace std;
5
6  int main()
7  {
8    int entero1;
9    cout << "Introduzca el primer entero\n";
10   cin >> entero1;
11
12   int entero2; // declaración
13   cout << "Introduzca el segundo entero\n";
14   cin >> entero2;
15
16   int suma = entero1 + entero2; // declaración
17   cout << "La suma es " << suma << endl;
18
19   return 0; // indica que el programa terminó de manera exitosa
20 } // fin de la función main
```

Comentarios de una sola línea

- Es común utilizar comentarios cortos al final de una línea de código
- C necesita abrir y cerrar el comentario con `/*` y `*/`
- C++ permite comentarios de una sola línea con `//`
- El comentario comienza con `//` y termina al final de la línea

Comentarios de una sola línea

- Es común utilizar comentarios cortos al final de una línea de código
- C necesita abrir y cerrar el comentario con `/*` y `*/`
- C++ permite comentarios de una sola línea con `//`
- El comentario comienza con `//` y termina al final de la línea

Comentario en C

```
/* Comentario de una sola línea */
. . .

/* Comentario largo que */
/* necesita de varias líneas */
. . .

/* Otro comentario largo que
   necesita de varias líneas */
. . .
```

Comentarios de una sola línea

- Es común utilizar comentarios cortos al final de una línea de código
- C necesita abrir y cerrar el comentario con `/*` y `*/`
- C++ permite comentarios de una sola línea con `//`
- El comentario comienza con `//` y termina al final de la línea

Comentario en C

```
/* Comentario de una sola línea */
. . .

/* Comentario largo que */
/* necesita de varias líneas */
. . .

/* Otro comentario largo que
   necesita de varias líneas */
. . .
```

Comentario en C++

```
// Comentario de una sola línea
. . .

// Comentario largo que
// necesita de varias líneas
. . .
```

Entrada y salida (flujo de E/S)

Entrada y salida (flujo de E/S)

C++ brinda una alternativa a las llamadas de funciones `printf()` y `scanf()` para manejar la entrada y salida de cadenas y de tipos de datos

Entrada y salida (flujo de E/S)

C++ brinda una alternativa a las llamadas de funciones `printf()` y `scanf()` para manejar la entrada y salida de cadenas y de tipos de datos

En C

```
printf("Ingrese un entero: ");
scanf("%d", &entero);
printf("El entero es: %d\n", entero);
```

En C++

```
cout << "Ingrese un entero: ";
cin >> entero;
cout << "El entero es: "
  << entero << '\n';
```

Entrada y salida (flujo de E/S)

C++ brinda una alternativa a las llamadas de funciones `printf()` y `scanf()` para manejar la entrada y salida de cadenas y de tipos de datos

En C

```
printf("Ingrese un entero: ");
scanf("%d", &entero);
printf("El entero es: %d\n", entero);
```

En C++

```
cout << "Ingrese un entero: ";
cin >> entero;
cout << "El entero es: "
  << entero << '\n';
```

- Los operadores de *"inserción y extracción de flujo"* (`<<` y `>>`), a diferencia de las funciones de biblioteca `printf` y `scanf`, no necesitan de cadenas de formato y de especificadores de conversión para indicar los tipos de datos que son extraídos o introducidos

Entrada y salida (flujo de E/S)

C++ brinda una alternativa a las llamadas de funciones `printf()` y `scanf()` para manejar la entrada y salida de cadenas y de tipos de datos

En C

```
printf("Ingrese un entero: ");
scanf("%d", &entero);
printf("El entero es: %d\n", entero);
```

En C++

```
cout << "Ingrese un entero: ";
cin >> entero;
cout << "El entero es: "
  << entero << '\n';
```

- Los operadores de *"inserción y extracción de flujo"* (`<<` y `>>`), a diferencia de las funciones de biblioteca `printf` y `scanf`, no necesitan de cadenas de formato y de especificadores de conversión para indicar los tipos de datos que son extraídos o introducidos
- C++ tiene muchos ejemplos como este en los cuales *"sabe"* de forma automática que tipos de datos utilizar
- Cuando se utiliza el operador de extracción de flujo no necesita del operador de dirección `&` (como `scanf`)

Entrada y salida (flujo de E/S)

C++ brinda una alternativa a las llamadas de funciones `printf()` y `scanf()` para manejar la entrada y salida de cadenas y de tipos de datos

En C

```
printf("Ingrese un entero: ");
scanf("%d", &entero);
printf("El entero es: %d\n", entero);
```

En C++

```
cout << "Ingrese un entero: ";
cin >> entero;
cout << "El entero es: "
  << entero << '\n';
```

- Los operadores de *"inserción y extracción de flujo"* (`<<` y `>>`), a diferencia de las funciones de biblioteca `printf` y `scanf`, no necesitan de cadenas de formato y de especificadores de conversión para indicar los tipos de datos que son extraídos o introducidos
- C++ tiene muchos ejemplos como este en los cuales *"sabe"* de forma automática que tipos de datos utilizar
- Cuando se utiliza el operador de extracción de flujo no necesita del operador de dirección `&` (como `scanf`)

Se debe incluir el archivo de cabecera `iostream`

Declaraciones en C++

Declaraciones en C++

- En C todas las declaraciones deben aparece, dentro del bloque, antes de cualquier enunciado ejecutable

Declaraciones en C++

- En C todas las declaraciones deben aparece, dentro del bloque, antes de cualquier enunciado ejecutable
- En C++ las declaraciones puede estar en cualquier parte de un enunciado ejecutable, siempre y cuando sea antes de su uso

```
cout << "Ingrese dos enteros: ";
int x, y;
cin >> x >> y;
cout << "La suma de " << x << " y " << y <<
<< " es " << x+y << '\n';
```

Declaraciones en C++

- En C todas las declaraciones deben aparece, dentro del bloque, antes de cualquier enunciado ejecutable
- En C++ las declaraciones puede estar en cualquier parte de un enunciado ejecutable, siempre y cuando sea antes de su uso

```
cout << "Ingrese dos enteros: ";
int x, y;
cin >> x >> y;
cout << "La suma de " << x << " y " << y <<
<< " es " << x+y << '\n';
```

- Se puede también declarar variables dentro de la sección de inicialización de una estructura `for`, y mantiene el alcance hasta el final del bloque del `for`

```
for(int i = 0; i <= 5; i++)
  cout << i << '\n';
```

Crear nuevos tipos de datos

Crear nuevos tipos de datos

En C++ se puede crear nuevos tipos de datos definidos por el usuario utilizando las palabras reservadas: `enum`, `struct`, `union` y `class`

Crear nuevos tipos de datos

En C++ se puede crear nuevos tipos de datos definidos por el usuario utilizando las palabras reservadas: `enum`, `struct`, `union` y `class`

Ejemplos

```
enum Boolean {FALSE, TRUE};
struct Name {
  char firt[80];
  char last[80];
};
union Number {
  int i;
  float f;
};
```

Crear nuevos tipos de datos

En C++ se puede crear nuevos tipos de datos definidos por el usuario utilizando las palabras reservadas: `enum`, `struct`, `union` y `class`

Ejemplos

```
enum Boolean {FALSE, TRUE};
struct Name {
  char firt[80];
  char last[80];
};
union Number {
  int i;
  float f;
};
```

Crea los tipos de datos `Boolean`, `Name` y `Number`, de los cuales se puede declarar variables como

```
Boolean done = FALSE;
Name student;
Number x;
```

Referencias y parámetros de referencias

Referencias y parámetros de referencias

Muchos lenguajes de programación tienen dos formas de pasar valores a las funciones

1. llamadas por valor
2. llamadas por referencia

Referencias y parámetros de referencias

Muchos lenguajes de programación tienen dos formas de pasar valores a las funciones

1. llamadas por valor
2. llamadas por referencia

En el lenguaje C

- Todas las llamadas de función son llamadas por valor
- Las llamadas por referencia son simuladas pasando un apuntador a un objeto y obteniendo a continuación acceso al objeto des-referenciando el apuntador

Referencias y parámetros de referencias

Muchos lenguajes de programación tienen dos formas de pasar valores a las funciones

1. llamadas por valor
2. llamadas por referencia

En el lenguaje C

- Todas las llamadas de función son llamadas por valor
- Las llamadas por referencia son simuladas pasando un apuntador a un objeto y obteniendo a continuación acceso al objeto des-referenciando el apuntador

(ventajas y desventajas?)

Referencias y parámetros de referencias

Muchos lenguajes de programación tienen dos formas de pasar valores a las funciones

1. llamadas por valor
2. llamadas por referencia

En el lenguaje C

- Todas las llamadas de función son llamadas por valor
- Las llamadas por referencia son simuladas pasando un apuntador a un objeto y obteniendo a continuación acceso al objeto des-referenciando el apuntador

(ventajas y desventajas?)

Un parámetro por referencia es un *alias* de su argumento correspondiente

Referencias y parámetros de referencias

Muchos lenguajes de programación tienen dos formas de pasar valores a las funciones

1. llamadas por valor
2. llamadas por referencia

En el lenguaje C

- Todas las llamadas de función son llamadas por valor
- Las llamadas por referencia son simuladas pasando un apuntador a un objeto y obteniendo a continuación acceso al objeto des-referenciando el apuntador

(ventajas y desventajas?)

Un parámetro por referencia es un *alias* de su argumento correspondiente

- Se indica un parámetro como referencia al colocar un `&` luego del tipo de dato
- Dentro de la función se la llama a la variable directamente por su nombre

Referencias y parámetros de referencias

Referencias y parámetros de referencias

Ejemplo de referencia

```
int cuenta = 1; // declara una variable tipo entero
int &refCuenta = cuenta; // refCuenta es un alias
de cuenta (referencia)
++refCuenta; // incrementa cuenta (por medio del alias)
```

Referencias y parámetros de referencias

Ejemplo de referencia

```
int cuenta = 1; // declara una variable tipo entero
int &refCuenta = cuenta; // refCuenta es un alias
de cuenta (referencia)
++refCuenta; // incrementa cuenta (por medio del alias)
```

Ejemplo en llamada de función

```
1  int cuadradoPorValor(int );
2  void cuadradoPorReferencia(int & );
3  . . .
4  int cuadradoPorValor(int a)
5  {
6    return a *= a;
7  }
8  . . .
9  void cuadradoPorReferencia(int &refCuenta)
10 {
11   refCuenta *= refCuenta;
12 }
```

Referencias y parámetros de referencias

Ejemplo de referencia

```
int cuenta = 1; // declara una variable tipo entero
int &refCuenta = cuenta; // refCuenta es un alias
de cuenta (referencia)
++refCuenta; // incrementa cuenta (por medio del alias)
```

Ejemplo en llamada de función

```
1  int cuadradoPorValor(int );
2  void cuadradoPorReferencia(int & );
3  . . .
4  int cuadradoPorValor(int a)
5  {
6    return a *= a;
7  }
8  . . .
9  void cuadradoPorReferencia(int &refCuenta)
10 {
11   refCuenta *= refCuenta;
12 }
```

(ver ejemplo D&D 4° ed. Fig.15.5)

Referencias y parámetros de referencias

Ejemplo de referencia

```
int cuenta = 1; // declara una variable tipo entero
int &refCuenta = cuenta; // refCuenta es un alias
de cuenta (referencia)
++refCuenta; // incrementa cuenta (por medio del alias)
```

Ejemplo en llamada de función

```
1  int cuadradoPorValor(int );
2  void cuadradoPorReferencia(int & );
3  . . .
4  int cuadradoPorValor(int a)
5  {
6    return a *= a;
7  }
8  . . .
9  void cuadradoPorReferencia(int &refCuenta)
10 {
11   refCuenta *= refCuenta;
12 }
```

(ver ejemplo D&D 4° ed. Fig.15.5)

Una variable de referencia debe inicializarse en su declaración, y no puede reasignarse como alias de otra variable

Funciones `inline`

Funciones `inline`

- Utilizar el calificador `inline` en la definición antes del tipo de regreso

Funciones `inline`

- Utilizar el calificador `inline` en la definición antes del tipo de regreso
- Le "sugiere" al compilador que genera una copia del código de la función *"in situ"* a fin de evitar la llamada de función

Funciones `inline`

- Utilizar el calificador `inline` en la definición antes del tipo de regreso
- Le "sugiere" al compilador que genera una copia del código de la función *"in situ"* a fin de evitar la llamada de función

```
1  #include <iostream>
2  using namespace std;
3
4  inline double cubo( const double s) { return s * s * s; }
5
6  int main()
7  {
8    double lado;
9
10   for(int k = 1; k < 4; k++) {
11     cout << "Introduzca la longitud del un lado del cubo: ";
12     cin >> lado;
13     cout << "El volumen del cubo con lado "
14       << lado << " es " << cubo(lado) << endl;
15   } // Fin de for
16
17   return 0;
18 } // Fin de la función main
```

Sobrecarga de funciones

Sobrecarga de funciones

- C++ permite que se definan funciones con el mismo nombre mientras tengan diferente conjunto de parámetros (aunque sea en sus tipos).

Sobrecarga de funciones

- C++ permite que se definan funciones con el mismo nombre mientras tengan diferente conjunto de parámetros (aunque sea en sus tipos).
- Se utiliza por lo general para crear funciones del mismo nombre, que realizan tareas similares con tipos de datos diferentes

Sobrecarga de funciones

- C++ permite que se definan funciones con el mismo nombre mientras tengan diferente conjunto de parámetros (aunque sea en sus tipos).
- Se utiliza por lo general para crear funciones del mismo nombre, que realizan tareas similares con tipos de datos diferentes

```
1  // Uso de funciones sobrecargadas
2  #include <iostream>
3
4  using namespace std;
5
6  int cuadrado(int x) { return x * x; }
7
8  double cuadrado(double y) { return y * y; }
9
10 int main()
11 {
12   cout << "El cuadrado del entero 7 es " << cuadrado(7) << endl
13     << "El cuadrado del double 7.5 es " << cuadrado(7.5) << endl;
14
15   return 0;
16 }// fin de la función main
```

Actividad práctica

1. Escribir un programa *"Hola mundo"* en C++
2. Escribir un programa que solicite al usuario un valor `int`, uno `float` y uno `double` y los imprima en pantalla

Actividad práctica

1. Escribir un programa *"Hola mundo"* en C++

2. Escribir un programa que solicite al usuario un valor `int`, uno `float` y uno `double` y los imprima en pantalla

3. Escribir un programa que realice una división entera
   - El programa debe pedirle al usuario el valor del dividendo y del divisor
   - Si el divisor es 0 debe mostrar un mensaje de error y salir
   - La división la debe realizar una función con el siguiente prototipo

     ```
     int div_entera(int &a, int &b, int &res);
     ```

     la cual devuelve -1 si el divisor es 0, o 0 en caso contrario.

4. Escribir funciones sobrecargadas para realizar la resta entre valores enteros y de punto flotante, y un programa que verifique su funcionamiento