UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Máster Universitario en Computación en la Nube y de Altas Prestciones

Tecnología de la Programación Paralela (TPP).

# Ejercicios C++ Sesión 1

## Introducción

Esta unidad temática contiene una serie de ejercicios propuestos de C++. El objetivo consiste en familiarizarse con la sintaxis de este lenguaje, entender las características que incluye relativas a la ejecución concurrente (*multithreading*) y que, además, resulte más sencillo trabajar después con Threading Building Blocks (TBB). Se sugiere utilizar siempre el compilador de C++ de GNU (`g++`). Es posible que sea necesario utilizar la opción de compilación `--std=c++11`.

### Ejercicio 1: C++, punteros, referencias y sobrecarga.

Implementad un programa en C++ que realice lo siguiente.

1. El programa debe mostrar un mensaje por pantalla utilizando la sentencia:

   ```
   cout << "Hola mundo." << endl;
   ```

2. Realizar las siguientes acciones:

   *a*) Declarar un entero (`e`), un puntero que apunte a dicho entero (`p`) y una referencia que apunte al mismo entero también (`r`).

   *b*) Asignar un valor al entero utilizando la variable `e` (p.e. `e = 2;`) y mostrar su contenido de tres formas diferentes, es decir, utilizando `e`, `p` y `r`.

   *c*) Asignar un valor al entero utilizando el puntero `p` y mostrar su contenido de tres formas diferentes, es decir, utilizando `e`, `p` y `r`.

   *d*) Asignar un valor al entero utilizando la referencia o alias `r` y mostrar su contenido de tres formas diferentes, es decir, utilizando `e`, `p` y `r`.

3. Ahora realizad lo siguiente:

   *a*) Implementad una función (`funcion1`) que reciba un puntero a entero y lo modifique asignándole, por ejemplo, un valor constante. Realizad una llamada a esta función para ver su funcionamiento.

   *b*) Implementad otra función (`funcion2`) que realice lo mismo recibiendo una referencia en este caso. Realizad una llamada a esta función para ver su funcionamiento. Observad que esta función puede ser llamada pasándole como argumento, tanto `e` como `r`.

   *c*) Utilizad ahora el mismo nombre para las dos funciones (*sobrecarga*).

## Ejercicio 2: Clases en C++.

1. Implementad la clase `Tabla` de manera que contenga dos atributos **privados**, un puntero a vector de enteros y un entero que contiene el número de elementos almacenados. Utilizad el código facilitado (`ejercicio2.cpp`) en la carpeta de `material_cpps1` para validar el código.

2. Implementad los constructores:

```
Tabla( );          // Crea una tabla con espacio para 10 elementos
Tabla( int n );    // Crea una tabla con espacio para  n elementos
```

3. Implementad un método (`size()`) que devuelva la cantidad de elementos que hay en la tabla y que tenga la característica de ser `inline`.

4. Implementad el operador de acceso (`[i]`) que devuelva el entero que hay en la posición `i`, de manera que, si `t` es de tipo `Tabla`, se pueda obtener el valor del entero de la posición 3 mediante `t[3]`, así como asignarle un valor.

5. Implementad el constructor de copia para que no sea "superficial" (*shallow*):

```
Tabla( Tabla t );  // Crea una tabla copia de otra
```

   Cambiad el último constructor para que reciba una referencia constante.

6. Implementad el destructor:

```
~Tabla( );
```

   de manera que elimine el espacio de memoria reservado para el vector.

7. Implementad también el operador de asignación `=`.

8. Implementad la función "amiga" para mostrar los elementos de una `Tabla` con el prototipo:

```
friend ostream& operator<<( ostream&, const Tabla& );
```

   Declarad la función con a signatura anterior dentro de la clase e implementarla fuera.

9. Ahora hay que añadir al final de la función `main` el código siguiente:

```
Tabla t5{5};
for( auto &e : t5 ) {
  e = rand() % 100;
}
cout << "Tabla 5: " << t5;
```

   y hacer que funcione implementando las funciones `begin()` y `end()` correspondientes.

## Ejercicio 3: Clases en C++: más operadores.

Para realizar este ejercicio hay que utilizar el código facilitado (`ejercicio3.cpp`) en la carpeta de `material_cpps1` para validar el código.

1. Implementad la clase `NumeroR2`. Esta clase debe representar números reales en el espacio bidimensional R2 ($R^2$), por lo tanto, tendrá dos atributos que serán números reales.

   Hay que implementar los constructores habituales:

   - El constructor por defecto (sin argumentos) que inicializa a cero las componentes del número.
   - El constructor que recibe como argumento dos números reales que serán las componentes del número.

- El constructor de “copia” que recibe como argumento otro número R2.

No es necesario implementar el destructor.

2. Implementar el operador `<<` en primer lugar para poder ir comprobando en la salida la implementación de cada método u operador.

3. Implementar las operaciones aritméticas habituales de los números sobrecargando los operadores correspondientes:

   `+=` `-=` `+` `-` `++` `--` `=`

   Suponemos que el autoincremento (autodecremento) afectan a las dos componentes. Además, hay que implementar la versión prefijo y la versión sufijo de ambos. Sugerencia: implementar los operadores en el orden indicado.