Programación Orientada a Objetos

Práctica 1 - C++: clases, objetos, funciones inline, espacio de nombres, el constructor de una clase, observadores, modificadores. CMake, Make y out-of-source builds.

Datos de contacto

- Rafael Berral Soler – rberral@uco.es

- Despacho: Edificio C2, Planta 3. Departamento de Informática y Análisis Numérico (Grupo AVA).

- Horario de Tutorías:
  - Enviar mensaje privado por Moodle para concertar cita.

Prerrequisitos

- Las prácticas deberán estar contenidas en un directorio ‘poo’.
- Cada práctica estará contenida en un subdirectorio separado ‘poo/p<num. práctica>’.
  - La práctica 1 deberá estar contenida en el subdirectorio ‘poo/p1’
- Requisitos: Make, Cmake, gcc, g++, gdb, git.
  - Usando Ubuntu/Debian/WSL2:

```
# apt install make cmake gcc g++ gdb git
```

- Nota: las prácticas deberán compilar y pasar los test en ThinStation.

Clases y objetos

- Clases: actúan como el “plano” que describe una categoría de entidades, en términos de sus propiedades (datos) y comportamiento (métodos).

- Objetos: *Instancias* de la clase; cada uno de los elementos construidos a partir del “plano” definido por la clase.

Clases y objetos

- En C++, las clases se definen con la palabra reservada ‘class’.
- Ejemplo:

```
class Person{

        private: // datos internos de la clase (solo se acceden desde las funciones internas)

                std::string name_;
                int age_;
                double rank_;

        public: // funciones internas de la clase (se acceden desde el objeto y el operador . )

                Person(std::string name, int age, double rank) {name_=name;age_=age;rank_=rank;}
                std::string GetName(){return name_;}
                int GetAge(){return age_;}
                double GetRank(){return rank_;}
};
```

Clases y objetos

- Un objeto se inicializa mediante una función **constructor.**
  - Ejemplo:

```
Person(std::string name, int age, double rank){name_ = name; age_ = age; rank_ = rank;}
```

  - Es la función que se invoca al **declarar** el objeto:

```
Person p1("Juan", 32, 4.568), p2("Ana", 41, 7.371);
```

- También se pueden modificar y consultar sus valores mediante métodos **get** y **set**:
  - Ejemplos:

```
std::string GetName(){return name_;}
void GetName(std::string name){name_ = name;}
```

- **Importante** usar guardas de inclusión en nuestros ficheros de cabecera (para evitar redeclaraciones).

Espacios de nombres

- Un espacio de nombres (*namespace*) es un bloque de código con definiciones en su interior.
- Este espacio se declara mediante la palabra reservada *namespace,* seguida del nombre del espacio de nombres.
- Ejemplo:

```
namespace ns1{
int a; // Esta es la variable ns1::a
int b; // Esta es la variable ns1::b
}

namespace ns2{
float a;  // Esta es la variable ns2::a
float c;  // Esta es la variable ns2::c
}
```

Espacios de nombres

- Los objetos o clases definidos en el espacio de nombres se identifican mediante <espacio de nombres>::<clase/objeto>.
  - P. ej., `std::string` para la clase 'string' que usaremos en esta práctica, o `std::cout` para el flujo de salida por consola.

- Los espacios de nombres nos permiten reutilizar identificadores, asociados a contextos diferentes:
  - P. ej. *casa::ventana* vs. *programa::ventana*.

Espacios de nombres

- Podemos importar un espacio de nombres *completo* para "ahorrarnos" indicarlo en cada declaración:
  - P. ej. `using namespace std;`

- Generalmente esto es una mala práctica, ya que puede llevar a colisiones. Mejor importar únicamente lo que vamos a usar:
  - P. ej. `using std::cout;`

CMake

- Es una utilidad para *automatizar* el proceso de compilación.
  - Esencial para proyectos grandes, con multitud de versiones, etc.
- Permite crear los *makefiles* de forma automatizada.
- Se controla mediante ficheros **CMakeLists**:
  - Permiten crear el *makefile* a partir de declaraciones con el nombre del proyecto, información acerca del compilador, etc.

CMake

- Para esta práctica sólo usaremos un fichero CMakeLists.txt, con las siguientes líneas:
  - cmake_minimum_required(VERSION 3.10) # La versión mínima necesaria de CMake.
  - project(Práctica1) # El nombre de nuestro proyecto
  - add_executable(person-main person-main.cc) # Compilar person-main a partir de person-main.cc

- En estas prácticas trabajaremos con ***out-of-source builds***.
  - Ejecutar CMake *desde el directorio build*: $ cmake ..

CMake

- Podemos indicar también la versión del compilador necesaria.
- Para usar C++17 como nuestro estándar, incluir las siguientes líneas *antes* de la llamada a la función project():
  - set(CMAKE_CXX_STANDARD 17)
  - set(CMAKE_CXX_STANDARD_REQUIRED True)
- *Es posible* que en vuestros sistemas esto *no sea necesario*. Debéis aseguraros de que el projecto compila y ejecuta en ThinStation.

Ejercicio

- Resumen:
  - Crear un fichero ‘poo/p1/person.h’ conteniendo la definición de la clase Person.
  - Escribir una función main() en el fichero ‘poo/p1/person-main.cc’, que instancie varios objetos Person y los presente por pantalla.
  - Crear los ficheros **CMakeLists** necesarios para la compilación del proyecto.
  - Crear un directorio ‘poo/p1/build’ el cuál contendrá el código objeto y los programas compilados.
  - Invocar **CMake** desde el directorio build, pasando como argumento la ruta del directorio superior:
    - $ cmake ..
  - Compilar usando el *Makefile* generado y ejecutar.