Programación Orientada a Objetos

Práctica 2 - C++: std::vector, plantillas, iteradores, auto, range-for y excepciones. Organizando el proyecto con CMake.

CMake

- Generalmente no es una buena práctica mantener todo el código fuente en un mismo directorio.
  - Diferentes clases, librerías, etc.

- CMake nos proporciona herramientas para organizar el código en subdirectorios.

- Desde el directorio raíz del proyecto, podemos ir añadiendo nuevos directorios usando add_subdirectory(<directorio>)
  - Desde cada subdirectorio podremos añadir sub-subdirectorios creando nuevos ficheros CMakeLists.txt y usando de nuevo add_subdirectory.

CMake

- Otras instrucciones:

  - add_library(<nombre_librería> <fichero_fuente.cc> <cabecera.h>): Informa a CMake de que hay una librería "nombre_librería", y los ficheros de los que se compone.

  - target_include_directories(<nombre_librería> PUBLIC **${CMAKE_CURRENT_LIST_DIR}**): Indica donde deberán buscar los *includes* los programas que usen la librería "nombre_librería".

CMake

- Otras instrucciones:
  - add_executable(<nombre_ejecutable> <fichero_fuente.cc>):
    Indica cómo generar un ejecutable a partir de un fichero fuente .cc

  - target_link_libraries(<nombre_ejecutable> PUBLIC <nombre_librería>):
    Indica que para compilar el ejecutable "nombre_ejecutable" será necesaria la librería "nombre_librería".

  - La palabra PUBLIC denota que esas declaraciones aplican a ese "target" y a cualquier otro que emplee dicho target.

STL: *Standard Template Library*

- Una plantilla o *template* define una familia de clases o funciones, a partir de un tipo de dato*.

- En C++ disponemos de la STL, una librería de clases y algoritmos que hace uso extensivo de las plantillas.

- Por ejemplo, std::vector es una plantilla que define un *vector* a partir de un tipo de dato.
  - std::vector<int> declararía un vector de enteros.

STL: Iteradores

- Permiten *indicar* una posición dentro de una estructura de datos.
  - Es decir, nos sirve para *navegar* dentro de la estructura de datos.

- Es una *generalización* del concepto de puntero.
  - Nos permite "despreocuparnos" de la estructura de datos a recorrer.

- Con vectores (std::vector):
  - .begin(): Apunta al primer elemento del vector.
  - .end(): Apunta al final del vector.

- Podemos usarlos como "base" para calcular posiciones dentro del vector.

Excepciones

- Se producen cuando se da una situación problemática "excepcional".
  - División por 0, accesos inválidos a memoria, etc.

- Estos casos deben ser tenidos en cuenta por motivos de seguridad y calidad de código.
  - Debemos tener mecanismos para determinar qué hacer cuando se están dando estas situaciones.

- C++ implementa manejo de excepciones usando bloques **try-catch.**

try-catch

- Un bloque **try-catch** tiene la siguiente estructura:

```
try
{
        <código_a_ejecutar>
}
catch (<tipo_excepción> const &<variable_excepción>)
{
        <código_manejo>
}
```

Ejercicio

- Resumen:
  - Mover la clase "Person" del ejercicio anterior a su propio subdirectorio.
  - Ampliar la clase "Person" con un listado de *preferencias* (implementadas como un vector de cadenas).
    - Añadir los métodos para añadir nuevas preferencias y mostrar las existentes.
  - Añadir el código para modificar una preferencia a partir de su posición en la lista.
    - Probar el manejo de excepciones usando posiciones dentro y fuera del vector.
  - CUIDADO: Funciones largas deberán implementarse en ficheros fuente adicionales (no **inline**).