Curso 2024-2025
Estructura de Datos y Algoritmia
Práctica 0

Departamento de Lenguajes y Sistemas Informáticos

Condiciones de entrega

- **Plazo de entrega: desde el lunes 16 de septiembre hasta el viernes 20 de septiembre a las 23:59 horas.**
- Se debe entregar la práctica en un fichero comprimido con todos los ficheros de la siguiente manera:

Terminal
```
tar cvfz practica0.tgz CityInfo.h CityInfo.cc infotest.cc
```

Normas generales

- La práctica debe ser un trabajo original de la persona que entrega; **en caso de detectarse indicios de copia de una o más entregas se suspenderá la práctica con un 0 a todos los implicados.**
- Debes entregar la práctica exclusivamente a través del servidor de prácticas del Departamento de Lenguajes y Sistemas Informáticos (DLSI): `https://pracdlsi.dlsi.ua.es`
- Cuestiones que debes tener en cuenta al hacer la entrega:
  - El usuario y la contraseña para entregar prácticas son los mismos que utilizas en UACloud.
  - Puedes entregar la práctica varias veces, pero sólo se corregirá la última entrega.
  - No se admitirán entregas por otros medios, como el correo electrónico o UACloud.
  - No se admitirán entregas fuera de plazo.
- Tu práctica debe poder ser compilada sin errores ni warnings con el compilador de C++ existente en la distribución de Linux de los laboratorios de prácticas.
- Si tu práctica no se puede compilar su calificación será 0.
- La corrección de la práctica se hará de forma automática, por lo que es imprescindible que respetes estrictamente los textos y los formatos de salida que se indican en este enunciado.
- Al comienzo de todos los ficheros fuente entregados debes incluir un comentario con tu número de documento de identidad (NIF, NIE o pasaporte, tal como aparece en UACloud) y tu nombre. Por ejemplo:

ejemplo.cc
```
// DNI 12345678X GARCIA GARCIA, JUAN MANUEL
...
```

- Los ficheros fuente deber estar adecuadamente documentados usando comentarios en el propio código, sin utilizar en ningún caso acentos ni caracteres especiales.
- El cálculo de la nota de la práctica y su relevancia en la nota final de la asignatura se detallan en las diapositivas de presentación de la asignatura.

1. Objetivo

Esta primera práctica te permitirá empezar a familiarizarte con C++ y la declaración e implementación de clases, que entre sus elementos debe contener la **forma canónica** de la clase. La forma canónica incluye un conjunto de métodos y constructores que aseguran que una clase se comporte correctamente en términos de gestión de recursos y al hacer copias de los objetos. Permite evitar problemas de programación, y permite que una clase declarada de esta forma se pueda usar de la misma manera que cualquier tipo tradicional de C. En concreto, las operaciones que definen la forma canónica de una clase son:

- Constructores
- Destructor
- Constructor de copia
- Sobrecarga del operador asignación (=)

2. Declaración y definición de la clase CityInfo

La clase `CityInfo` almacena información relevante sobre localidades que estarán situadas en un mapa [1]. Para escribir esta clase puedes usar las plantillas para los ficheros `CityInfo.h` y `CityInfo.cc` que se proporcionan en el apartado 4. En concreto contendrá:

Atributos (parte privada):

- `int museums;`
- `int monuments;`
- `int hotels;`
- `int restaurants;`
- `bool airport;`
- `string mostImportant;`

Forma canónica (parte pública):

- `CityInfo();`
  Constructor por defecto, inicializa el string a la cadena vacía, el boolean a false, y los enteros a 0.
- `CityInfo(int mu, int mo, int h, int r, bool a);`
  Constructor que inicializa los enteros y el booleano con los valores recibidos y deja vacía la variable `mostImportant`. Si algún valor es negativo se inicializará dicho valor a 0.
- `CityInfo(const CityInfo &);`
  Constructor de copia.
- `~CityInfo();`
  Destructor que inicializa el objeto a los mismos valores que el constructor por defecto.
- `CityInfo & operator=(const CityInfo &);`
  Sobrecarga del operador asignación. Esta operación devuelve una referencia a un objeto de tipo `CityInfo` que es una copia exacta del objeto pasado por parámetro.

Métodos (parte pública):

- `bool operator!=(const CityInfo &);`
  Sobrecarga del operador `!=`. Se considera que un objeto `CityInfo` es distinto a otro cuando al menos uno de sus atributos es distinto.

[1] Más información sobre el mapa y la posición de las localidades en la siguiente práctica.

- `bool operator==(const CityInfo &);`
  Sobrecarga del operador ==. Se considera que un objeto `CityInfo` es igual a otro cuando todos sus atributos son iguales.

- `vector<int> getInterestPoints() const;`
  Devuelve los valores de las variables `museums`, `monuments`, `hotels` y `restaurants`, en ese orden, en un vector de enteros.

- `bool hasAirport() const;`
  Devuelve el valor de la variable `airport`.

- `string getMostFrequent() const;`
  Devuelve el elemento turístico del que haya más cantidad, considerando también el aeropuerto. Si hay más de uno, se devolverá el primero en orden lexicográfico. El string devuelto debe ser: "airport", "hotel", "monument", "museum" o "restaurant" .

- `string getMostImportant() const;`
  Devuelve el valor de la variable `mostImportant`.

- `void setMostImportant(string name);`
  Actualiza la variable `mostImportant` con el string pasado por parámetro.

Funciones amigas:

- `friend ostream & operator<<(ostream &, const CityInfo &);`
  Muestra en una línea la información turística almacenada en el objeto, separando todos los elementos con un espacio en blanco: número de museos, número de monumentos, número de hoteles, número de restaurantes, aeropuerto[2], monumento más importante y salto de línea.

3. Aplicación `infotest`

Escribe un fichero llamado `infotest.cc` que contendrá un `main` y que te permitirá ejecutar una pequeña aplicación que use la clase `CityInfo`, en el que tendrás que:

1. Crear un array dinámico (vector) de objetos de tipo `CityInfo`;

2. Leer de la entrada estándar enteros separados por un espacio en blanco que representan los valores de un objeto `CityInfo`. Por cada 5 enteros, se creará un objeto de tipo `CityInfo`, de forma que todas sus variables de instancia de tipo entero tomarán el valor de los 4 primeros números, y el aeropuerto se inicializará con el quinto, estableciendo su valor a falso si es 0 y verdadero en caso contrario. Puedes asumir que los valores siempre se proporcionarán en grupos de 5. Cada objeto será añadido al vector por el final;

3. Crear otro array dinámico de objetos `CityInfo` en el que aparezcan sólo los objetos del primer array que tengan aeropuerto o tengan al menos 5 hoteles;

4. Mostrar en la salida estándar, uno por línea, cada uno de los `CityInfo` del segundo array, según el formato de la operación $<<$ definida en la clase `CityInfo`.

La entrada estándar es el teclado y se leerá utilizando `cin` como flujo de entrada. En la ejecución, para terminar de leer la entrada, puedes escribir un fin de línea y `Ctrl-D` como final de fichero.

[2]Se mostrará el valor 1 si tiene aeropuerto y 0 en caso contrario.

4. Plantillas para los ficheros a implementar

CityInfo.h

En este fichero se hace la declaración de la clase:

```
#ifndef CITYINFO_H // Directiva de preprocesamiento para evitar errores
#define CITYINFO_H //   si se incluye dos veces el fichero

#include <iostream>
#include <vector>

using namespace std;

class CityInfo {
  // Funciones amigas (friend)
  friend ostream & operator<<(ostream &, const CityInfo &);

  private:
  // Atributos y metodos privados

  public:
  // Declaracion de operaciones canonicas, operadores y metodos
};

#endif
```

CityInfo.cc

En este fichero se hace la implementación de las operaciones canónicas, operadores y métodos:

```
#include "CityInfo.h"

// Constructores
CityInfo::CityInfo() { ... }
CityInfo::CityInfo(int mu, int mo, int h, int r, bool a) { ... }
CityInfo::CityInfo(const CityInfo &t) { ... }

// Destructor
CityInfo::~CityInfo() { ... }

// Operadores
CityInfo & CityInfo::operator=(const CityInfo &t) { ... }
bool CityInfo::operator!=(const CityInfo &t) { ... }
bool CityInfo::operator==(const CityInfo &t) { ... }

// Metodos
vector<int> CityInfo::getInterestPoints() const { ... }
bool CityInfo::hasAirport() const { ... }
string CityInfo::getMostFrequent() const { ... }
string CityInfo::getMostImportant() const { ... }
void CityInfo::setMostImportant(string s) { ... }

// Sobrecarga operador salida
ostream & operator<<(ostream &os, const CityInfo &c){
  os << c.museums << " "  << c.monuments <<  " "  << c.hotels <<  " "
     << c.restaurants << " "  << c.airport <<  " "  << c.mostImportant << endl;
  return os;
}
```

ejemplo.cc

Este fichero sirve para probar las operaciones y métodos de la clase `CityInfo`, aunque no es imprescindible para la práctica y no se debe entregar. Es simplemente un ejemplo de uso de algunas operaciones y operadores, pero te puede servir de ayuda para crear la aplicación `infotest.cc`:

```
#include <iostream>
#include "CityInfo.h"
using namespace std;

int main() {
   // Crea varios objetos de tipo CityInfo
   CityInfo A, B(1,0,0,1, true), C(0, 0, 0, 0, false);

   if (A!=B) cout << "Los objetos CityInfo A y B son diferentes" << endl; // true
   if (A!=C) cout << "Los objetos CityInfo A y C son diferentes" << endl;
   else cout << "Los objetos CityInfo A y C son iguales" << endl; // Son iguales

   CityInfo D(B); // Crea el objeto CityInfo D como una copia de B

   cout << D.getMostFrequent() << endl; // Muestra el elemento mas frecuente en D
   cout << D << endl; // Muestra el objeto CityInfo D con el formato definido
}
```

5. Compilación y enlace

Puedes compilar cada uno de los ficheros `.cc` con estas instrucciones:

Terminal

```
g++ -c CityInfo.cc
g++ -c ejemplo.cc
g++ -c infotest.cc
```

Es bueno acostumbrarse a utilizar otras opciones de compilación, como optimizadores para aumentar la velocidad `(-O3)` y warnings `(-Wall)`:

Terminal

```
g++ -O3 -Wall -c CityInfo.cc
```

Por último, enlaza los ficheros objeto `CityInfo.o` y `ejemplo.o` para obtener el ejecutable `ejemplo`, o los objetos `CityInfo.o` y `infotest.o` para obtener el ejecutable `infotest`:

Terminal

```
g++ -o ejemplo ejemplo.o CityInfo.o
g++ -o infotest infotest.o CityInfo.o
```

También puedes crear un `Makefile` para obtener los ejecutables [3].

[3] Ver detalles de cómo hacerlo en el seminario de C++

6. Probar la práctica

- En Moodle se publicará un corrector de la práctica con algunas pruebas (se recomienda realizar pruebas más exhaustivas).
- El corrector viene en un archivo comprimido llamado `correctorP0.tgz`. Para descomprimirlo debes copiar este archivo donde quieras realizar las pruebas de tu práctica y ejecutar:

  Terminal
  ```
  tar xzvf correctorP0.tgz
  ```

  De esta manera se extraerá una carpeta con el siguiente contenido:

  - Fichero `corrige.sh`: *shell-script* que tienes que ejecutar.
  - Carpeta `pruebas`: dentro de esta carpeta están los ficheros con las pruebas que se van a ejecutar.
  - El fichero `README.md`: fichero con información adicional sobre el funcionamiento de las prueba e instrucciones para que puedas crear y ejecutar tus propias pruebas.
- Una vez descomprimido el fichero, debes copiar tus ficheros fuente a la misma carpeta donde está el fichero `corrige.sh`.
- Sólo queda ejecutar el *shell-script*. Primero debes darle permisos de ejecución. Para ello ejecuta:

  Terminal
  ```
  chmod +x corrige.sh
  ./corrige.sh
  ```

Apéndice. Trabajar con strings en C++

La clase `string` permite usar cadenas de tamaño variable, no estando limitado el número de caracteres que contiene. Para poder usarla, se debe incluir la librería `<string>` o `<iostream>`.

```
#include <string> // Opcional si ya se ha incluido <iostream>
using namespace std;
```

Las operaciones más frecuentes son:

- Longitud del `string` con el método `length()`:

```
string s = "hello world";
unsigned int n = s.length();
```

- Búsqueda de subcadena con el método `find()`:

```
// Si no se encuentra devuelve la constante string::npos
string s = "hello world";
if (s.find("world") != string::npos) cout << "Encontrado" << endl;
```

- Operadores que se pueden usar con strings:
  - Comparaciones ($==, !=, >, <, >=, <=$)
  - Asignación ($=$)
  - Concatenación ($+$)
  - Acceso a componentes como si fuera un array de caracteres ($[\,]$), pero sólo si el string contiene información. Si se accede a un índice que no existe en el string, se produce un error.

```
string s1="hello", s2="world";
string s3 = s1+s2; // s3="helloworld"
if (s3 == "helloworld") cout << "Son iguales" << endl;
s3[0] = 'H'; // s3="Helloworld"
s3[100] = 'a'; // Error: fuera de rango
```

- Conversión de `int` a `string` con la función global `to_string()`, definida en `<string>`:

```
int n=100;
string numero=to_string(n);
```

- Conversión de `string` a array de caracteres con el método `c_str()`.
- Conversión de `string` a `int` con la función `atoi()` combinada con `c_str()`:

```
string numero="100";
int n=atoi(numero.c_str()); // atoi convierte un array de caracteres a entero
```

- Tokenizar un string usando la clase `stringstream`:

```
#include <iostream>
#include <sstream> // Necesario para usar stringstream
using namespace std;

int main() {
  stringstream iss("ser o no ser");
  string token;

  while (iss >> token) {
    cout << "Token: " << token << endl;
  }
}
```