Curso 2019-2020
Estructura de Datos y Algoritmia
Práctica 0 (22 agosto)

Departamento de Lenguajes y Sistemas Informáticos

Fechas importantes

Plazo de entrega: desde el lunes 23 de septiembre hasta el viernes 27 de septiembre a las 23:59 horas en el servidor de prácticas del departamento de Lenguajes y Sistemas Informáticos (`http://pracdlsi.dlsi.ua.es`)

Objetivo

Esta primera práctica nos permitirá empezar a familiarizarnos con C++ y la declaración e implementación de clases, que entre sus elementos debe contener la **forma canónica** de la clase. La forma canónica es la forma ortodoxa de declarar una clase en C++. 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 (=)

1. Declaración y definición de la clase InfoTur

Hay que construir una clase que representa una estructura que almacena información relevante sobre localidades que estarán situadas en un mapa [1]. En concreto contendrá:

Parte privada:

- `int museo;`
- `int monumento;`
- `int hotel;`
- `int restaurante;`

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

- `bool aeropuerto;`
- `string top;`

Forma canónica (parte pública):

- `InfoTur( )`; //constructor por defecto, inicializa a cadena vacía el string, a false el boolean, y a 0 los enteros
- `InfoTur(int mu, int mo, int ho, int r, bool ae)`; //constructor que inicializa la estructura interna de la clase y deja vacía la variable `top`. Si algún valor es negativo se inicializará a 0.
- `InfoTur(const InfoTur &)`; //constructor de copia
- `~InfoTur()`; //destructor que inicializa el objeto a los mismos valores que el constructor por defecto
- `InfoTur & operator=(const InfoTur &)`; //sobrecarga del operador asignación. Esta operación devuelve una referencia a un objeto de tipo `InfoTur` que es una copia exacta del objeto pasado por parámetro

Métodos (parte pública):

- `bool operator!=(const InfoTur &);` //sobrecarga del operador `!=`. Consideramos que un `InfoTur A` es distinto a otro `B` cuando al menos uno de los elementos definidos es distinto
- `bool operator==(const InfoTur &);` //sobrecarga del operador ==. Consideramos que un `InfoTur A` es igual a otro `B` cuando todos sus elementos son iguales
- `vector<int> getInfoTur();` //devuelve la información turística asociada al objeto (exceptuando la de la variable `top`) en un vector en el mismo orden que se han definido las variables en el enunciado anteriormente. La existencia o no de aeropuerto se devuelve con 0 si no tiene, y 1 si sí lo tiene
- `string getMasFrecuente();` //devuelve el elemento turístico del que haya más cantidad. Si hay más de uno, se imprimirá el primero en orden lexicográfico [2]
- `void setTop(string n);` //actualiza la variable `top` con el string pasado por parámetro
- `string getTop();` //devuelve el valor de la variable `top`

[2] el string devuelto debe ser: “museo”, “monumento”, “hotel”, “restaurante” o “aeropuerto”

Funciones amigas:

- `friend ostream & operator<<(ostream &, const InfoTur &);` //muestra la información turística almacenada en una línea y separados todos los elementos con un espacio en blanco: número de museos, número de monumentos, número de hoteles, número de restaurantes, aeropuerto, top y nueva línea.

  En caso de que alguna variable no contenga información se mostrará la información por defecto.

2. Aplicación

Implementa un fichero denominado `Probando.cc` que contendrá un `main` y que nos permitirá ejecutar una pequeña aplicación de la clase `InfoTur` y en el que tendremos que:

1. crear un array dinámico de objetos de tipo `InfoTur`;
2. leer de la entrada estándar enteros separados por un espacio en blanco que representan números. Para cada entero, se creará un objeto de tipo `InfoTur`, de forma que todas sus variables de instancia de tipo entero tomarán ese valor. Para el último valor del constructor, se pasará un booleano con valor `false` si dicho número es par y `true` en caso contrario. Cada objeto será añadido en el vector por el final;
3. crear otro array dinámico de `InfoTur` en el que aparezcan sólo los objetos del primer array que tengan aeropuerto y además ocupen una posición par en dicho array [3];
4. mostrar en la salida estándar, uno por línea, cada uno de los `InfoTur` del segundo array, según el formato de la operación $<<$ definida en la clase `InfoTur`.

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

3. Ficheros InfoTur.h, InfoTur.cc y Ejemplo.cc

InfoTur.h

La declaración de la clase se realiza en el fichero de cabecera `InfoTur.h`. Formato del fichero:

[3] el 0 se considera par

```
#ifndef INFOTUR_h //directiva de preprocesamiento para evitar incluir dos veces el
    fichero
#define INFOTUR_h

#include <iostream>
#include <string>
#include <vector>

using namespace std;

class InfoTur{
  private:
  //estructuras que definen la clase

  public:
  //declaracion de operaciones canonicas, operadores y metodos

  //operaciones amigas (friend)
  friend ostream & operator<<(ostream &, const InfoTur &);
};

#endif
```

InfoTur.cc

La implementación de las operaciones canónicas, operadores y métodos se realiza en el fichero `InfoTur.cc`

```
#include ``InfoTur.h''

InfoTur::InfoTur(){ ... }   //constructores
InfoTur::InfoTur( int m, int n, int h, int r, bool a ){ ... }
InfoTur::InfoTur( const InfoTur &t ){ ... }

InfoTur::~InfoTur(){ ... } //destructor

InfoTur & InfoTur::operator=(const InfoTur &t){ ... } //operadores
bool InfoTur::operator!=(const InfoTur &t){ ... }
bool InfoTur::operator==(const InfoTur &t){ ... }

vector<int> InfoTur::getInfoTur() { ... } //metodos
string InfoTur::getMasFrecuente() { ... }
string InfoTur::getTop() { ... }
void InfoTur::setTop( string s ){ ... }

ostream & operator<<(ostream &os, const InfoTur &c){ //funcion amiga sobrecarga del
    operador salida
  os << c.museo << `` '' << c.monumento << `` '' << c.hotel << `` '' << c.
     restaurante `` '' << c.aeropuerto << `` '' << c.top << endl;
  return os;
}
```

Ejemplo.cc

Probaremos la clase desde otro fichero. Este fichero no hay que implementarlo para la práctica, pero es conveniente que lo analicéis y ejecutéis para probar las operaciones y métodos. Es simplemente un ejemplo de uso de algunas operaciones y operadores:

```
#include <iostream>
#include ``InfoTur.h''
using namespace std;

int main( int argc, char *argv[] )
{
   //Ejemplo con objetos estaticos
   InfoTur A, B(1,0,0,1, true), C(0, 0, 0, 0, false);

   if( A!=B ) cout << ``los InfoTur A y B son diferentes'' << endl; //true
   if( A!=C ) cout << ``los InfoTur A y C son diferentes'' << endl;
   else cout << ``los InfoTur A y C son iguales'' << endl; //son iguales

   InfoTur D(B); //crea el InfoTur D como una copia de B

   cout << D.getMasFrecuente() << endl; //muestra el elemento más frecuente en D
   cout << D << endl; //muestra el InfoTur D con el formato definido

   return 0;
}
```

4. Compilación y enlace

Compilaremos cada uno de los ficheros .cc como

```
g++ -c InfoTur.cc
g++ -c Ejemplo.cc
g++ -c Probando.cc
```

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

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

Por último, enlazaremos los ficheros objeto creados `InfoTur.o` y `Ejemplo.o` para obtener el ejecutable `Ejemplo`, o los objetos `InfoTur.o` y `Probando.o` para obtener el ejecutable `Probando`:

```
g++ -o Ejemplo InfoTur.o Ejemplo.o
```

También se puede crear un `Makefile` para obtener los ejecutables [4].

5. Apéndice. Clase string de C++

Es de tamaño variable, no estando limitado el número de caracteres que contiene. Los métodos más utilizados son:

- Longitud del `string`:

  ```
  unsigned int length();
  ```

[4] ver detalles de cómo hacerlo en el seminario de C++

- Búsqueda de subcadena:

```
// Si no se encuentra devuelve la constante string::npos
unsigned int find (const string str, unsigned int pos=0);
```

- Operaciones de `string`:
  - comparaciones ($==, !=, >, <, >=, <=$)
  - asignación ($=$)
  - concatenación ($+$)
  - acceso a componentes como si fuera un array de caracteres (sólo si el string contiene información) ([ ])
- conversión de array de caracteres a `string` (con la asignación =)
- conversión de `string` a array de caracteres (función `c_str()`)
- conversión de `int` a `string`

```
#include <sstream>
int n=100;
stringstream ss;
ss << n;
// Tambien se pueden concatenar mas cosas, por ejemplo:
// ss << "El numero es: " << n << endl;
string numero=ss.str();
```

- conversión de `string` a `int`

```
string numero="100";
int n=atoi(numero.c_str());
```

- tokenizar un string (usando la clase `stringstream`)

```
#include <iostream>
#include <sstream>

int main()
{
  stringstream iss(``ser o no ser'');
  string sal;

  while(iss>>sal)
  {
    cout << "Substring: " << sal << endl;
  }

  return(0);
}
```

6. Normas generales

Entrega de la práctica:

- Lugar de entrega: servidor de prácticas del DLSI, dirección `http://pracdlsi.dlsi.ua.es`
- Plazo de entrega: desde el lunes 23 de septiembre hasta el **viernes 27 de septiembre a las 23:59 horas**.
- Se debe entregar la práctica en un fichero comprimido con todos los ficheros creados y ningún directorio de la siguiente manera

```
tar cvfz practica0.tgz InfoTur.h InfoTur.cc Probando.cc
```

- No se admitirán entregas de prácticas por otros medios que no sean a través del servidor de prácticas.
- El usuario y contraseña para entregar prácticas es el mismo que se utiliza en UACloud.
- La práctica se puede entregar varias veces, pero sólo se corregirá la última entregada.
- Los programas deben poder ser compilados sin errores ni warnings con el compilador de C++ existente en la distribución de Linux de los laboratorios de prácticas; si la práctica no se puede compilar su calificación será 0. Se recomienda compilar y probar la práctica con el autocorrector inmediatamente antes de entregarla.
- Los ficheros fuente deber estar adecuadamente documentados usando comentarios en el propio código, sin utilizar en ningún caso acentos ni caracteres especiales.
- Es imprescindible que se respeten estrictamente los formatos de salida indicados, ya que la corrección se realizará de forma automática.
- Al principio de cada fichero entregado (primera línea) debe aparecer el DNI y nombre del autor de la práctica (dentro de un comentario) tal como aparece en UACloud (pero sin acentos ni caracteres especiales).

  Ejemplo:
  `DNI 23433224 IÑESTA MARTÍNEZ, JÚLIA` $\Rightarrow$ NO
  `DNI 23433224 INESTA MARTINEZ, JULIA` $\Rightarrow$ SI

Sobre la evaluación en general:

- 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.

- La influencia de la nota de esta práctica sobre la nota final de la asignatura está publicada en la ficha oficial de la asignatura (apartado evaluación).

7. Probar la práctica

- En UACloud 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 se debe copiar este archivo donde queramos realizar las pruebas de nuestra práctica y ejecutar: `tar xfvz correctorP0.tgz`.

  De esta manera se extraerán de este archivo:

  - El fichero `corrige.sh`: *shell-script* que tenéis que ejecutar.
  - El directorio `practica0-prueba`: dentro de este directorio están los ficheros
    - `p01.cc`: programa en C++ con un método main que realiza una serie de pruebas sobre la práctica.
    - `p01.txt`: fichero de texto con la salida correcta a las pruebas realizadas en `p01.cc`.
- Una vez que tenemos esto, debemos copiar nuestros ficheros fuente (`InfoTur.h`, `InfoTur.cc y Probando.cc`) al mismo directorio donde está el fichero `corrige.sh`.
- Sólo nos queda ejecutar el *shell-script*. Primero debemos darle permisos de ejecución. Para ello ejecutar:

```
chmod +x corrige.sh
corrige.sh
```