ST0244-031-032
Taller 2

DIS
Revisión 2

Entrega: 16 de octubre 2015, Hora: 18:00

1. Preliminares

Este taller es calificable y hace parte del 20 % de seguimiento que tiene la materia. La entrega del taller se debe hacer utilizando *subversion*.

1.1. Estructura del directorio de entregas

Cada estudiante ha creado un repositorios en *riouxsvn* que tiene como nombre: 244*s*⟨username⟩.

1.1.1. Registro del área de trabajo

Suponga que no ha hecho un registro del área de trabajo (*checkout*) de su área de trabajo en su equipo (Si ya lo hizo pase al punto siguiente). Para hacer el registro abra una terminal en `cygwin`[1] y digite los siguientes comandos:

```
$ svn co <url-repo-segu>
$ cd 244s<username>
```

Ahora crea una carpeta para guardar los talleres, si ya lo hizo puede omitir este punto:

```
$ svn mkdir talleres
$ cd talleres
```

[1] El taller está basado en cygwin, pero puede funcionar igualmente con otras terminales en mac, linux y unix.

1.1.2. Creación del directorio de taller 2

Ahora en el directorio de talleres, cree un nuevo directorio para este segundo taller:

```
$ svn mkdir taller02
$ cd taller02
```

En esta carpeta (`taller02`) va a guardar los problemas que se soliciten entregar (subir[2]) del taller. Los ejercicios no se entregan (suben[3]).

1.2. Nombres de los ejercicios y programas de los talleres.

Cada problema que debe ser entregado tiene un nombre entre paréntesis que indica el nombre del fichero (con extensión) que va a ser registrado (`add`) y enviado (`commit`) al control de versiones. Por ejemplo: el primer problema de este taller se tiene que crear un fichero vacío llamado `emptyfile.cpp`. Piense un momento como se puede resolver el problema, cree el fichero utilizando su editor de texto favorito[4] y registre su nuevo fichero en el control de versiones y una vez halla obtenido la respuesta correcta regístrelo en el control de versiones, una posible secuencia sería esta:

```
$ emacs -nw emptyfile.cpp
-- final de la ejecucion de emacs: C-x C-s C-x C-c
$ svn add emptyfile.cpp
$ # Compilar y funciono
$ svn ci -m "Primer punto del taller solucionado"
```

2. Ficheros fuentes

Los programas en C++ son almacenados en dos tipos tipos de ficheros: encabezados (`.h` - *header files*) y fuentes (`.cpp`[5] - *source files*). Cualquiera puede almacenar una secuencia de declaraciones o definiciones de alto orden:

[2] *checking*
[3] *checking*
[4] Después de todas estas semana, no creo que tengan otro editor favorito diferente a `emacs`.
[5] Son varias las extensiones utilizadas para los fichero fuentes: `.cc`, `.cpp`, `.cxx`, `.C`

$$\langle Fichero_{c++}\rangle \;:=\; [\,\langle DefDeclAltoNivel\rangle\,]\ldots$$
$$\langle DefDeclAltoNivel\rangle \;:=\; \langle DefAltoNivel\rangle \mid \langle DeclAltoNivel\rangle$$

Al observar la sintaxis anterior se puede generar un fichero fuente vacío:

Problema 1. (`emptyfile.cpp`) Cree un fichero vacío llamado `emptyfile.cpp` y pruebe que el compilador acepta un fichero vacío (Compilar con la opción `-c`).

Ejercicio 1. Es posible generar un ejecutable con el fichero del problema 2.

3. Declaración y definición

Según Stroustrup (ver [3, pág. 78]) una declaración relaciona un nombre con el elemento que lo identifica y antes que un nombre sea utilizado este debe ser declarado. "Esto es, que el tipo debe ser especificado para informa al compilador un nombre que clase de entidad se está refiriendo"([3, pág. 78])[6].

La siguiente es la sintaxis de una declaración[7]:

$$\begin{aligned}
\langle DecAltoNivel\rangle &:= \langle DecVarsAltoNivel\rangle \\
&\mid \langle DecFuncAltoNivel\rangle \\
\langle DecVarsAltoNivel\rangle &:= [\langle Mod\rangle]\ \langle Tipo\rangle\ [\langle Deter\rangle](\mathrm{Id}[=\langle Literal\rangle])\ldots \\
\langle Mod\rangle &:= (\mathrm{unsigned} \mid \mathrm{signed} \mid \mathrm{long} \mid \mathrm{short})\ldots \\
\langle Tipo\rangle &:= \mathrm{int} \mid \mathrm{bool} \mid \mathrm{char} \mid \mathrm{float} \mid \mathrm{double} \\
\langle Deter\rangle &:= *\ldots \mid \& \\
\langle DecFuncAltoNivel\rangle &:= [\langle Mod\rangle]\ \langle Tipo\rangle\ [\langle Deter\rangle](\mathrm{Id}([\langle ListParmDeclFun\rangle]))\ldots \\
\langle ListParmDeclFun\rangle &:= \langle Tipo\rangle[\mathrm{Id}][\langle Tipo\rangle[\mathrm{Id}]]\ldots
\end{aligned}$$

En este segmento mostramos la declaración de variables de memoria. Esto permite ligar un nombre (un identificador) con el tipo correspondiente y si lo quiere inicializar con su correspondiente valor constante.

[6] El original en inglés: traducción libre hecha por el autor

[7] Esta sintaxis está modificada un poco con respecto a la sintaxis de C++ [3] por propósitos explicativos.

Problema 2. (`varlits.cpp`) Escriba en el fichero las siguientes variables con sus correspondientes tipos y literales.

Variable	Tipo	Valor inicial
a	Entero con signo	por omisión
b	Boolean	por omisión
c	flotante	10,23
d	double	$23e+23$
e	Entero largo sin signo	23430303
f	Caracter	con el carácter de salto de línea[8]
g	Entero con signo	*f*

El programa debe ser compilado y generar código objeto (extensión `.o`).

Problema 3. (**declfun.cpp**) Escriba un fichero fuente que declare las siguiente funciones:

Nombre	Argumentos entrada	Valor salida
a	2 argumentos enteros	No retorna nada
b	3 argumentos enteros cortos sin signo	Retorna un carácter
c	Sin argumentos	No retorna nada
d	10 argumentos enteros largos con signo	Retorna un bool
e	Sin argumentos	Retorna un entero

El programa debe ser compilado y generar código objeto (extensión `.o`).

4. Definición de funciones

La definición introduce una entidad[jfcm] como un tipo de dato nuevo (*tipos definidos de usuario)* o una función. La siguiente gramática muestra como se define una función:

$$
\begin{aligned}
\langle DefFunción\rangle &:= [\langle Mod\rangle][\langle Tipo\rangle][\langle Deter\rangle]\text{IdFunción}(\langle LstParaDefFun\rangle)\langle BloqueInstr\rangle \\
\langle Mod\rangle &:= (\text{unsigned} \mid \text{signed} \mid \text{long} \mid \text{short})\ldots \\
\langle Tipo\rangle &:= \text{int} \mid \text{float} \mid \text{double} \mid \text{bool} \mid \text{char} \mid \text{void} \\
\langle Deter\rangle &:= *\ldots \mid \&
\end{aligned}
$$

El lenguaje de programación C++ permite definir funciones de usuario. Según [4, pág. 307] “Una definición de una función es una función declaración en la cual el cuerpo de la función está presente”.

Problema 4. (ficheros: `funciones.h`, `funciones.cpp`) En el fichero `funciones.h`, están declaradas varias funciones que están documentadas en el mismo fichero. En el fichero `funciones.cpp` defina las funciones correspondientes.

Problema 5. (ficheros: `funciones.h`, `funciones.cpp` `main.cpp`). Escriba un programa `programa1.exe` que utilice las funciones del punto anterior y implemente la siguiente expresión:

$$a + b + (c * ((a * b) + c) * (a - b))$$

Donde las variables a, b y c se leen de la entrada estándar y el programa hace la expresión con las funciones definidas en `funciones.h`.

Al ejecutar `programa1.exe` con la entrada (la línea seguiente a la ejecución) en la siguiente línea se obtiene el resultado.

```
\$ ./programa1
10 20 30
-68970
```

Problema 6. (ficheros: `funciones2.h`, `funciones2.cpp`) En el fichero `funciones2.h` se encuentra la definición de dos funciones: `mcd` (máximo común divisor) y la función de `simpFrac` (simplificación de fracciones). La definición formal se encuentra en [1]. Implemente ambas funciones en el fichero `funciones2.cpp`.[jfcm]

5. Instrucciones y expresiones

5.1. Expresiones

Según Scott[2] "Una expresión consiste de tanto de un objeto (por ejemplo un literal constante o una variable o una constante) o de un operador o una función aplicada a una colección de operandos o argumentos, cada uno de los cuales en turno es una expresión".

El lenguaje de programación C++ es lenguaje que tiene expresiones y estas están asociadas al tipo de dato correspondiente. Los tipos de datos fundamentales casi todos ellos son valores numéricos: aceptan operaciones de suma, resta, multiplicación y división. Las operaciones numéricas de enteros tiene operaciones a nivel de bits: desplazamiento a la derecha, desplazamiento a la izquierda, negación de bits, o-a-nivel-de-bit, y-a-nivel-de-bits, o-exclusivo-a-nivel-de-bits.

También existen operaciones de comparación para los tipos de datos fundamentales: igualdad, diferencia, menor que, mayor que, menor igual que, mayor o igual que.

Las operaciones también tiene una precedencia de tal forma que las expresiones tienen un orden predeterminado, en el siguiente enlace precedencia de operadores.

Problema 7. (fichero: `expr01.cpp`) En el fichero `expr01.cpp` se encuentra la evaluación de dos expresiones con una gran cantidad de paréntesis, elimine el máximo número de paréntesis para obtener el mismo resultado, utilizando la precedencia de operadores que existe en C++.

Un posible ejemplo puede ser:

```
$ ./expr01
Lea tres valores enteros separados por espacios:
10 20 1
Res1: 100
Res2: 5242964
```

Problema 8. (fichero: `expr02.cpp`) Ordenamiento de 3 enteros. Escriba un programa (`expr02.cpp`) [jfcm]que lee tres valores enteros de la terminal y retorne los tres valores ordenados de manera ascendente utilizando las instrucciones: max y min.

Un posible ejemplo puede ser:

```
$ ./expr02
2342 212345 233
233 2342 212345
```

Problema 9. (fichero: `expr03.cpp`) Ordenamiento de 3 enteros. Escriba un programa (`expr03.cpp`) [jfcm]que lee tres valores enteros de la terminal y retorne los tres valores ordenados de manera descendente utilizando las instrucciones: max y min.

Un posible ejemplo puede ser:

```
$ ./expr03
2342 212345 233
212345 2342 233
```

Ejercicio 2. Ordenamiento de 3 valores dobles. Escriba un programa que lee tres valores dobles de la terminal y que retorne los tres valores ordenados de manera ascendente utilizando las instrucciones max y min.

5.2. Instrucciones

La siguiente es la sintaxis vista en clase para las instrucciones del lenguaje de programación C++.

⟨*Instrucción*⟩ := ⟨*Asignación*⟩
| ⟨*Condición*⟩
| ⟨*Ciclo*⟩
| ⟨*Expresión*⟩
| ⟨*BloqueInstr*⟩
| ⟨*ReturnInstr*⟩
| ⟨*Decl*⟩

⟨*Asignación*⟩ := Id = ⟨*Expresión*⟩ ;

⟨*Condición*⟩ := if (⟨*Expresión*⟩) ⟨*Instrucción*⟩ [else ⟨*Instrucción*⟩]

⟨*Ciclo*⟩ := ⟨*CicloWhile*⟩ | ⟨*CicloFor*⟩ | ⟨*CicloDoWhile*⟩

⟨*CicloWhile*⟩ := while (⟨*Expresión*⟩) ⟨*Instrucción*⟩

⟨*CicloFor*⟩ := for ([⟨*Decl*⟩] ; [⟨*Expresión*⟩] ;[⟨*Expresión*⟩]) ⟨*Instrucción*⟩

⟨*CicloDoWhile*⟩ := do ⟨*Instrucción*⟩ while (⟨*Expresión*⟩) *term*;

⟨*BloqueInstr*⟩ := {[⟨*Instrucción*⟩]...}

⟨*ReturnInstr*⟩ := return [⟨*Expresión*⟩] ;

5.3. Condiciones y ciclos

En C++, las condiciones y los ciclos tienen una expresión condicional, está expresión es numérica, a diferencia de java que las expresiones son de tipo booleano. Esto implica que al evaluar una expresión el valor es verdadero cuando su valor es diferente de cero y cero en caso contrario.

```
1 int i = 0;
2
3 if (i)
4   cout << "Verdadero" << endl;
5 else
6   cout << "Falso" << endl;
```

El anterior código imprime `Falso`, el siguiente código imprime `Verdadero`

```
int i = 1;

if (i)
  cout << "Verdadero" << endl;
else
  cout << "Falso" << endl;
```

Los siguientes son ejemplos de ciclos infinitos:

```
// Primer ejemplo infinito
for (;;)
  cout << "Siempre imprime esto" << endl;

// Segundo ejemplo infinito
while (1)
  cout << "Siempre imprime esto" << endl;

// Tercer ejemplo infinito
do {
  cout << "Siempre imprime esto" << endl;
} while(1);
```

5.3.1. Condiciones

Problema 10. (fichero: `condicion01.cpp`) Reescriba el siguiente código de programa de tal manera que la operación de entrada y salida estén en una sola instrucción.

```
#include <iostream>

using namespace std;

int
main() {
  int c;

  cin >> c;

  if (c == 0) {
    cout << "El usuario digito un valor de cero" << endl;
```

```
  }
  else {
    cout << "El usario digito un valor diferente de cero" <<
    endl;
  }

  return 0;
}
```

Problema 11. (fichero: `condicion02.cpp`) Escriba un programa en C++ que lea dos valores a y b. En una instrucción posterior~~, en la misma instrucción~~[jfcm] verifique que ambos valores son diferentes de cero.

Ejercicio 3. (fichero: `condicion03.cpp`) Compile y ejecute el siguiente programa en C++:

```
#include <iostream>

using namespace std;

int
main() {
  int a;
  int b;
  int suma;
  int diff;

  cin >> a;
  cin >> b;

  if ((suma = a + b) || (diff = a - b)) {
    cout << "suma: " << suma << " resta: " << diff << endl;
  }
  else {
    cout << "suma: " << suma << " resta: " << diff << endl;
  }

  return 0;
}
```

Ejecute con varios valores y entienda y explique por que se obtiene estos resultados.

Problema 12. (fichero: `condicion04.cpp`) Reescriba el código del ejercicio 32[jfcm] de tal manera que trabaje correctamente.

Problema 13. (fichero: `condicion05.cpp`) Un triángulo puede ser clasificado basado en la longitud de sus lados como equiláteros, isósceles o escaleno. Según wikipedia: "Un triángulo es equilátero, cuando los tres lados del triángulo tiene la misma longitud", "Un triángulo es isósceles, si tiene dos lados de la misma longitud.", "Un triángulo es escaleno, si todos sus lados tiene longitudes diferentes". Escribir un programa que lea tres lados de un triángulo y determine el tipo de triángulo que ha entrado el usuario.

Los siguientes son ejemplos de ejecución del programa:

```
$ ./condicion05
2 2 2
equilatero
$ ./condicion05
1 2 1
isoceles
$ ./condicion05
1 2 3
escaleno
```

Problema 14. (fichero: `condicion06.cpp`) Las posiciones en un tablero de ajedrez son identificadas por una letra y un número. Una letra identifica la columna, mientras que el número identifica la columna como se observa en la figura 1.

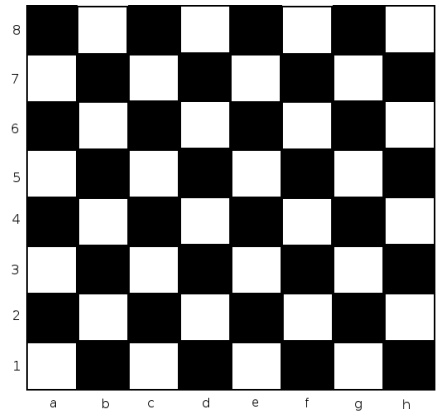

Figura 1: Tablero de ajedrez

Escriba un programa (`condicion06.cpp`) que lee una posición dada por el usuario. Utilice una instrucción `if` para determinar reportar el color de la posición. (Asuma que siempre los valores entrados son válidos).

Ejemplos:

```
$ ./condicion06
a1
blanco
$ ./condicion06
e6
negro
```

5.3.2. Ciclos

Ejercicio 4. (fichero: `ciclo01.cpp`) Sin compilar el siguiente programa: ¿Cuántas veces se ejecuta el ciclo?. Ahora revise su resultado con la ejecución obtenida del programa. ¿Por qué puede existir una diferencia? ¿Cuál es valor final de `i`?

```
#include <iostream>

using namespace std;

const int N = 10;

int main() {
  int i;

  i = 0;
  while ((++i < N)) {
    cout << "veces: " << i << endl;
  }

  cout << i << endl;
  return 0;
}
```

Ejercicio 5. (fichero: `ciclo02.cpp`) Sin compilar el siguiente programa: ¿Cuántas veces se ejecuta el ciclo?. Ahora revise su resultado con la ejecución obtenida del programa. ¿Por qué puede existir una diferencia? ¿Cuál es valor final de `i`?

```
#include <iostream>

using namespace std;

const int N = 10;

int main() {
  int i;

  i = 0;
  while ((i++ < N)) {
    cout << "veces: " << i << endl;
  }

  cout << i << endl;
```

```
  return 0;
}
```

Ejercicio 6. (fichero: `ciclo03.cpp`) Sin compilar el siguiente programa: ¿Cuántas veces se ejecuta el ciclo?. Ahora revise su resultado con la ejecución obtenida del programa. ¿Por qué puede existir una diferencia? ¿Cuál es valor final de `i`?

```
#include <iostream>

using namespace std;

const int N = 10;

int main() {
  int i;

  i = 0;
  for (i = 0; i < N; i++) {
    cout << "veces: " << i << endl;
  }

  cout << i << endl;
  return 0;
}
```

Ejercicio 7. (fichero: `ciclo04.cpp`) Sin compilar el siguiente programa: ¿Cuántas veces se ejecuta el ciclo?. Ahora revise su resultado con la ejecución obtenida del programa. ¿Por qué puede existir una diferencia? ¿Cuál es valor final de `i`?

```
#include <iostream>

using namespace std;

const int N = 10;

int main() {
  int i;

  i = 0;
```

```
11    for (i = 0; i < N; ++i) {
        cout << "veces: " << i << endl;
13    }
      cout << i << endl;
15    return 0;
}
```

Ejercicio 8. (fichero: `ciclo05.cpp`) Sin ejecutar el siguiente código, indique que valores produce la correspondiente salida. Ahora compile el siguiente código y mire si su solución esta en lo correcto.

```
#include <iostream>
2
using namespace std;
4
void nada() { }
6
int main() {
8   int x, y = 10, z = 20;

10  for (x = 0; x < z + y; x += 5) {
      nada();
12  }

14  cout << "x: "  << x
         << " y: " << y
16       << " z: " << z << endl;

18  return 0;
}
```

Problema 15. (fichero:`ciclo06.cpp`) Escribir un programa que lea una secuencia de pares de números, y calcule la suma de estos entre el primer y el último.

Con la siguiente entrada[9]:

```
1 2
1 3
1 5
1 1000
234 23434
234 1222
23 2333
```

Se obtiene la siguiente salida:

```
3
6
15
500500
274560634
719992
2722358
```

6. Utilidad *make*

Problema 16. (fichero: `MakeFunciones`) Crear un fichero llamado `MakeFunciones` que permite compilar los programas de los problemas 4 y 5, para generar un ejecutable llamado `main`.

7. Arreglos

Uno de los primeros tipos definidos de usuario es la posibilidad de tener un conjunto de elementos del mismo tipo indexado por un valor entero, esto es un arreglo. "Para el tipo `T`, `T[size]` es un arreglo de tamaño `size` de tipo `T`"([4][Pág. 174]).

[9]Que es uno de los casos con el que se prueba el programa, por lo tanto hay mas casos de prueba.

Los arreglos siempre tiene un tamaño que esta establecido por un valor constante y no puede ser modificado hasta que el programa es recompilado nuevamente.

Problema 17. (fichero: `arreglo01.cpp`) El fichero `arreglo01.cpp` muestra un ejemplo se puede leer dos arreglos: `a1` y `a2` de tipo entero. Complete el programa de tal forma que el arreglo `a2` es concatenado al final del arreglo `a1`, sin que ser pierdan los elementos del primero.

Ejercicio 9. En el problema 7 se encuentra un código de inicialización de los arreglos algo extraño.

```
1 const int N = 5 ;
  int a1[N * 2];
3 int a2[N];
```

En la primera línea se observa que se crea una constante entera llamada `N` con un valor inicial de 5 y en la siguiente linea se crea un arreglo llamado `a1` con un tamaño dos veces `N`. En la semántica de C++ el tamaño del arreglo debe ser un valor constante, pero en la segunda línea existe un cálculo que al parecer no es constante. ¿Si lo anterior es así por que este programa compila?

Problema 18. (fichero: `arreglo02.cpp` `arreglo02.h`) Reescriba el código del problema 7 de tal forma se utilice la función definida en `arreglo02.h` y no se tenga que repetir código.

8. Clase, interfaces y clases abstractas en C++

Problema 19. (fichero de encabezado: `Difusores.h` y fichero: `Difusores.cpp`) El lenguaje de modelamiento unificado (UML) permite describir un modelo de clases con herencia. En la figura 2 se muestra la herencia definida en varias clases. Implemente cada una de las clases que ve en diagrama como el diagrama indica que se debe hacer.

En el fichero de encabezado queda la declaración de las clases y en fichero fuente de C++ queda la implementación.

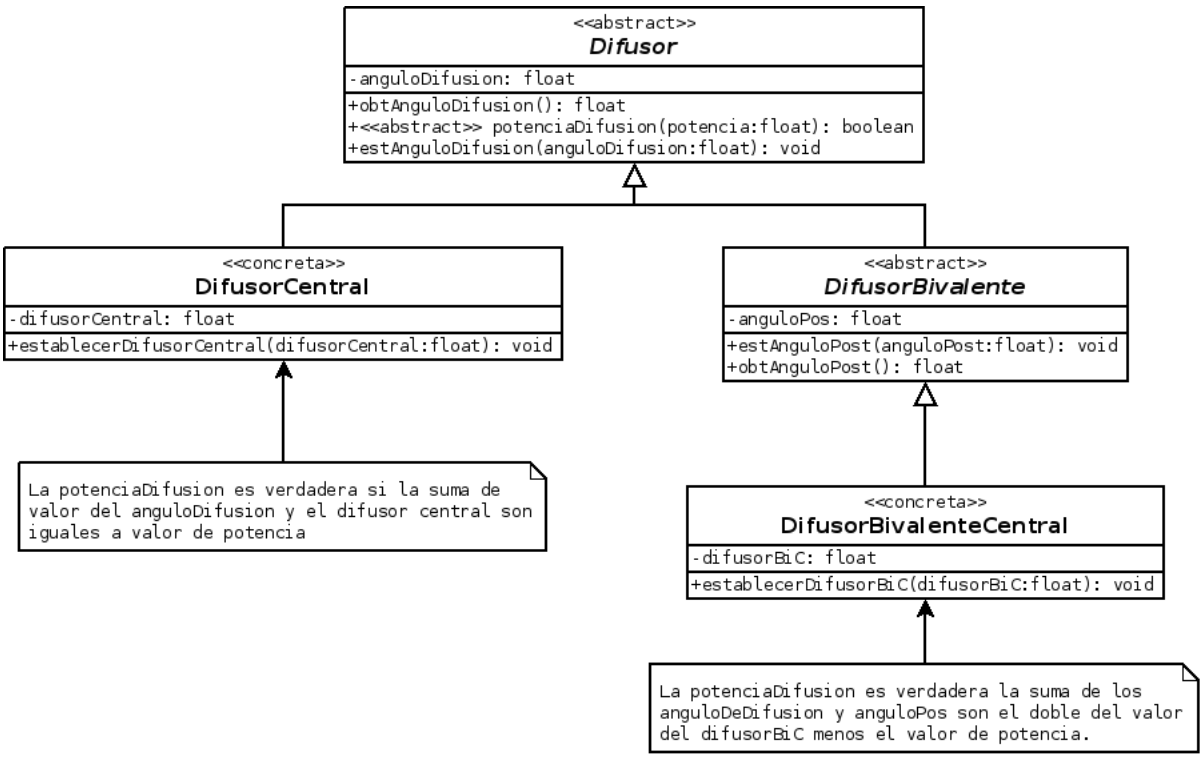

Figura 2: UML y herencia

Problema 20. (ficheros de encabezado: `Interfaces.h`, `Lampara.h` y fichero: `Lampara.cpp`) En la figura 3 se muestra tres interfaces `IEnergia`, `IPrender`, `IApagar` y una clase `Lampara`. La clase lampara debe implementar las tres interfaces es decir implementar cada uno de los métodos que están definidos en cada una de las tres interfaces.

En el fichero de encabezado `Interfaces.h` queda la declaración de las interfaces, en el fichero de encabezado: `Lampara.h` la declaración de la clase `Lampara` y en fichero fuente de C++ queda la implementación de la clase `Lampara`

Cada implementación debe cambiar el estado interno de la lampara de manera coherente.

Problema 21. (ficheros de entrada: `INaves.h`, `Barcos.h`, `Marina.h` y ficheros fuentes: `Barcos.cpp` y `Marina.cpp`) En el diagrama de clases se muestra una posible configuración de una `Armada` con una `ArmadaMarina` y una `ArmadaFluvial`. Se puede observar que este momento todas las armadas son abstractas. La idea es

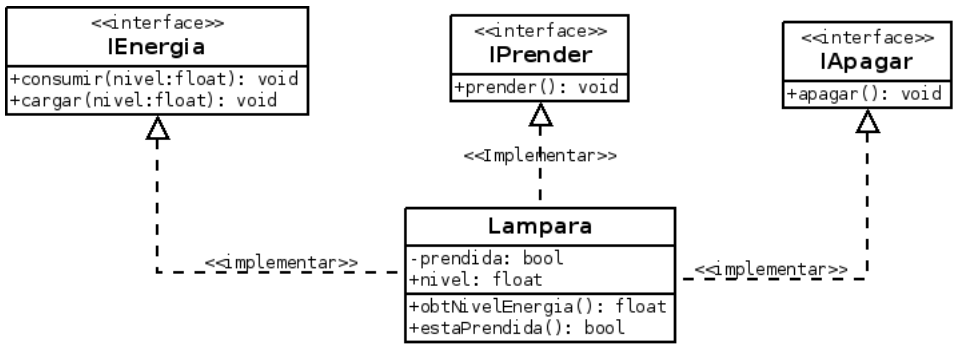

Figura 3: Interfaces e implementación

hacer que las dos clases: `ArmadaMarina` y `ArmadaFluvial` no sean abstractas, para ello deben implementar el método `registrarBarco`, pero la `ArmadaMarina` solamente acepta barcos del tipo de `IMar` y la `ArmadaFluvial` solamente acepta barcos del tipo de `IRio`, los barcos con ambas interfaces puede ser aceptados en ambas marinas. Observe que aunque existen tres subclases de la clase barco, para probar el problema pueden existir más subclases de barco que implementan la interfaz `IMar` ó `IRio`.

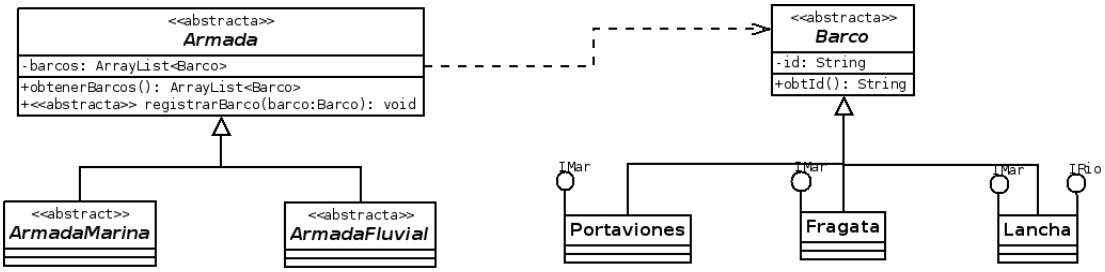

Figura 4: Armada

En el diagrama de UML clases: `ArmadaMarina` y `ArmadaFluvial` tiene una restricción de abstracción, en la implementación ambas clases son concretas e implemente en cada una de ellas el método `registrarBarco` que tiene la siguiente firma (*signatura*):

```
1 void registrarBarco(Barco barcoRegistrar);
```

Debido a que C++ no tiene un constructor `instaceof` de Java que permite reconocer si una clase es una instancia o subclase de una clase específica se debe utilizar un mecanismo similar en C++ llamado RTTI (*RunTime Type Interface*. Este ejemplo tomado de StackOverflow nos muestra de como puede ser utilizado para resolver nuestro problema:

```
if(NewType* v = dynamic_cast<NewType*>(old)) {
    // old was safely casted to NewType
    v->doSomething();
}
```

Problema 22. (fichero de encabezado: `Strategy.h`, fichero fuente: `Strategy.cpp`) En el siguiente diagrama de clases 5 se muestra el patrón de comportamiento (*Strategy*) cuyo propósito es "definir un conjunto de clases que representa un conjunto de comportamientos posibles". Estos comportamientos pueden ser fácilmente intercambiados, modificando la funcionalidad en cualquier instante.

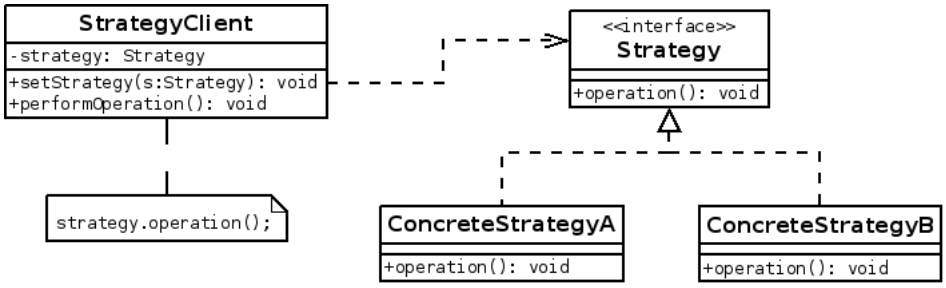

Figura 5: Patrón Estrategia (*Estrategy*)

En este problema vamos a implementar completamente las clases del patrón estrategia, el comportamiento de las cada una de las clases concretas es imprimir un mensaje que distinga a cada una como por ejemplo: "Hello Strategy A" o "Hello Strategy B".

9. C-Strings

En C++ los arreglos de char son conocidos como cadenas de caracteres o *c-strings*.

Problema 23. (fichero encabezado: funcionesCadena.h[jfcm], fichero fuente: funcionesCadena.cpp) Escriba una función saltarblancosinicio que se encarga de reemplazar los espacios en blanco al inicio de una cadena de carácteres por el carácter '\0'. A continuación su propotipo:

```
char* saltarblancosinicio(char *s);
```

Esta función retorna la dirección del primer carácter no blanco.

Problema 24. (fichero encabezado: funcionesCadena.h[jfcm], fichero fuente: funcionesCadena.cpp) Escribala función indiceAlaDerecha que se encarga de de localizar el último carácter que coincide con c (convertido a carácter) en una cadena de carácteres terminado con nulo ('\0').

El siguiente es el prototipo de la función:

```
1 const char* indiceAlaDerecha(const char *s, int c);
```

Esta función retorna la dirección del último carácter c en la cadena s, si el carácter no existe en la cadena retorna nulo ('\0').

Referencias

[1] Algoritmo de Euclides. Algoritmo de euclides — Wikipedia, the free encyclopedia, 2010. [En linea, accedido 9 de Octubre-2015].

[2] Michael L. Scott. *Programming Languages Pragmatics*. Morgan Kaufmann Publishers, Second edition, 2006.

[3] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley, Second edition, 2000.

[4] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley, Third edition, 2014.