

Unidad 6:

Programación Orientada a Objetos - Parte 2

<http://bit.ly/2HRBWgq>

Profesores:

Ernesto Cuadros- Vargas, PhD.

ecuadros@utec.edu.pe

María Hilda Bermejo, M. Sc.

mbermejo@utec.edu.pe

Telegram:

1. Configurar tu cuenta

2. <http://bit.ly/2TJnwBq>

Logro de la sesión:

Al finalizar la sesión, los alumnos se familiarizan con el paradigma de la programación orientada a Objetos.

- **Clase – Objeto**
- **Métodos de acceso (setter y getters)**
- **Constructores, destructores**
- **Uso de archivos**

Ejemplo 1:

Escriba un programa Orientado a Objetos - POO, que permita hallar el área y el perímetro de un rectángulo.

Se diseñará la clase CRectangulo.

El proyecto tiene código en 3 archivos:

main.cpp

CRectangulo.h

CRentangulo.cpp

Ahora declaramos el constructor:

CRectangulo.h

```
#ifndef
RECTANGULO_00_CRECTANGULO_H
#define
RECTANGULO_00_CRECTANGULO_H

class CRectangulo
{
private:
    float largo;
    float ancho;
public:
    CRectangulo(float _largo, float _ancho);
    float Area();
    float Perimetro();
    //----metodos de acceso
    //--- setter que permiten asignar un valor al
atributo
    void setlargo(float _largo) { largo = _largo; }
    void setancho(float _ancho) { ancho =
_ancho; }
    //--- getters
    float getlargo() { return largo; }
    float getancho() { return ancho; }
};

#endif
//RECTANGULO_00_CRECTANGULO_H
```

CRectangulo.cpp

```
#include "CRectangulo.h"

CRectangulo::CRectangulo(float _largo, float _ancho)
{
    largo = _largo;
    ancho = _ancho;
}

float CRectangulo::Area()
{
    return (largo*ancho);
}

float CRectangulo::Perimetro()
{
    return( 2*largo + 2*ancho);
}
```

main.cpp

```
#include <iostream>
#include "CRectangulo.h"
using namespace std;
int main()
{ CRectangulo R(43.5,22.5);

    cout <<"El area del segundo rectangulo es " << R.Area() <<
"\n";
    cout <<"El perimetro del segundo rectangulo es " <<
R.Perimetro();
    return 0;
}
```

Se utiliza typedef para facilitar el mantenimiento del programa.

CRectangulo.h

```
#ifndef RECTANGULO_00_CRECTANGULO_H
#define RECTANGULO_00_CRECTANGULO_H

typedef float TipoFloat;

class CRectangulo
{
private:
    TipoFloat largo;
    TipoFloat ancho;
public:
    TipoFloat Area();
    TipoFloat Perimetro();
    //----metodos de acceso
    //--- setter que permiten asignar un valor al atributo
    void setLargo(TipoFloat _largo) { largo = _largo; }
    void setAncho(TipoFloat _ancho) { ancho = _ancho; }
    //--- getters
    TipoFloat getLargo() { return largo; }
    TipoFloat getAncho() { return ancho; }
};

#endif //RECTANGULO_00_CRECTANGULO_H
```

CRectangulo.cpp

```
#include "CRectangulo.h"

TipoFloat CRectangulo::Area()
{
    return (largo*ancho);
}

TipoFloat CRectangulo::Perimetro()
{
    return( 2*largo + 2*ancho);
}
```

main.cpp

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

int main()
{
    CRectangulo R1;

    R1.setLargo(110.49);
    R1.setAncho(55.25);
    cout <<"\nArea y perimetro de R1 \n";
    cout <<"El area    : " << R1.Area() << "\n";
    cout <<"El perimetro : " << R1.Perimetro();
    return 0;
}
```

Ahora, vamos a crear un segundo objeto de la clase CRectangulo, y los datos para los atributos los leemos desde el teclado.


```

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

int main()
{ CRectangulo R1;

  R1.setLargo(110.49);
  R1.setAncho(55.25);
  cout << "\nArea y perimetro de R1 \n";
  cout << "El area : " << R1.Area() << "\n";
  cout << "El perimetro : " << R1.Perimetro();

  CRectangulo R2;
  TipoFloat l,a;
  cout << "\n\nDatos para el segundo rectangulo \n";
  cout << "Largo : ";
  cin >> l;
  cout << "Ancho : ";
  cin >> a;
  R2.setLargo(l);
  R2.setAncho(a);
  cout << "\nArea y perimetro de R2 \n";
  cout << "El area : " << R2.Area() << "\n";
  cout << "El perimetro : " << R2.Perimetro();
  return 0;
}

```

Pantalla de salida del programa:

Area y perimetro de R1

El area : 6104.57

El perimetro : 331.48

Datos para el segundo rectángulo

Largo : 19

Ancho : 8

Area y perimetro de R2

El area : 152

El perimetro : 54

Constructores y destructores

El constructor:

- Un constructor es una función especial que tiene el “**mismo nombre**” que el de la clase, sin un tipo de retorno (no se debe especificar “void”).
- Como su nombre lo indica, el constructor se llama cada vez que se declara una instancia (o se construye).
- Si no se define ningún constructor en la clase, el compilador inserta un constructor predeterminado, que no toma ningún argumento y no hace nada, es decir:

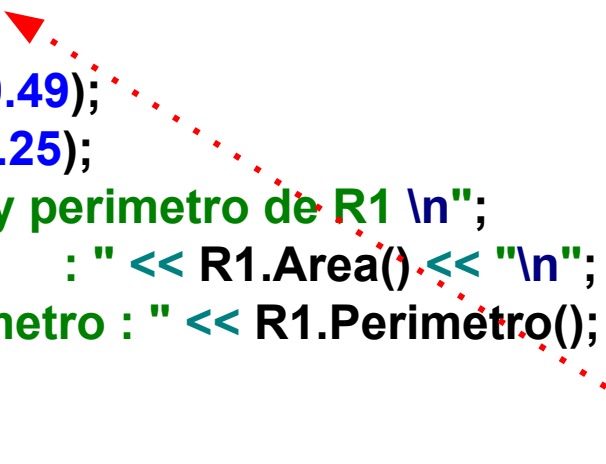
ClassName :: ClassName () {}

- Sin embargo, **si se define uno (o más) constructores, el compilador no insertará el constructor por defecto.** Si es que se requiere, debe definirse explícitamente su constructor por defecto.
- El constructor por defecto suele ser sin parámetros. O en el cuerpo del método inicializar los atributos con valores fijos.

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

int main()
{ CRectangulo R1;

  R1.setLargo(110.49);
  R1.setAncho(55.25);
  cout << "\nArea y perimetro de R1 \n";
  cout << "El area      : " << R1.Area() << "\n";
  cout << "El perimetro : " << R1.Perimetro();
  return 0;
}
```



En el ejemplo anterior, no se ha declarado un constructor, entonces se está utilizando el constructor por defecto.

El constructor:

Lista de inicializadores de atributos:

- Cuando se declara el constructor, se utiliza para inicializar los miembros de datos de las instancias creadas. La sintaxis es:

```
//----- En el archivo *.h o header
// Declaración del constructor dentro de la declaración de la clase
class ClassName {
    ClassName(parameter-list);    // solo prototipo
}
```

```
//----- En el archivo *.cpp
// Implementación del Constructor - identificado via operator alcance
ClassName::ClassName(parameter-list) {
    function-body;
}
```

Ahora declaramos el constructor:

CRectangulo.h

```
#ifndef RECTANGULO_00_CRECTANGULO_H
#define RECTANGULO_00_CRECTANGULO_H

typedef float TipoFloat;

class CRectangulo
{
private:
    TipoFloat largo;
    TipoFloat ancho;
public:
    CRectangulo(TipoFloat _largo, TipoFloat _ancho);
    TipoFloat Area();
    TipoFloat Perimetro();
    //----metodos de acceso
    //--- setter que permiten asignar un valor al atributo
    void setLargo(TipoFloat _largo) { largo = _largo; }
    void setAncho(TipoFloat _ancho) { ancho = _ancho; }
    //--- getters
    TipoFloat getLargo() { return largo; }
    TipoFloat getAncho() { return ancho; }
};

#endif //RECTANGULO_00_CRECTANGULO_H
```

CRectangulo.cpp

```
#include "CRectangulo.h"
CRectangulo::CRectangulo(TipoFloat _largo, TipoFloat _ancho)
{
    largo = _largo;
    ancho = _ancho;
}

TipoFloat CRectangulo::Area()
{
    return (largo*ancho);
}

TipoFloat CRectangulo::Perimetro()
{
    return( 2*largo + 2*ancho);
}
```

main.cpp

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

int main()
{
    CRectangulo R1(110.49, 55.25);
    //-- Se llama al constructor
    //-- que se ha definido en la clase

    cout <<"\nArea y perimetro de R1 \n";
    cout <<"El area      : " << R1.Area() << "\n";
    cout <<"El perimetro : " << R1.Perimetro();
    return 0;
}
```

Según la sintaxis de las últimas versiones de C++, se puede declarar el constructor así:

CRectangulo.h

```
#ifndef RECTANGULO_00_CRECTANGULO_H
#define RECTANGULO_00_CRECTANGULO_H

typedef float TipoFloat;

class CRectangulo
{
private:
    TipoFloat largo;
    TipoFloat ancho;
public:
    CRectangulo(TipoFloat _largo, TipoFloat _ancho):largo(_largo),ancho(_ancho){};
    TipoFloat Area();
    TipoFloat Perimetro();
    //----metodos de acceso
    void setLargo(TipoFloat _largo) { largo = _largo; }
    void setAncho(TipoFloat _ancho) { ancho= _ancho; }
    TipoFloat getLargo() { return largo; }
    TipoFloat getAncho() { return ancho; }
};

#endif //RECTANGULO_00_CRECTANGULO_H
```

CRectangulo.cpp

```
#include "CRectangulo.h"

TipoFloat CRectangulo::Area()
{
    return (largo*ancho);
}

TipoFloat CRectangulo::Perimetro()
{
    return( 2*largo + 2*ancho);
}
```

main.cpp

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

int main()
{
    CRectangulo R1(110.49, 55.25);
    //-- Se llama al constructor
    //-- que se ha definido en la clase

    cout <<"\nArea y perimetro de R1 \n";
    cout <<"El area      : " << R1.Area() << "\n";
    cout <<"El perimetro : " << R1.Perimetro();
    return 0;
}
```


Sobrecarga de funciones : Constructor

- Una función (incluyendo el constructor) puede tener muchas versiones, diferenciadas por su lista de parámetros (número, tipos y orden de los parámetros).
- La invocación o llamada a la función (o al constructor) puede optar por invocar una versión determinada haciendo coincidir la lista de parámetros.

Sobrecarga de funciones : Constructor.

La clase puede tener más de un constructor, con una cantidad diferente de parámetros.

CRectangulo.h

```
#ifndef RECTANGULO_00_CRECTANGULO_H
#define RECTANGULO_00_CRECTANGULO_H

typedef float TipoFloat;

class CRectangulo
{
private:
    TipoFloat largo;
    TipoFloat ancho;
public:
    CRectangulo(){}; //--Es necesario incluir este constructor.
    CRectangulo(TipoFloat _largo, TipoFloat _ancho):largo(_largo),ancho(_ancho){};
    TipoFloat Area();
    TipoFloat Perimetro();
    //----metodos de acceso
    void setLargo(TipoFloat _largo) { largo = _largo; }
    void setAncho(TipoFloat _ancho) { ancho= _ancho; }
    TipoFloat getLargo() { return largo; }
    TipoFloat getAncho() { return ancho; }
};

#endif //RECTANGULO_00_CRECTANGULO_H
```

CRectangulo.cpp

```
#include "CRectangulo.h"

TipoFloat CRectangulo::Area()
{
    return (largo*ancho);
}

TipoFloat CRectangulo::Perimetro()
{
    return( 2*largo + 2*ancho);
}
```

main.cpp

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

int main()
{
    CRectangulo R1;
    R1.setLargo(23.45);
    R1.setAncho(12.09);
    cout <<"Rectangulo 1 \n";
    cout <<"El area    : " << R1.Area() << "\n";
    cout <<"El perimetro : " << R1.Perimetro();

    CRectangulo R2(43.5,22.5);
    cout <<"\n\nRectangulo 2 \n";
    cout <<"El area    : " << R2.Area() << "\n";
    cout <<"El perimetro : " << R2.Perimetro();
    return 0;
}
```

Rectangulo 1
El area : 283.51
El perimetro : 71.08

Rectangulo 2
El area : 978.75
El perimetro : 132

Destructor:

- Similar a un constructor, un destructor tiene el mismo nombre como el de la clase, pero precedido con una tilde (~)
- El destructor es llamado automáticamente cuando la instancia expira.
- No tiene argumentos y no retorna un tipo.
- Solo debe haber un destructor en una clase.
- Si no se ha definido un destructor, el compilador provee un destructor que no hace nada.
- El destructor realizará la limpieza de la memoria, en particular, si la memoria ha sido asignada dinámicamente.
- Si el constructor usa *new* para asignar dinámicamente almacenamiento, el destructor deberá usar *delete* para eliminarlo.

CRectangulo.h

```
#ifndef RECTANGULO_00_CRECTANGULO_H
#define RECTANGULO_00_CRECTANGULO_H

typedef float TipoFloat;

class CRectangulo
{
private:
    TipoFloat largo;
    TipoFloat ancho;
public:
    CRectangulo(){}; //--Es necesario incluir este constructor.
    CRectangulo(TipoFloat _largo, TipoFloat _ancho):largo(_largo),ancho(_ancho){};
    virtual ~CRectangulo(){};
    TipoFloat Area();
    TipoFloat Perimetro();
    //---metodos de acceso
    void setLargo(TipoFloat _largo) { largo = _largo; }
    void setAncho(TipoFloat _ancho) { ancho= _ancho; }
    TipoFloat getLargo() { return largo; }
    TipoFloat getAncho() { return ancho; }
};

#endif //RECTANGULO_00_CRECTANGULO_H
```

CRectangulo.cpp

```
#include "CRectangulo.h"

TipoFloat CRectangulo::Area()
{
    return (largo*ancho);
}

TipoFloat CRectangulo::Perimetro()
{
    return( 2*largo + 2*ancho);
}
```

main.cpp

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

int main()
{
    CRectangulo R1;
    R1.setLargo(23.45);
    R1.setAncho(12.09);
    cout <<"Rectangulo 1 \n";
    cout <<"El area    : " << R1.Area() << "\n";
    cout <<"El perimetro : " << R1.Perimetro();

    CRectangulo R2(43.5,22.5);
    cout <<"\n\nRectangulo 2 \n";
    cout <<"El area    : " << R2.Area() << "\n";
    cout <<"El perimetro : " << R2.Perimetro();
    return 0;
}
```

Rectangulo 1
El area : 283.51
El perimetro : 71.08

Rectangulo 2
El area : 978.75
El perimetro : 132



1. ¿Qué es un objeto?
2. ¿Qué es una clase?
3. ¿Por qué es necesario utilizar métodos de acceso?
4. ¿Una clase puede tener más de un destructor?
5. ¿Una clase puede tener varios constructores?

Creando objetos dinámicos

Utilizando archivos

Ejemplo:

En el curso de Programación Orientada a Objetos 1, se utiliza el siguiente sistema de evaluación:

$$\begin{aligned} \text{NF} = & 0.25 * \text{E1} + 0.05 * \text{C1} + 0.05 * \text{C2} + \\ & 0.10 * \text{PC1} + 0.10 * \text{PC2} + 0.10 * \text{PC3} + 0.10 * \text{PC4} + \\ & 0.10 * \text{P1} + 0.15 * \text{P2} \end{aligned}$$

Donde:

E1: Examen (1) corresponde a las evaluaciones de las clases desarrolladas en el Auditorio.

EC : Evaluación Continua: EC1 (semanas 1 - 7), EC2 (Semanas 8 - 15)

PC : Prácticas Calificada (4)

P : Proyecto (2)

Desarrolle un Programa Orientado a Objetos - (POO), que permita leer los siguientes datos de un alumno:

Código

Nombre

Apellido Paterno

Apellido Materno y sus

9 notas

y el programa calcule el promedio ponderado.

Pantalla de salida:

Codigo : 201820030

Nombre : William

Apellido Paterno : Berrocal

Apellido Materno : Alvarado

Examen 1 : 15

Evaluación continua 1 : 16

Evaluación continua 2 : 17

Practica 1 : 18

Practica 2 : 17

Practica 3 : 16

Practica 4 : 15

Proyecto 1 : 18

Proyecto 2 : 18

Codigo : 201820030

Nombre : William Berrocal Alvarado

Promedio : 16.5

Solución 1:
Utilizando un objeto estático

```

#ifndef EJEMPLO1_PROMEDIO_CALUMNO_H
#define EJEMPLO1_PROMEDIO_CALUMNO_H

#include <string> // para usar datos de tipo string
using namespace std;

typedef string TipoCadena;
typedef double TipoDouble;

class CAumno {
private:
    TipoCadena codigo;
    TipoCadena nombre;
    TipoCadena aPaterno;
    TipoCadena aMaterno;
    TipoDouble e1, c1,c2, pc1, pc2, pc3, pc4, p1, p2;
public:
    CAumno(){} //-- es el constructor por defecto
    CAumno(TipoCadena _codigo, TipoCadena _nombre, TipoCadena _aPaterno, TipoCadena _aMaterno,
           TipoDouble _e1, TipoDouble _c1, TipoDouble _c2,
           TipoDouble _pc1, TipoDouble _pc2, TipoDouble _pc3, TipoDouble _pc4,
           TipoDouble _p1, TipoDouble _p2): codigo(_codigo), nombre(_nombre), aPaterno(_aPaterno), aMaterno(_aMaterno),
                                           e1(_e1), c1(_c1), c2(_c2), pc1(_pc1), pc2(_pc2), pc3(_pc3), pc4(_pc4),
                                           p1(_p1), p2(_p2) {};

    virtual ~CAumno(){} //-- destructor
    TipoDouble PromedioPonderado();
    //--- metodos setters

```

```
//--- metodos setters
void setCodigo(TipoCadena _codigo) { codigo = _codigo;}
void setNombre(TipoCadena _nombre) { nombre = _nombre;}
void setaPaterno(TipoCadena _aPaterno) {aPaterno = _aPaterno;}
void setaMaterno(TipoCadena _aMaterno) {aMaterno = _aMaterno;}
void set_e1(TipoDouble _e1){ e1 = _e1;}
void set_c1(TipoDouble _c1){ c1 = _c1;}
void set_c2(TipoDouble _c2){ c2 = _c2;}
void set_pc1(TipoDouble _pc1){ pc1 = _pc1;}
void set_pc2(TipoDouble _pc2){ pc2 = _pc2;}
void set_pc3(TipoDouble _pc3){ pc3 = _pc3;}
void set_pc4(TipoDouble _pc4){ pc4 = _pc4;}
void set_p1(TipoDouble _p1){ p1 = _p1;}
void set_p2(TipoDouble _p2){ p2 = _p2;}

//--- metodos getters
TipoCadena getCodigo(){ return codigo;}
TipoCadena getNombre(){ return nombre;}
TipoCadena getaPaterno() { return aPaterno;}
TipoCadena getaMaterno() { return aMaterno;}
TipoDouble get_e1(){ return e1;}
TipoDouble get_c1(){ return c1;}
TipoDouble get_c2(){ return c2;}
TipoDouble get_pc1(){return pc1;}
TipoDouble get_pc2(){return pc2;}
TipoDouble get_pc3(){return pc3;}
TipoDouble get_pc4(){return pc4;}
TipoDouble get_p1(){return p1;}
TipoDouble get_p2(){return p2;}
};
#endif //EJEMPLO1_PROMEDIO_CALUMNO_H
```

```
TipoDouble CAumno::PromedioPonderado()
{
    return( 0.25*e1 + 0.05*c1 + 0.05*c2 + 0.1*pc1 +
0.1*pc2 +0.1*pc3 + 0.1*pc4 +
0.1* p1 + 0.15*p2);
}
```

main.cpp

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

```
int main()
{TipoCadena codigo, nombre, aPaterno, aMaterno;
TipoDouble e1, c1,c2, pc1, pc2, pc3, pc4, p1, p2;

cout << "Codigo           : ", cin >> codigo;
cout << "Nombre           : "; cin >> nombre;
cout << "Apellido Paterno   : "; cin >> aPaterno;
cout << "Apellido Materno   : "; cin >> aMaterno;
cout << "Examen 1           : "; cin >> e1;
cout << "Evaluacion continua 1 : "; cin >> c1;
cout << "Evaluacion continua 2 : "; cin >> c2;
cout << "Practica 1          : "; cin >> pc1;
cout << "Practica 2          : "; cin >> pc2;
cout << "Practica 3          : "; cin >> pc3;
cout << "Practica 4          : "; cin >> pc4;
cout << "Proyecto 1         : "; cin >> p1;
cout << "Proyecto 2         : "; cin >> p2;
```

```
CAumno pAlumno(codigo,nombre,aPaterno,aMaterno, e1,c1,c2,pc1,pc2,pc3,pc4,p1,p2);
```

```
cout << "\n";
cout << "Codigo   : " << pAlumno.getCodigo() << "\n";
cout << "Nombre   : " << pAlumno.getNombre() << " " << pAlumno.getaPaterno() << " " << pAlumno.getaMaterno();
cout << "\nPromedio : " << pAlumno.PromedioPonderado();
return 0;
}
```

Pantalla de salida:

```
Codigo           : 201820030
Nombre           : William
Apellido Paterno   : Berrocal
Apellido Materno   : Alvarado
Examen 1           : 15
Evaluación continua 1 : 16
Evaluación continua 2 : 17
Practica 1          : 18
Practica 2          : 17
Practica 3          : 16
Practica 4          : 15
Proyecto 1         : 18
Proyecto 2         : 18

Codigo   : 201820030
Nombre   : William Berrocal Alvarado
Promedio : 16.5
```

Solución 2:
Utilizando un objeto dinámico

Se usa la misma clase.

El cambio se da en la función main(), al momento de crear el objeto y al llamar a los métodos.

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

main.cpp

```
int main()
{TipoCadena codigo, nombre, aPaterno, aMaterno;
TipoDouble e1, c1,c2, pc1, pc2, pc3, pc4, p1, p2;
```

```
cout << "Codigo          : ", cin >> codigo;
cout << "Nombre          : "; cin >> nombre;
cout << "Apellido Paterno   : "; cin >> aPaterno;
cout << "Apellido Materno   : "; cin >> aMaterno;
cout << "Examen 1           : "; cin >> e1;
cout << "Evaluacion continua 1 : "; cin >> c1;
cout << "Evaluacion continua 2 : "; cin >> c2;
cout << "Practica 1          : "; cin >> pc1;
cout << "Practica 2          : "; cin >> pc2;
cout << "Practica 3          : "; cin >> pc3;
cout << "Practica 4          : "; cin >> pc4;
cout << "Proyecto 1          : "; cin >> p1;
cout << "Proyecto 2          : "; cin >> p2;
```

```
CAlumno *pAlumno= nullptr;
```

```
pAlumno = new CAlumno(codigo,nombre,aPaterno,aMaterno, e1,c1,c2,pc1,pc2,pc3,pc4,p1,p2 );
```

```
CAlumno &r = *pAlumno;
cout << "\n";
cout << "Codigo : " << pAlumno->getCodigo() << "\n";
cout << "Nombre : " << (* pAlumno).getNombre() << " " << r.getaPaterno() << " " << pAlumno->getaMaterno();
cout << "\nPromedio : " << pAlumno->PromedioPonderado();
delete pAlumno;
pAlumno=nullptr;
return 0;
}
```

Note que ahora pAlumno apunta a donde está un objeto de la Clase CAlumno.

Por ello se debe utilizar “->”

Solución 3:

Se leen datos a través de un método y se imprimen datos a través de un método.

Se utiliza el teclado y la pantalla de salida como si fueran archivos.

class

std::fstream

<fstream>

```
typedef basic_fstream<char> fstream;
```

Input/output file stream class



```
#ifndef EJEMPLO1_PROMEDIO_CALUMNO_H
#define EJEMPLO1_PROMEDIO_CALUMNO_H
```

```
#include <string> // para usar datos de tipo string
using namespace std;
```

```
typedef string TipoCadena;
typedef double TipoDouble;
```

```
class CAumno {
```

```
private:
```

```
    TipoCadena codigo;
    TipoCadena nombre;
    TipoCadena aPaterno;
    TipoCadena aMaterno;
    TipoDouble e1, c1, c2, pc1, pc2, pc3, pc4, p1, p2;
```

```
public:
```

```
    CAumno(){} //-- es el constructor por defecto
```

```
    CAumno(TipoCadena _codigo, TipoCadena _nombre, TipoCadena _aPaterno, TipoCadena _aMaterno,
           TipoDouble _e1, TipoDouble _c1, TipoDouble _c2,
           TipoDouble _pc1, TipoDouble _pc2, TipoDouble _pc3, TipoDouble _pc4,
           TipoDouble _p1, TipoDouble _p2): codigo(_codigo), nombre(_nombre), aPaterno(_aPaterno), aMaterno(_aMaterno),
           e1(_e1), c1(_c1), c2(_c2), pc1(_pc1), pc2(_pc2), pc3(_pc3), pc4(_pc4),
           p1(_p1), p2(_p2) {};
```

```
    virtual ~CAumno(){} //-- destructor
```

```
    TipoDouble PromedioPonderado();
```

```
    void LeerDatos(std::ostream &os, std::istream &is);
```

```
    void ImprimirDatos(std::ostream &os);
```

//--- metodos setters

```
void setCodigo(TipoCadena _codigo) { codigo = _codigo;}
void setNombre(TipoCadena _nombre) { nombre = _nombre;}
void setaPaterno(TipoCadena _aPaterno) {aPaterno = _aPaterno;}
void setaMaterno(TipoCadena _aMaterno) {aMaterno = _aMaterno;}
void set_e1(TipoDouble _e1){ e1 = _e1;}
void set_c1(TipoDouble _c1){ c1 = _c1;}
void set_c2(TipoDouble _c2){ c2 = _c2;}
void set_pc1(TipoDouble _pc1){ pc1 = _pc1;}
void set_pc2(TipoDouble _pc2){ pc2 = _pc2;}
void set_pc3(TipoDouble _pc3){ pc3 = _pc3;}
void set_pc4(TipoDouble _pc4){ pc4 = _pc4;}
void set_p1(TipoDouble _p1){ p1 = _p1;}
void set_p2(TipoDouble _p2){ p2 = _p2;}
```

//--- metodos getters

```
TipoCadena getCodigo(){ return codigo;}
TipoCadena getNombre(){ return nombre;}
TipoCadena getaPaterno() { return aPaterno;}
TipoCadena getaMaterno() { return aMaterno;}
TipoDouble get_e1(){ return e1;}
TipoDouble get_c1(){ return c1;}
TipoDouble get_c2(){ return c2;}
TipoDouble get_pc1(){return pc1;}
TipoDouble get_pc2(){return pc2;}
TipoDouble get_pc3(){return pc3;}
TipoDouble get_pc4(){return pc4;}
TipoDouble get_p1(){return p1;}
TipoDouble get_p2(){return p2;}
```

```
};
```

#endif //EJEMPLO1_PROMEDIO_CALUMNO_H

CAumno.cpp

```
#include <iostream>
#include "CAumno.h"

TipoDouble CAumno::PromedioPonderado()
{
    return( 0.25*e1 + 0.05*c1 + 0.05*c2 + 0.1*pc1 + 0.1*pc2 + 0.1*pc3 + 0.1*pc4 +
           0.1* p1 + 0.15*p2);
}

void CAumno::LeerDatos(std::ostream &os, std::istream &is)
{
    //-----
    os << "Codigo          : "; is >> codigo;
    os << "Nombre           : "; is >> nombre;
    os << "Apellido Paterno    : "; is >> aPaterno;
    os << "Apellido Materno     : "; is >> aMaterno;
    os << "Examen 1            : "; is >> e1;
    os << "Evaluacion continua 1 : "; is >> c1;
    os << "Evaluacion continua 2 : "; is >> c2;
    os << "Practica 1          : "; is >> pc1;
    os << "Practica 2          : "; is >> pc2;
    os << "Practica 3          : "; is >> pc3;
    os << "Practica 4          : "; is >> pc4;
    os << "Proyecto 1          : "; is >> p1;
    os << "Proyecto 2          : "; is >> p2;
}

void CAumno::ImprimirDatos(std::ostream &os)
{
    os << "\nDatos del alumno\n";
    os << "Codigo      : " << codigo << "\n";
    os << "Nombre      : " << nombre << " " << aPaterno << " " << aMaterno << "\n";
    os << "Promedio    : " << PromedioPonderado();
}
```

main.cpp

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

int main()
{
    CAumno *pAlumno= nullptr;

    //-- se crea el objeto, utilizando
    // el constructor por defecto
    pAlumno = new CAumno();
    pAlumno->LeerDatos(cout, cin);
    pAlumno->ImprimirDatos(cout);

    delete pAlumno;
    pAlumno=nullptr;
    return 0;
}
```

```
Codigo      : 201820030
Nombre       : William
Apellido Paterno : Berrocal
Apellido Materno : Alvarado
Examen 1      : 17
Evaluacion continua 1 : 15
Evaluacion continua 2 : 18
Practica 1     : 14
Practica 2     : 16
Practica 3     : 19
Practica 4     : 13
Proyecto 1     : 15
Proyecto 2     : 15
```

```
Datos del alumno
Codigo      : 201820030
Nombre      : William Berrocal Alvarado
Promedio    : 15.85
```

Archivos

Archivo: Introducción

Un archivo (file) es: Una colección de datos.

- **Son almacenados en un dispositivo de almacenamiento digital.**
- **Los archivos se organizan en una Jerarquía usando Folder's (o Directorios).**
- **Usan la extensión de nombre del archivo, dependiendo del sistema operativo, para identificar el tipo (formato) de archivo.**
- **En sistemas similares a Unix, todo dispositivo es manejado como si fuera un archivo: Teclado, Monitor, Impresora, Interfaces de Red, etc.**
- **Un archivo de texto, es un tipo especial de archivo que no tiene un formato específico y que contiene únicamente texto que puede ser leído por las personas.**

Pasos para usar un Archivo

- 1. Abrir el Archivo**
- 2. Usar (leer de, escribir a) el archivo**
- 3. Cerrar el Archivo**

Objetos de Archivos Stream

- El uso de archivos requiere de objetos de archivo stream
- Existen tres tipos de objetos de archivo stream:
 - (1) `ifstream`: usado para leer
 - (2) `ofstream`: usado para escribir
 - (3) `fstream`: usado para ambos leer y escribir

Nombre del Archivo

- El nombre del archivo puede ser el nombre de la ruta completa al archivo:

`c:\datos\notas.dat` en Microsoft Windows

`/usr/ubuntu/notas.dat` en linux

esto le indica al compilador exactamente donde ubicarlo.

- El nombre del archivo puede ser también un nombre simple:

`notas.dat`

este debe estar en el mismo directorio del programa ejecutable, o en el directorio por defecto del compilador.

ifstream: Abriendo un Archivo para Leer

- Crea un objeto `ifstream` en tu programa
`ifstream archivoEntrada;`
- Abre el archivo para pasar su nombre a la función miembro `open` del objeto stream
`archivoEntrada.open("notas.dat");`

ofstream: Abriendo un Archivo para Escribir

- Crea un objeto `ofstream` en tu programa
`ofstream archivoSalida;`
- Abre el archivo por pasar el nombre a la función miembro
open del objeto stream

```
archivoSalida.open("notas.dat");
```

fstream: Archivo para Leer o Escribir

- El objeto `fstream` puede ser usado para Leer o Escribir

```
fstream archivoEntradaSalida;
```

- Para Leer se debe de especificar `ios::in` como el segundo argumento para abrir el archivo

```
archivoEntradaSalida.open("notas.dat",ios::in);
```

- Para Escribir se debe de especificar `ios::out` como el segundo argumento para abrir el archivo

```
archivoEntradaSalida.open("notas.dat",ios::out);
```

Abriendo un Archivo para Leer y Escribir

- El objeto `fstream` puede ser usado para Leer y Escribir al mismo tiempo
- Crea el objeto `fstream` y especifica ambos `ios::in` y `ios::out` como el segundo argumento para la función miembro `open`

```
fstream archivoEntradaSalida;
```

```
    archivoEntradaSalida.open("notas.dat", ios::in|ios::out);
```

Abriendo Archivos con Constructores

- Incluyen el `open`

```
fstream archivoEntrada("notas.dat", ios::in);
```

Modos de Abrir un archivo

- El modo de abrir un archivo especifica como el archivo es abierto y que se puede hacer con el archivo una vez abierto.
- `ios::in` y `ios::out` son ejemplos de modos de abrir un archivo, también llamado indicadores del modo de archivo
- Los modos de archivo pueden ser combinados y pasados como segundo argumento de la función miembro `open`

Indicadores del Modo de Archivo

<code>ios::app</code>	crea un nuevo archivo, o agrega al final de un archivo existente
<code>ios::ate</code>	va al final de un archivo existente; y puede escribir en cualquier parte del archivo
<code>ios::binary</code>	lee/escribe en modo binario (no en modo texto)
<code>ios::in</code>	abre para leer
<code>ios::out</code>	abre para escribir

`app` seek to end before each write `ate` open and seek to end immediately after opening

With `ios::app` the write position in the file is "sticky" -- all writes are at the end, no matter where you seek.

Modos por Defecto de Abrir un archivo

- **ofstream:**
 - abre solo para escribir
 - no puede ser leído el contenido del archivo
 - se crea el archivo si no existe
 - el contenido es borrado si existe el archivo
- **ifstream:**
 - abre solo para leer
 - no puede ser escrito el archivo
 - falla al abrir si el archivo no existe

Detectando errores abriendo un Archivo

Dos métodos para detectar si falla al abrir un archivo

(1) Llamar a la función miembro `fail()` en el stream

```
archivoEntrada.open("notas.dat");  
if (archivoEntrada.fail())  
{ cout << "No puede abrir el archivo";  
  exit(1);  
}
```

Detectando errores abriendo un Archivo

(2) Verificando el estatus del stream

```
archivoEntrada.open("notas.dat");  
if (!archivoEntrada.is_open())  
{ cout << "No se puede abrir el archivo";  
  exit(1);  
}
```

Usando fail() para detectar eof

Example de lectura de todos los enteros en un archivo

```
// intentando leer
int x;

archivoEntrada >> x;
while (!archivoEntrada.fail())
{ // Exitoso, no es un eof
    cout << x;
    // lee nuevamente otro entero
    archivoEntrada >> x;
}
```

Usando >> para Detectar eof

- El operador de extracción retorna el mismo valor que será retornado por la siguiente llamada a fail:
 - (`archivoEntrada >> x`) es no cero
si >> es exitoso
 - (`archivoEntrada >> x`) es cero
si >> es fin de archivo

Detectando el Final de un archivo

Se lee enteros desde un archivo y se imprimen

```
int x;
while (archivoEntrada >> x)
{
    // la lectura fue exitosa
    cout << x;
    // va al tope del bucle while e
    // intenta otra lectura
}
```

Solución 4:

- 1. Se lee datos del teclado de un alumno y se graba los datos en un archivo, cuyo nombre se ingresa como dato.**
- 2. Se lee datos desde un archivo cuyo nombre se ingresa como dato y se graba datos en un archivo cuyo nombre se ingresa como dato.**

Para la solución el proyecto consta de los siguientes archivos:

main.cpp

Definiciones.h

Definiciones.cpp

CAlumno.h

CAlumno.cpp

Definiciones.h

```
#ifndef EJEMPLO1_PROMEDIO_DEFINICIONES_H
#define EJEMPLO1_PROMEDIO_DEFINICIONES_H

#include <iostream>
using namespace std;

void AbreArchivo(string nombreFisico, std::fstream &archivo, ios_base::openmode modo);
void AbrirDosArchivos(string nomArchivoDeEntrada, std::fstream &archivoDeEntrada,
                     string nomArchivoDeSalida, std::fstream &archivoDeSalida);

#endif //EJEMPLO1_PROMEDIO_DEFINICIONES_H
```


Definiciones.cpp

```
#include <fstream>
#include "Definiciones.h"

void AbreArchivo(string nombreFisico, std::fstream &archivo, ios_base::openmode modo)
{
    //-----
    archivo.open(nombreFisico, modo);
    if(!archivo.is_open())
    {
        cout << "Error no se puede abrir el archivo ";
        exit(EXIT_FAILURE);
    }
}

void AbrirDosArchivos(string nomArchivoDeEntrada, std::fstream &archivoDeEntrada,
                     string nomArchivoDeSalida, std::fstream &archivoDeSalida)
{
    //-----
    //--- Abre el archivo de entrada
    AbreArchivo(nomArchivoDeEntrada, archivoDeEntrada, std::ios::in);
    //--- Abre el archivo de salida
    AbreArchivo(nomArchivoDeSalida, archivoDeSalida, std::ios::out);
}
```

```
#ifndef EJEMPLO1_PROMEDIO_CALUMNO_H
#define EJEMPLO1_PROMEDIO_CALUMNO_H
```

```
#include <string> // para usar datos de tipo string
using namespace std;
```

```
typedef string TipoCadena;
typedef double TipoDouble;
```

```
class CAumno {
private:
    TipoCadena codigo;
    TipoCadena nombre;
    TipoCadena aPaterno;
    TipoCadena aMaterno;
    TipoDouble e1, c1,c2, pc1, pc2, pc3, pc4, p1, p2;
public:
    CAumno(){} //-- es el constructor por defecto
    CAumno(TipoCadena _codigo, TipoCadena _nombre, TipoCadena _aPaterno, TipoCadena _aMaterno,
           TipoDouble _e1, TipoDouble _c1, TipoDouble _c2, TipoDouble _pc1, TipoDouble _pc2, TipoDouble _pc3,
           TipoDouble _pc4, TipoDouble _p1, TipoDouble _p2): codigo(_codigo), nombre(_nombre), aPaterno(_aPaterno),
                                                         aMaterno(_aMaterno),
                                                         e1(_e1), c1(_c1), c2(_c2), pc1(_pc1), pc2(_pc2), pc3(_pc3), pc4(_pc4),
                                                         p1(_p1), p2(_p2) {};

    virtual ~CAumno(){} //-- destructor
    TipoDouble PromedioPonderado();
    void LeerDatos(std::ostream &os, std::istream &is);
    void LeerDatos(std::iostream &is);
    void ImprimirDatos(std::ostream &os);
```

```
//--- metodos setters
```

```
void setCodigo(TipoCadena _codigo) { codigo = _codigo;}
void setNombre(TipoCadena _nombre) { nombre = _nombre;}
void setaPaterno(TipoCadena _aPaterno ){aPaterno = _aPaterno;}
void setaMaterno(TipoCadena _aMaterno ){aMaterno = _aMaterno;}
void set_e1(TipoDouble _e1){ e1 = _e1;}
void set_c1(TipoDouble _c1){ c1 = _c1;}
void set_c2(TipoDouble _c2){ c2 = _c2;}
void set_pc1(TipoDouble _pc1){ pc1 = _pc1;}
void set_pc2(TipoDouble _pc2){ pc2 = _pc2;}
void set_pc3(TipoDouble _pc3){ pc3 = _pc3;}
void set_pc4(TipoDouble _pc4){ pc4 = _pc4;}
void set_p1(TipoDouble _p1){ p1 = _p1;}
void set_p2(TipoDouble _p2){ p2 = _p2;}
```

```
//--- metodos getters
```

```
TipoCadena getCodigo(){ return codigo;}
TipoCadena getNombre(){ return nombre;}
TipoCadena getaPaterno() { return aPaterno;}
TipoCadena getaMaterno() { return aMaterno;}
TipoDouble get_e1(){ return e1;}
TipoDouble get_c1(){ return c1;}
TipoDouble get_c2(){ return c2;}
TipoDouble get_pc1(){return pc1;}
TipoDouble get_pc2(){return pc2;}
TipoDouble get_pc3(){return pc3;}
TipoDouble get_pc4(){return pc4;}
TipoDouble get_p1(){return p1;}
TipoDouble get_p2(){return p2;}
```

```
};
```

```
#endif //EJEMPLO1_PROMEDIO_CALUMNO_H
```

```
#include <iostream>
#include "CAlumno.h"
```

CAlumno.cpp

```
TipoDouble CAlumno::PromedioPonderado()
```

```
{
    return( 0.25*e1 + 0.05*c1 + 0.05*c2 + 0.1*pc1 + 0.1*pc2 + 0.1*pc3 + 0.1*pc4 +
            0.1* p1 + 0.15*p2);
}
```

```
void CAalumno::LeerDatos(std::ostream &os, std::istream &is)
```

```
{//-----
    os << "Codigo          : "; is >> codigo;
    os << "Nombre          : "; is >> nombre;
    os << "Apellido Paterno   : "; is >> aPaterno;
    os << "Apellido Materno   : "; is >> aMaterno;
    os << "Examen 1          : "; is >> e1;
    os << "Evaluacion continua 1 : "; is >> c1;
    os << "Evaluacion continua 2 : "; is >> c2;
    os << "Practica 1         : "; is >> pc1;
    os << "Practica 2         : "; is >> pc2;
    os << "Practica 3         : "; is >> pc3;
    os << "Practica 4         : "; is >> pc4;
    os << "Proyecto 1         : "; is >> p1;
    os << "Proyecto 2         : "; is >> p2;
}
```

```
void CAalumno::ImprimirDatos(std::ostream &os)
```

```
{
    os << "\nDatos del alumno\n";
    os << "Codigo      : " << codigo << "\n";
    os << "Nombre      : " << nombre << " " << aPaterno << " " << aMaterno << "\n";
    os << "Promedio    : " << PromedioPonderado();
}
```

```
void CAalumno::LeerDatos(std::istream &is)
```

```
{//-----
    is >> codigo;
    is >> nombre;
    is >> aPaterno;
    is >> aMaterno;
    is >> e1;
    is >> c1;
    is >> c2;
    is >> pc1;
    is >> pc2;
    is >> pc3;
    is >> pc4;
    is >> p1;
    is >> p2;
}
```

```

#include <iostream>
#include <fstream> //-- para usar archivos
#include "CAumno.h"
#include "Definiciones.h"

using namespace std;

int main()
{
    ///-- Se lee datos del teclado y se graba
    ///-- en un archivo
    CAumno *pAlumno= nullptr;
    TipoCadena nomArchivoDeSalida;
    fstream archivoDeSalida;

    cout << "Nombre del archivo : ";
    cin >> nomArchivoDeSalida;
    AbreArchivo(nomArchivoDeSalida, archivoDeSalida, std::ios::out);
    pAlumno = new CAumno();
    pAlumno->LeerDatos(cout, cin);
    pAlumno->ImprimirDatos(archivoDeSalida);
    archivoDeSalida.close();
    delete pAlumno;
    pAlumno = nullptr;

```

Continua ...

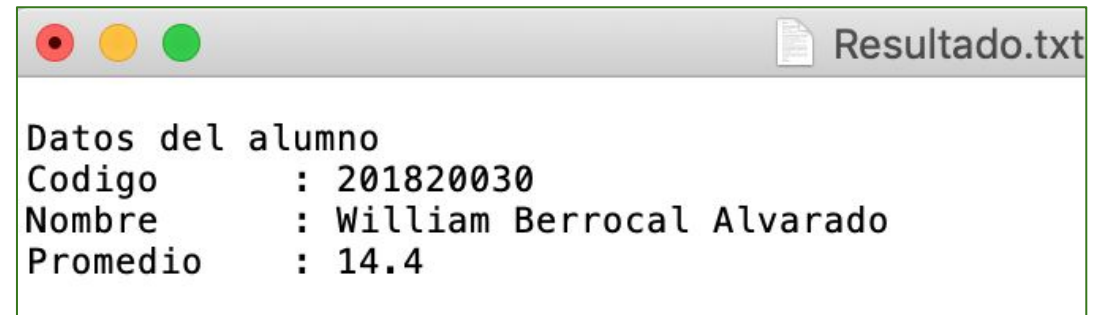
Si se ingresan estos datos:

```

Nombre del archivo : Resultado.txt
Codigo              : 201820030
Nombre              : William
Apellido Paterno    : Berrocal
Apellido Materno    : Alvarado
Examen 1            : 15
Evaluacion continua 1 : 12
Evaluacion continua 2 : 15
Practica 1          : 16
Practica 2          : 11
Practica 3          : 13
Practica 4          : 14
Proyecto 1          : 15
Proyecto 2          : 16

```

Se almacena en el archivo Resultados.txt



The screenshot shows a window titled 'Resultado.txt' with the following content:

```

Datos del alumno
Codigo      : 201820030
Nombre      : William Berrocal Alvarado
Promedio    : 14.4

```

```

//--- Se lee datos de un archivo y
//--- se graba en otro archivo
TipoCadena nArchivodeSalida, nArchivodeEntrada;
fstream aEntrada, aSalida;
cout << "\nSe lee de un archivo y se graba en otro archivo\n";
cout << "Nombre del archivo de entrada: ";
cin >> nArchivodeEntrada;
cout << "Nombre del archivo de salida : ";
cin >> nArchivodeSalida;
AbrirDosArchivos(nArchivodeEntrada, aEntrada, nArchivodeSalida, aSalida);

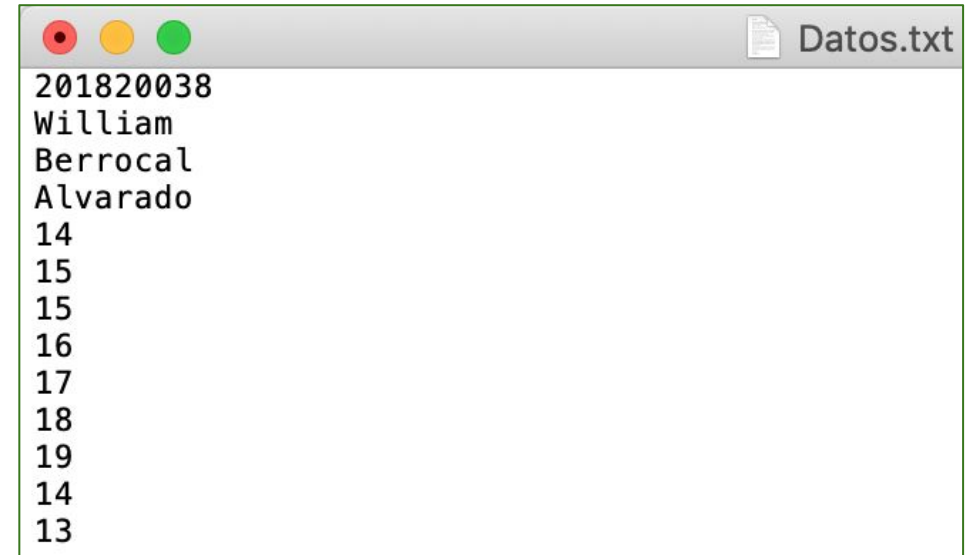
CAlumno *pA2= nullptr;
pA2 = new CAlumno();
pA2->LeerDatos(aEntrada);
pA2->ImprimirDatos(aSalida);
aEntrada.close();
aSalida.close();
delete pA2;
pA2 = nullptr;

return 0;
}

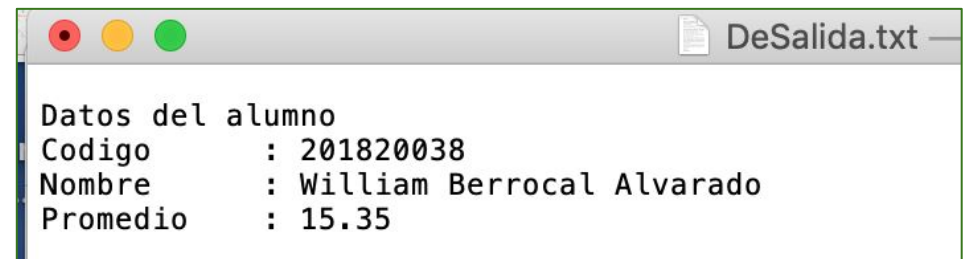
```

Consola:

Se lee de un archivo y se graba en otro archivo
Nombre del archivo de entrada: *Datos.txt*
Nombre del archivo de salida : *DeSalida.txt*



201820038
William
Berrocal
Alvarado
14
15
15
16
17
18
19
14
13



Datos del alumno
Codigo : 201820038
Nombre : William Berrocal Alvarado
Promedio : 15.35

Unidad 6:

Programación Orientada a Objetos - Parte 2

<http://bit.ly/2HRBWgq>

Profesores:

Ernesto Cuadros- Vargas, PhD.

ecuadros@utec.edu.pe

María Hilda Bermejo, M. Sc.

mbermejo@utec.edu.pe