TECNOLÓGICO NACIONAL DE MÉXICO.

INSTITUTO TECNOLÓGICO DE CHIHUAHUA.

INGENIERÍA ELECTRÓNICA.

SISTEMAS EMBEBIDOS INTELIGENTES.

ARQUITECTURA DE PROGRAMACIÓN PARA CONTROL DE HARDWARE.

LAB 3. HERENCIA Y POLIMORFISMO EN PROGRAMACIÓN ORIENTADA A OBJETOS

Anahí González Holguín 19060741

Título.

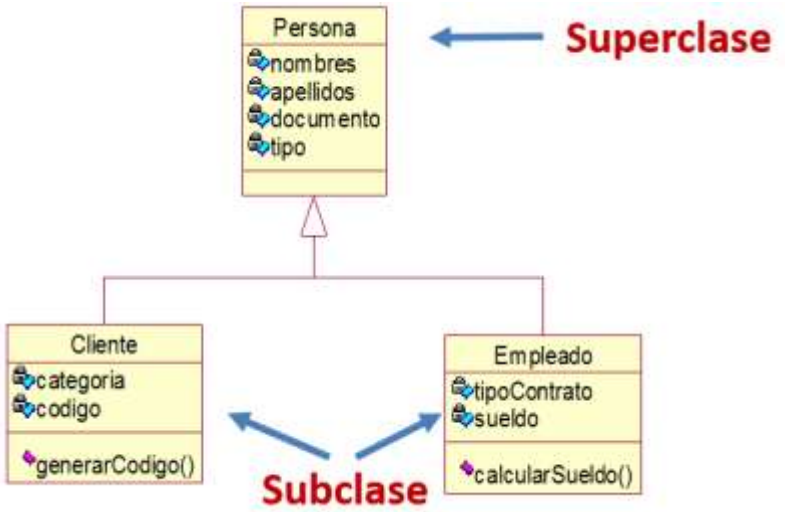
Laboratorio 3. HERENCIA Y POLIMORFISMO EN PROGRAMACIÓN ORIENTADA A OBJETOS

Antecedentes.

1. Defina y ejemplifique el concepto herencia.

Herencia.

La herencia en programación, es la capacidad de crear clases que pertenezcan al ámbito de otra clase y por lo tanto, puedas reutilizar atributos y métodos de una clase a otra. La herencia te permite establecer una jerarquía a tus clases y definir cuáles atributos y métodos deseas pasar. La clase que está en el punto más alto de esta jerarquía se puede denominar "clase madre" o "superclase" y las que heredan sus atributos y métodos se pueden llamar "subclases".



Terminología importante:

- **Superclase:** la clase cuyas características se heredan se conoce como superclase (o una clase base o una clase principal).
- **Subclase:** la clase que hereda la otra clase se conoce como subclase (o una clase derivada, clase extendida o clase hija). La subclase puede agregar sus propios campos y métodos, además de los campos y métodos de la superclase.
- **Reutilización:** la herencia respalda el concepto de "reutilización", es decir, cuando queremos crear una clase nueva y ya hay una clase que incluye parte del código que queremos, podemos derivar nuestra nueva clase de la clase existente. Al hacer esto, estamos reutilizando los campos/atributos y métodos de la clase existente.

Sintaxis:

Cuando se declara una clase D derivada de otra clase-base B, se utiliza la siguiente sintaxis:

class-key nomb-clase : <mod-acceso> clase-base {<lista-miembros>};

<mod-acceso> es un especificador opcional denominado modificador de acceso, que determina como será la accesibilidad de los miembros que se heredan de la clase-base en los objetos de las subclases.

El significado del resto de miembros es el que se indicó al tratar de la declaración de una clase, con la particularidad que, en la herencia simple, <: lista-base> se reduce a un solo identificador: clase-base.

Ejemplo

En este ejemplo puede comprobarse que cada instancia de la clase derivada tiene su propio juego de variables, tanto las privativas como las heredadas; también que unas y otras se direccionan del mismo modo.

```
#include <iostream.h>
class B {               // clase raíz
  public: int x;
};
class D : public B {    // D deriva de B
  public: int y;        // y es privativa de la clase D
};

void main() {           // ================================
  B b1;                 // b1 es instancia de B (clase raíz)
  b1.x =1;
  cout << "b1.x = "  << b1.x  << endl;

  D d1;                 // d1 es instancia de D (clase derivada)
  d1.x = 2;             // este elemento es herdado de la clase padre
  d1.y = 3;             // este elemento es privativo de d1
  D d2;                 // otra instancia de D
  d2.x = 4;
  d2.y = 5;
  cout << "d1.x = " << d1.x << endl;
  cout << "d1.y = " << d1.y << endl;
  cout << "d2.x = " << d2.x << endl;
  cout << "d2.y = " << d2.y << endl;
}
```

Salida

```
b1.x = 1
d1.x = 2
d1.y = 3
d2.x = 4
d2.y = 5
```

Es necesario introducir el concepto de nivel de acceso protegido, ya antes se había revisado los niveles de acceso público y privado. El nivel de acceso protegido, que en C++ se especifica con la palabra Protected se comporta de manera similar al nivel de acceso privado con la diferencia de que permite que los miembros de clase con este nivel de acceso puedan ser heredados a clases derivadas. Es importante aclarar que los miembros de una clase con acceso privado NO son heredables, como tampoco lo son los constructores, destructores, clases y funciones amigas, y operadores sobrecargados.

2. Defina y ejemplifique el concepto polimorfismo.

Polimorfismo es la capacidad de un objeto de adquirir varias formas. El uso más común de polimorfismo en programación orientada a objetos se da cuando se utiliza la referencia de una clase padre, para referirse al objeto de la clase hijo.

Con las funciones virtuales y el polimorfismo es posible diseñar e implementar sistemas que con extensibles con mayor facilidad. Los programas se pueden escribir para que procesen objetos en forma genérica, como objetos de clase base de todas las clases existentes en una jerarquía. Las clases que no existen durante el desarrollo del programa se pueden agregar con poca o sin modificaciones a la parte genérica del programa, siempre y cuando esas clases sean parte de la jerarquía que está siendo procesada en forma genérica. Las únicas partes del programa que necesitarán modificaciones son aquellas que requieren un conocimiento directo de la clase particular que se está agregando a la jerarquía.

Funciones virtual

Si tenemos un conjunto de clases con las formas: **Circle**, **Triangle**, **Rectangle**, **Square**, que están derivadas de la clase base **Shape**. En la programación orientada a objetos a cada una de estas figuras se le podría dar la habilidad de dibujarse a sí misma. Aunque cada clase tiene su propia función **draw**, esta es bastante diferente en cada forma. Cuando se dibuja una forma sin importar el tipo, sería agradable tratar a todas esta formas genéricamente como objetos de la clase base **Shape**.
Luego, para dibujar cualquier forma podríamos llamar a la función **draw** de la clase base **Shape** y dejar que el programa determine dinámicamente (en tiempo de ejecución) cual función **draw** de la clase derivada se debe utilizar.

Para permitir este tipo de comportamiento declaramos a **draw** de la clase base como función **virtual**, y sobreponemos a **draw** en cada una de las clases derivadas para dibujar la forma adecuada.

Una función de este tipo se declara precediendo la palabra clave **virtual**, al prototipo de la función, en la definición de la clase, por ejemplo:

```
virtual void draw();
```

Polimorfismo

C++ permite el polimorfismo, que es la habilidad de los objetos de diferentes clases que están relacionados mediante la herencia para responder en forma diferente al mismo mensaje (es decir,

a la llamada de función miembro). El mismo mensaje que se envía a muchos tipos de objetos diferentes toma "muchas formas", y de ahí viene el término polimorfismo.

Por ejemplo, si la clase Rectangle se deriva de la clase Quadrilateral, un objeto Rectangle es una versión más específica de un objeto Quadrilateral. Una operación (como el cálculo del perímetro o el área) que puede realizarse en un objeto Quadrilateral también puede realizarse en un objeto Rectangle.

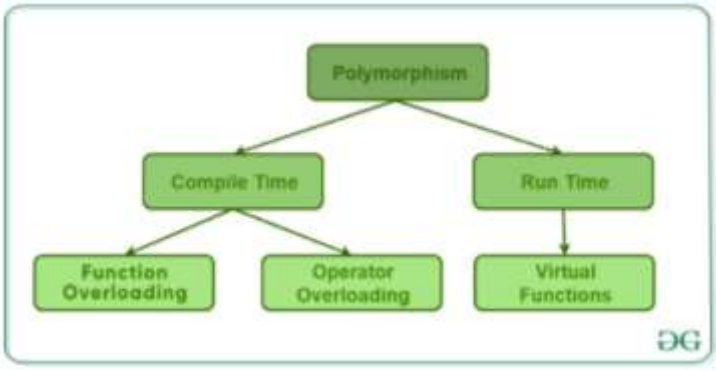
El polimorfismo se implementa por medio de funciones virtuales.

Cuando se hace una petición por medio de un apuntador de clase base (o referencia) , para utilizar una función virtual, C++ elige la función sobrepuesta correcta en la clase derivada adecuada que está asociada con ese objeto. Hay muchas veces en que una función miembro no virtual está definida en la clase base y sobrepuesta en una clase derivada. Si a una función de estas se le llama mediante un apuntador de clase base al objeto de la clase derivada, se utiliza la versión de la clase base. Si la función miembro se llama mediante un apuntador de la clase derivada, se utiliza la versión de dicha clase derivada. Este comportamiento no es polimórfico.

Mediante el uso de funciones virtual y el polimorfismo, una llamada de función miembro puede causar que sucedan diferentes acciones, dependiendo del tipo de objeto que recibe la llamada. Esto le da una capacidad expresiva tremenda al programador.

El polimorfismo promueve la extensibilidad: el software que está escrito para llamar al comportamiento polimórfico se escribe en forma independiente de los tipos de objetos a los cuales se envían los mensajes. Por lo tanto los nuevos tipos de objetos que pueden responder a los mensajes existentes se pueden agregar a un sistema, sin modificar el sistema base.

Un ejemplo de polimorfismo de la vida real, una persona al mismo tiempo puede tener diferentes características. Como un hombre al mismo tiempo es un padre, un esposo, un empleado. Entonces, la misma persona posee un comportamiento diferente en diferentes situaciones.



Ejemplo:

Se tiene una clase madre ADC y se declara una función virtual llamada captura().

```
using namespace std;

//Clase madre ADC
class ADC{
    private:

        int _num;
        int _resolution;
        float _lectura; // Lectura ya hecha la conversion
        float _lectura_v; //Lectura de entrada en Volts

    public:
        int _canal;
        int _numeroCanales;
        int _opcion; //variable para la seleccion de la frecuencia
        int _ack;
        double _Fs;

        ADC(int, int, float, float, int, int, int,int,double);  // CONSTRUCTOR1
        ADC();                                                  // CONSTRUCTOR2

        //METODOS

        void MostrarDatos();
        virtual void Captura(); //POLIMORFISMO
};
```

Luego se crea una subclase Frecuency que heredará esa función virtual y podrá utilizarla como un método propio.

```
#ifndef FRECUENCY_H
#define FRECUENCY_H
#include<iostream>
#include "adc_h.hpp"
using namespace std;

//SUBCLASE DE ADC
class Frecuency: public ADC{
  private:

  public:
    Frecuency(int, int, float, float, int, int, int,int,double);
    Frecuency();
   void Captura();
};
#endif
```

3.a. Realice un programa con el paradigma orientado a objetos.
Genere un programa que simule la configuración de un ADC

Objetivo: aplicar los conocimientos adquiridos en clase sobre el paradigma de programación orientada a objetos.

Metodología: Realizar un programa en lenguaje C++ en el que se utilicen Clases para simular la configuración de un ADC.

Materiales: DEV C++

Desarrollo:

1. Se creó el archivo “adc_h.hpp” para declarar la clase ADC con sus atributos y métodos.

```
class ADC{
    private:

        int _num;
        int _resolution;
        int _Fs;
        float _lectura;
        float _lectura_v;

    public:
        int _canal;
        int _numeroCanales;

        ADC(int, int, int, float, float, int, int); // CONSTRUCTOR1
        ADC();                                      // CONSTRUCTOR2
        //GETTERS
        int getNumCanales();
        int getResolution();
        int getFs();

        //METODOS
        void Captura();
        void MostrarDatos();
};
```

2. Se creó el archivo “adc_setup.cpp” para instanciar la clase y sus métodos e inicializar sus atributos. Se incluyó la librería math.h para utilizar funciones matemáticas y adc_h.hpp para incluir la clase declarada previamente.

```
#include <math.h>
#include "adc_h.hpp"

ADC::ADC(int num,int resolution,int Fs,float lectura, float lectura_v, int canal,

    _num=num;
    _resolution=resolution;
    _Fs=Fs;
    _lectura=lectura;
    _lectura_v=lectura_v;
    _canal=canal;
    _numeroCanales=numeroCanales;
}


ADC::ADC(){
    _canal=0;
    _resolution=0;
    _Fs=0;
}
```

3. A continuación se instancia el método Captura para capturar los datos, haciendo validación de las entradas y haciendo el cálculo de la lectura del ADC utilizando la función pow.

```
//MÉTODOS

void ADC::Captura(){

    cout<<endl<<"** Introduce Datos **"<<endl;
    val1:
    cout<<"Dame la resolucion (Opciones: 8, 10, 12): ";
    cin>> _resolution;
    if(_resolution!=8&&_resolution!=10&&_resolution!=12)
    {
        cout<<"Error. Ingresa una resolucion valida."<<endl;
        goto val1;
    }
    cout<<"Dame la frecuencia de muestreo: ";
    cin>> _Fs;
    cout<<"Dame la lectura en volts: ";
    cin>> _lectura_v;
    _lectura=_lectura_v * (pow(2,_resolution))/3.3;
}
```

4. Y se utiliza otro método para mostrar los datos del objeto perteneciente a la clase.

```
void ADC::MostrarDatos(){
    std::cout<<"\n"<<"*** Imprimiendo Datos *** "<<std::endl;
    std::cout<<"Resolucion: "<<_resolution<<std::endl;
    std::cout<<"Fs: "<<_Fs<<std::endl;
    std::cout<<"Lectura en volts: "<<_lectura_v<<std::endl;
    std::cout<<"Lectura: "<<_lectura<<std::endl;
}
```

Se crea el archivo main.cpp, también se agrega #include "adc_h.hpp". Se pide el número de canales y se valida su entrada. Se crea el objeto de clase ADC tipo apuntador y se utiliza new para reservar una localidad de memoria en el heap de tamaño numeroCanales y de tipo ADC. Se tiene que hacer esto porque C++ no permite el típico adc [numeroCanales] como se haría en C.

```
int numeroCanales;
val2:
cout<<"Introduzca el numero de canales (Opciones: 1-32)"<<endl;
cin>>numeroCanales;
if(numeroCanales>32||numeroCanales<1)
{
    cout<<"Error. Ingresa un numero de canales valido"<<std::endl;
    goto val2;
}

cout<<"Se activaron "<<numeroCanales<<" canales"<<endl;
ADC *adc1 = new ADC[numeroCanales];
```

Se capturan los datos del arreglo de tipo ADC creado, se procesan dentro de la función Captura() y después se muestran.

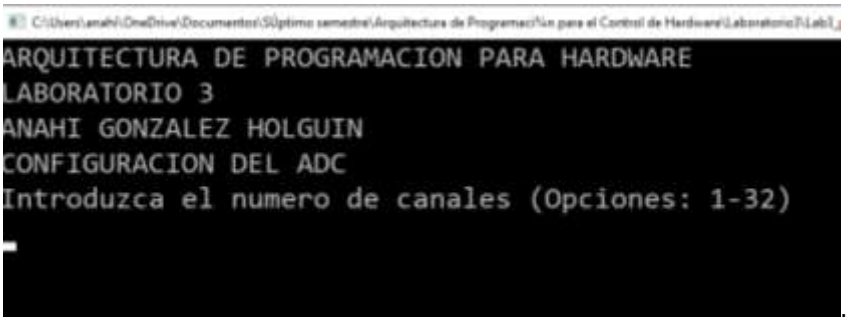
```
//CAPTURA DE LOS DATOS
for (i=0;i<numeroCanales;i++)
{
    cout<<"Canal numero "<<i+1<<endl;
    adc1[i].Captura();
}

//SALIDA DE LOS DATOS
  for (int i=0;i<numeroCanales;i++)
{
    cout<<"Canal numero "<<i+1<<endl;
    adc1[i].MostrarDatos();
}

delete [] adc1;

return 0;
```

Resultados:

Se obtienen los resultados esperados. A continuación se muestra el programa en ejecución:

Entradas:

- Se introduce el número de canales, para este caso se ingresa un 2.



- Se configuran ambos canales y se muestra como al introducir una resolución inválida el programa busca que el usuario lo corrija.

```
CONFIGURACION DEL ADC
Introduzca el numero de canales (Opciones: 1-32)
2
Se activaron 2 canales
Canal numero 1

** Introduce Datos **
Dame la resolucion (Opciones: 8, 10, 12): 8
Dame la frecuencia de muestreo: 10
Dame la lectura en volts: 1
Canal numero 2

** Introduce Datos **
Dame la resolucion (Opciones: 8, 10, 12): 11
Error. Ingresa una resolucion valida.
Dame la resolucion (Opciones: 8, 10, 12): 10
Dame la frecuencia de muestreo: 50
Dame la lectura en volts: 2
Canal numero 1
```

- Se imprimen los datos con los atributos del objeto y se observa la lectura que arroja el ADC una vez que hace la conversión.

```
*** Imprimiendo Datos ***
Resolucion: 8
Fs: 10
Lectura en volts: 1
Lectura: 77.5758
Canal numero 2

*** Imprimiendo Datos ***
Resolucion: 10
Fs: 50
Lectura en volts: 2
Lectura: 620.606
```

Conclusiones.

Las clases nos permiten crear tipos de objetos no sólo con sus atributos (como las estructuras) sino también con métodos. De esta manera podemos construir nuestro código basados en objetos de cierto tipo que responden a cierto comportamiento y que utilizan ciertas variables. También nos permite encapsular parte de la información del objeto, algo que no se podía con estructuras. Por lo que nuestro programa se vuelve más eficiente y más robusto.

4. Cambie el programa para implementar esta parte con polimorfismo y con herencia.

Objetivo: aplicar los conocimientos adquiridos en clase sobre herencia y polimorfismo.

Metodología: Modificar el programa anterior para que ahora se utilicen subclases (herencia) y funciones virtuales (polimorfismo).

Materiales: DEV C++

Desarrollo:

1. Se modifican los atributos de la clase. Agregando ack y opción; además, haciendo pública la frecuencia.

```
class ADC{
    private:

        int _num;
        int _resolution;
        float _lectura; // Lectura ya hecha la conversión
        float _lectura_v; //Lectura de entrada en Volts

    public:
        int _canal;
        int _numeroCanales;
        int _opcion; //variable para la seleccion de la frecuencia
        int _ack;
        double _Fs;
```

2. Se modifica el método Captura convirtiéndolo en una función virtual y se hacen los debidos cambios en su constructor, así como en el archivo adc_setup.cpp donde se inicializan los atributos de la clase.

```
  //METODOS

  void MostrarDatos();
  virtual void Captura(); //POLIMORFISMO
```

3. En el archivo adc_setup.cpp también se modifica la función captura, para elegir la opción de cómo procesar la frecuencia, si pedirla al usuario o calcularla con ack.

```
val3:
cout<<"Opciones para fruencia: "<<endl;
cout<<"1.Ingresarla manualmente"<<endl;
cout<<"2.Calcularla mediante ack"<<endl;
cin>> _opcion;
if(_opcion!=1&&_opcion!=2)
{
    cout<<"Error. Ingresa una opcion valida."<<endl;
    goto val3;
}
if(_opcion==1)
{
    cout<<"Dame la frecuencia: ";
    cin>> _Fs;
}
```

4. Después se creó el archivo frecuency_h.hpp donde se declara la subclase de ADC llamada Frecuency, que será el tipo de objeto que calculará la frecuencia cuando la opción sea 2.

```
//SUBCLASE DE ADC
class Frecuency: public ADC{
  private:

  public:
    Frecuency(int, int, float, float, int, int, int,int,double);
    Frecuency();
   void Captura();
};
```

5. Luego se crea el archivo frecuency_setup.cpp donde se instancia la clase con sus atributos y métodos. Se instancia el método captura, al igual que en la clase madre, ya que al convertir a captura en una función virtual puede tomar el tipo de ambos objetos de clase. Aquí se calcula la frecuencia con base a “ack” y se valida al respectiva entrada.

```
//MÉTODO DE LA SUBCLASE
void Frecuency::Captura(){
  ADC::Captura();
  if(_opcion==2)
  {
    valido4:
    cout<<"Ingresa ACK (Opciones: 2,4,8,16,32,64)"<<endl; //INGRESAR ACK
    cin>>_ack;
    if( _ack!=2 && _ack!=4 && _ack!=8 && _ack!=16 && _ack!=32 &&_ack!=64)
    {
        cout<<"Error. Ingresa un valor de ACK valido"<<endl;
        goto valido4;
    }                                                    //CALCULAR FRECUENCIA
    _Fs=8000/_ack;
  }

}
```

6. Finalmente, se modifica main.cpp cambiando únicamente la línea que crea el objeto. Se cambia el tipo ADC por el tipo de la subclase Frecuency.

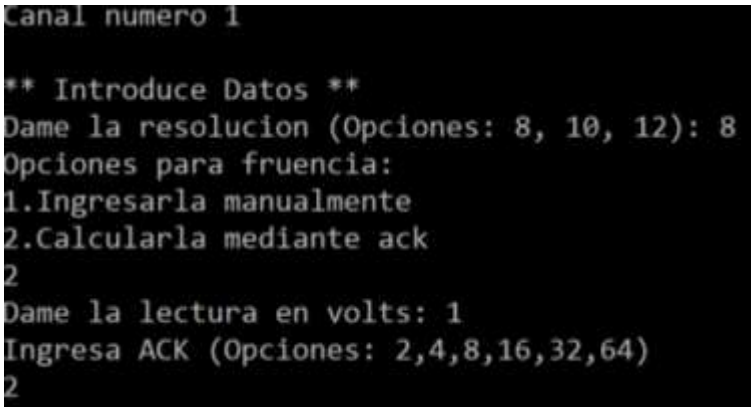
```
Frecuency *adc1 = new Frecuency[numeroCanales];
```

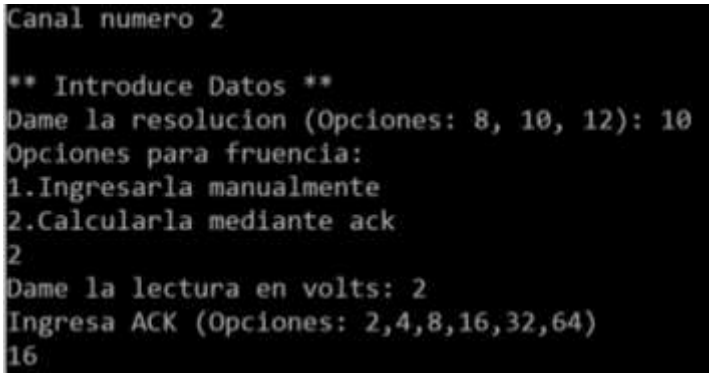
Resultados.

- Al igual que el programa anterior inicia solicitando el número de canales.

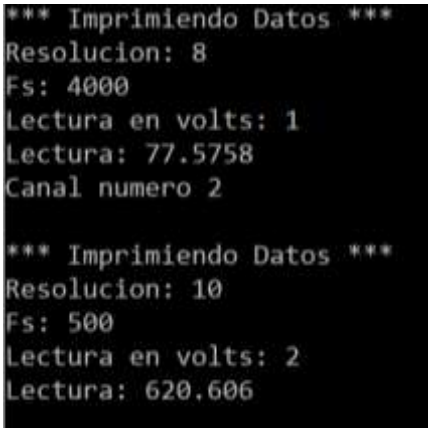
```
ARQUITECTURA DE PROGRAMACION PARA HARDWARE
LABORATORIO 3
ANAHI GONZALEZ HOLGUIN
CONFIGURACION DEL ADC
Introduzca el numero de canales (Opciones: 1-32)
```

- Se ingresa un 2 y se configuran los dos canales, eligiendo la opción 2 para calcular la frecuencia.





- Y finalmente se imprimen los datos.



Conclusión.

La herencia y el polimorfismo en programación orientada a objetos nos permite reutilizar código, así nos ahorramos tiempo y recursos. También nos permite manejar los objetos de una forma más ordenada y eficiente.

Referencias.

Escuela Técnica Superior de Ingeniería en Telecomunicaciones. (2017). Memoria Dinámica. Universidad de Málaga. España.

Soto de Giorgis, R. (2010). Estructuras Dinámicas. Escuela de Ingeniería Informática de la Pontificia Universidad Católica de Valparaiso. Chile.