

**UNIVERSIDAD NACIONAL DEL ALTIPLANO**

**FACULTAD DE INGENIERÍA MECÁNICA ELÉCTRICA,  
ELECTRÓNICA Y SISTEMAS  
ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS**



**EVALUACIÓN DE PROGRAMACIÓN  
ORIENTADA A OBJETOS EN C++**

Docente: Mg. Aldo Hernan Zanabria Galvez

Alumno: Yoel Nhelio Canaza Chagua

Curso:

Programación Orientada a Objetos II

CICLO III – SEMESTRE 2023 – II

PUNO, PERÚ

2023

# 1. Pregunta 1: Conceptos Básicos

## 1.1. Herencia: ¿Qué es la herencia en C++ y cuál es su propósito principal en la programación orientada a objetos?

La herencia es un concepto que permite la creación de nuevas clases tomando como base clases ya existentes. Una clase puede heredar los atributos y métodos de otra clase, llamada clase base, y luego extender o modificar esa funcionalidad para adaptarla a sus propias necesidades.

Se puede comparar la herencia con la relación entre padres e hijos. Un hijo hereda ciertos rasgos y características de sus padres, pero también puede desarrollar sus propias características únicas.

El propósito principal de la herencia en la programación orientada a objetos es:

- Reutilizar código
- Permitir la implementación del polimorfismo
- Permitir extender y especializar el comportamiento de la clase base según las necesidades específicas de la clase derivada.
- Modelar relaciones jerárquicas entre conceptos del mundo real.

## 1.2. Polimorfismo: Describa qué es el polimorfismo y proporcione un ejemplo práctico en C++.

Para explicarlo de manera sencilla imaginemos que tenemos tres elementos: una puerta, una ventana y una caja. Estos tres objetos tendrían el método abrir. En la programación orientada a objetos, cada clase de estos objetos debería saber cómo realizar esa operación.

Esta característica le permite a los objetos de diferente clase intercambiar mensajes con una misma estructura.

Algunos autores también definen el polimorfismo como "la capacidad de un objeto de tomar muchas formas".

Link al repositorio en GitHub en el que se encuentra el siguiente ejemplo: <https://github.com/YoelCanaza/evaluacion-poo2-1/blob/40ac42fb3afd7be90304a34d771ef4edde731d76/p-1-b.cpp>

```
1 #include <iostream>
2 #include <string>
3
4 using namespace std;
5
6 class Animal {
7 protected:
8     static int numero_animales;
9     string alimento;
10 public:
11     Animal();
12     ~Animal();
13     static int obtenerNumeroAnimales();
14
15     void comer(Animal *a){
16         cout<<"Este animal esta comiendo " << a->alimento<<endl;
17     };
18 };
19
20 int Animal::numero_animales = 0;
```

```

21
22 Animal::Animal()
23 {
24     cout<<"Creando nuevo animal ... "<<endl;
25     numero_animales += 1;
26 }
27
28 Animal::~~Animal()
29 {
30     cout<<"Borrando animal..."<<endl;
31     numero_animales -=1;
32 }
33
34 int Animal::obtenerNumeroAnimales()
35 {
36     return numero_animales;
37 }
38
39 class Herviboro : public Animal {
40 public:
41     Herviboro():Animal(){
42         this->alimento = "plantas ";
43     }
44 };
45
46 class Carnivoro : public Animal {
47 public:
48     Carnivoro():Animal(){
49         this->alimento = "carne ";
50     }
51 };
52
53 int main(){
54     Animal *a = new Animal();
55     Herviboro *h = new Herviboro();
56     Carnivoro *c = new Carnivoro();
57
58     cout<<"Numero de animeles: "<<Animal::obtenerNumeroAnimales()<<endl;
59
60     a->comer(h);
61
62     a->comer(c);
63
64     delete a;
65     delete h;
66     delete c;
67     cout<<"Numero de animales: "<<Animal::obtenerNumeroAnimales()<<endl;
68 }

```

Listing 1: Polimorfismo en C++.

En el ejemplo anterior podemos ver que la misma función o método comer puede recibir cualquier tipo de animal, y sin importar cuál sea el tipo que reciba, va a poder obtener su nombre.

## 2. Pregunta 2: Implementación de Clases

Implemente una clase llamada Rectangulo en C++ que represente un rectángulo. Incluya atributos para el ancho y el alto, así como métodos para calcular el área y el perímetro. Muestre un ejemplo de cómo usar esta clase en un programa principal.

El siguiente código se encuentra almacenado en el siguiente repositorio de GitHub:

<https://github.com/YoelCanaza/evaluacion-poo2-1/blob/f5b87ebd5733f5e520cbce8d34dc42603700p2.cpp>

```
1 #include <iostream>
2 using namespace std;
3
4 class Rectangulo
5 {
6 public:
7     Rectangulo(float ancho, float alto);
8
9     float area();
10    float perimetro();
11 protected:
12    float anchoR;
13    float altoR;
14 };
15
16 Rectangulo::Rectangulo(float ancho, float alto)
17     : anchoR(ancho), altoR(alto)
18 {
19     //El cuerpo estara vacio ya que ya se inicializaron los datos miembro
20 }
21
22 float Rectangulo::area()
23 {
24     return anchoR * altoR;
25 }
26
27 float Rectangulo::perimetro()
28 {
29     return 2 * (anchoR + altoR);
30 }
31
32
33 int main()
34 {
35     float ancho, alto;
36     cout << "Ingrese el ancho del rectangulo: "; cin >> ancho;
37     cout << "Ingrese la altura del rectangulo: "; cin>> alto;
38
39     Rectangulo r1(ancho, alto);
40
41     cout << "El area del rectangulo es: " << r1.area() << endl;
42     cout << "El perimetro del rectangulo es: " << r1.perimetro() << endl;
43 }
```

Listing 2: Implementación de clases en C++

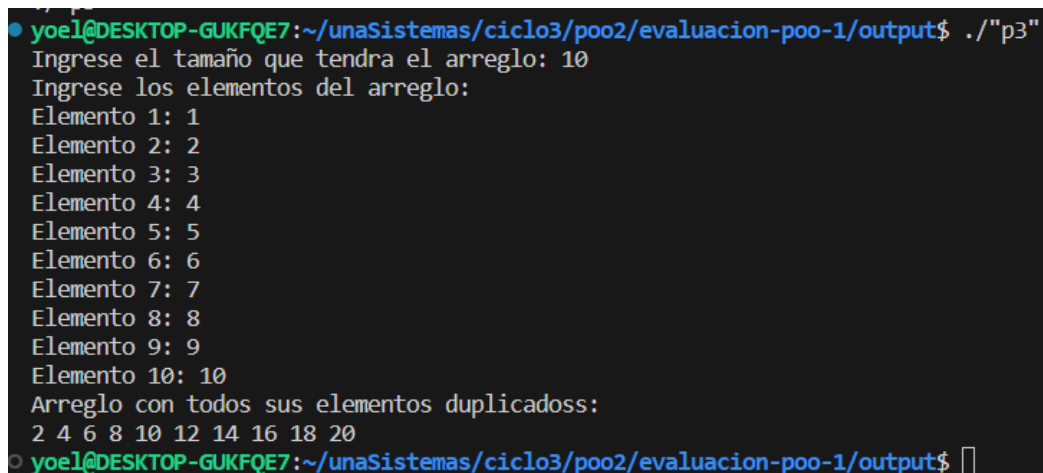
### 3. Pregunta 3: Uso de Punteros y Memoria Dinámica:

Escriba un programa en C++ que utilice punteros y memoria dinámica para crear un arreglo dinámico de enteros. Luego, realice una operación específica (por ejemplo, duplicar todos los elementos) y muestre el resultado

El código se encuentra almacenado en: <https://github.com/YoelCanaza/evaluacion-poo2-1/blob/cebac1d8a3867831acfae8376a70a25059b0549f/p3.cpp>

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6     int tamano;
7
8     cout << "Ingrese el tamaño que tendrá el arreglo: ";
9     cin >> tamano;
10
11     int* arreglo = new int[tamano];
12
13     cout << "Ingrese los elementos del arreglo:" << endl;
14     for (int i = 0; i < tamano; i++) {
15         cout << "Elemento " << i + 1 << ": "; cin >> arreglo[i];
16     }
17
18     for (int i = 0; i < tamano; ++i) {
19         arreglo[i] *= 2;
20     }
21
22     cout << "Arreglo con todos sus elementos duplicados:" << endl;
23     for (int i = 0; i < tamano; i++) {
24         cout << arreglo[i] << " ";
25     }
26     cout << endl;
27
28     delete[] arreglo;
29
30 }
```

Listing 3: Pregunta 3



```
yoel@DESKTOP-GUKFQE7:~/unaSistemas/ciclo3/poo2/evaluacion-poo-1/output$ ./"p3"
Ingrese el tamaño que tendrá el arreglo: 10
Ingrese los elementos del arreglo:
Elemento 1: 1
Elemento 2: 2
Elemento 3: 3
Elemento 4: 4
Elemento 5: 5
Elemento 6: 6
Elemento 7: 7
Elemento 8: 8
Elemento 9: 9
Elemento 10: 10
Arreglo con todos sus elementos duplicados:
2 4 6 8 10 12 14 16 18 20
yoel@DESKTOP-GUKFQE7:~/unaSistemas/ciclo3/poo2/evaluacion-poo-1/output$
```

#### **4. Pregunta 4: Manejo de Excepciones**

El manejo de excepciones en C++ es una técnica que permite lidiar con situaciones excepcionales o errores durante la ejecución de un programa. Una excepción es básicamente una señal de que algo inesperado ha ocurrido, y el manejo de excepciones proporciona una forma estructurada de gestionar estas situaciones.