

Vigésimo tercera Sesión

**Metodologías y Técnicas de Programación II**

**Programación Orientada a  
Objeto (POO)  
C++**

**Aspectos Avanzados I**

# Programa

## Introducción a la POO

Historia de la Programación  
Conceptos de POO

C++  
Mi primera Clase

## Repaso de Conceptos

Estándares de Programación  
Punteros y Memoria

E/S  
Control y Operadores

## Clases y Objetos en C++

Uso y aplicación  
Constructores  
Constantes e "inline"

Funciones Amigas  
Sobrecarga de Funciones



## Sobrecarga

De Operadores

Creación Dinámica de Objetos



## Herencia.

Tipos de Visibilidad

Herencia Múltiple

## Polimorfismo

Funciones Virtuales

Polimorfismo y Sobrecarga.

## Plantillas

Contenedores

Iteradores



# 24.3 Ejemplos Plantillas

## Arrays Genéricos: Compilar el ejercicio 1

```
template <class T>
class Array
{
public:
    Array(int nElem);
    ~Array();
    T& operator[](int indice)
    {
        return pT[indice];
    }
    const int NElementos()
    {
        return nElementos;
    }
private:
    T *pT;
    int nElementos;
};

template <class T>
Array<T>::Array(int nElem) :
    nElementos(nElem)
{
    pT = new T[nElementos];
}
```

```
template <class T>
Array<T>::~~Array()
{
    delete[] pT;
}
//-----
cont int nElementos = 10;
int main()
{
    Array<int> ArrayInt(nElementos);
    Array<float> ArrayFloat(nElementos);
    for(int i = 0; i < nElementos; i++)
        ArrayInt[i] = nElementos-i;
    for(int i = 0; i < nElementos; i++)
        ArrayFloat[i] = 1/(1+i);
    for(int i = 0; i < nElementos; i++)
    {
        cout << "ArrayInt[" << i << "] = "
              << ArrayInt[i] << endl;
        cout << "ArrayFloat[" <<i<< "] = "
              << ArrayFloat[i] << endl;
    }
    return 0;
}
```

# 24.3 Ejemplos Plantillas

## Arrays Genéricos: Cadenas – Ejercicio 2

```
cont int nElementos = 10;
int main()
{
    Array<char *> ArrayCad(nElementos);
    char cadena[20];
    for(int i = 0; i < nElementos; i++)
    {
        sprintf(cadena, "Numero: %5d", i);
        ArrayCad[i] = cadena;
    }
    strcpy(cadena, "Modificada");
    for(int i = 0; i < nElementos; i++)
        cout << "ArrayCad[" << i << "] = "
            << ArrayCad[i] << endl;
    system("pause");
    return 0;
}
```

¿Cuál es la salida por pantalla?

¿Por qué no “funciona”?

La solución pasa por crear nuestra propia clase cadena con un operador asignación que haga lo correcto y con su constructor copia.

# 24.3 Ejemplos Plantillas

## Arrays Genéricos: Ejercicio 3

```
class Cadena
{
public:
    Cadena(char *cad)
    {
        cadena = new char[strlen(cad)+1];
        strcpy(cadena, cad);
    }
    Cadena() : cadena(0) {}
    Cadena(const Cadena &c) :
        cadena(0) {*this = c;}
    ~Cadena()
    {
        if(cadena)
            delete[] cadena;
    }
    const char* Lee() const
    {
        return cadena;
    }
private:
    char *cadena;
};
```

```
Cadena & Cadena::operator=(const
Cadena &c)
{
    if(this != &c)
    {
        if(cadena)
            delete[] cadena;
        if(c.cadena)
        {
            cadena =
                new char[strlen(c.cadena)+1];
            strcpy(cadena, c.cadena);
        }
        else
            cadena = NULL;
    }
    return *this;
}
ostream& operator<<(ostream &os, const
Cadena& cad)
{
    os << cad.Lee();
    return os;
}
```

# 24.3 Punteros y Memoria

## Punteros a Objetos

Ya hemos visto que en C++ podemos tener punteros a los tipos predefinidos y también a los tipos y clases definidos por nosotros como programadores.

Al definir un puntero no estoy creando un objeto y por lo tanto no se reserva memoria para el objeto ni tampoco se llama a su constructor.

Punto x; // Hay llamada al constructor.

Punto\* y; // No hay llamada al constructor.

Punto z[10] ; // Hay 10 llamadas al constructor.

Punto\* pz= new Punto[20]; // Y aquí?

```
class Punto
{
    private:
        int x_ ;
        int y_ ;
    public:
        ...
        ...
};
```

Ejercicio 4

# 24.3 Punteros y Memoria

## Punteros a Objetos

También podemos usar otras facilidades de las variables dinámicas como son los operadores **new** y **delete**.

```
Punto *p1, *p2;
p1 = new Punto();
p2 = new Punto(3,4);
delete p2;

Punto *p;
p = new Punto[100];

// Para que llame a //
// todos los //
// constructores.
delete [] p;
```

```
class Punto
{
    private:
        int x_ ;
        int y_ ;
    public:
        ...
        ...
};
```

El operador **new** sigue los siguientes pasos:

1. Crea espacio para el objeto.
2. Llama al constructor especificado.

El operador **delete** realiza los pasos inversos:

1. Llama al destructor.
2. Libera la memoria ocupada por el objeto.

## 24.3 Punteros y Memoria

### Objetos con atributos dinámicos

Cuando una clase declara un atributo como un puntero en lugar de que sea un objeto “completo”.

Al ser un puntero tendremos:

O que asignarle memoria en algún momento.

Apuntarlos a una dirección de memoria “conocida”.

```
class Punto
{
  private:
    int x_ ;
    int y_ ;
  public:
    ...
    ...
};
```

```
class Punto
{
  private:
    int* x_ ;
    int* y_ ;
  public:
    ...
    ...
};
```

Al ser los atributos privados al “cliente” de nuestra clase le tiene que resultar transparente usar una implementación u otra.

# 24.3 Punteros y Memoria

## Creando Objetos con atributos dinámicos

Normalmente creamos objetos para que sean apuntados por nuestros atributos puntero.

Lo hacemos en el constructor del objeto que los alberga.

```
class Punto
{
    private:
        int x_ ;
        int y_ ;
    public:
        Punto(int x=0, int
y=0);
        ...
};
```

```
Punto::Punto(int x, int y)
{
    x_ = a;
    y_ = b;
};

!! MAL !!
Es lo mismo que nos pasaba con la
clase Cadena y su puntero a char.
```

Tenemos que crear dos objetos 'int' y luego asignarles su valor.

# 24.3 Punteros y Memoria

## Objetos con atributos dinámicos

Hay que tratar con cabeza a los atributos tipo puntero.

```
int main()
{
    Punto p1(20,30) ;
    Punto p2;

    p2 = p1; // !!!!!!!
}
```

```
Punto& Punto :: operator = ( Punto p)
{
    *x_ = *p.x_;
    *y_ = *p.y_;
    return *this;
}
```

El operador de asignación debe copiar los contenidos de los punteros, no los punteros.

Como el objeto sobre el que se copia ya tiene creados los atributos dinámicos, simplemente se copia lo apuntado.

Revisar el operador asignación de la clase Cadena