# LENGUAJE C++

## Tema 4 – Administración dinámica de memoria y clases abstractas.

www.profmatiasgarcia.com.ar

# Administración de Memoria

- **new:** asigna dinámicamente memoria suficiente para que en *var_ptr* exista un valor del *tipo_var* y asigna la dirección de memoria en var_ptr.
  - *var_ptr= **new** tipo_var;*
    - var_ptr==NULL si no hay memoria disponible.
  - No se requiere el calculo de la memoria total, usando el sizeof como malloc
- **delete:** solo libera dinámicamente la memoria asignada por **new**.
  - ***delete** var_ptr;*

# Ejemplo memoria dinámica

```
int main(){
   char *c; int *i = NULL;
   float **f; int n;
   c = new char[123]; // Cadena de 122 caracteres
   f = new float *[10]; // Array de 10 punteros a float
   i= new int;
   for(n = 0; n < 10; n++)
        f[n] = new float[10]; // Cada elemento del array es un array de 10 float
   // f es un array de 10*10
   f[0][0] = 10.32; f[9][9] = 21.39;
   c[0] = 'a'; c[1] = 0;
   // liberar memoria dinámica
   for(n = 0; n < 10; n++) delete[] f[n];
   delete[] f; delete[] c; delete i;
}
```

# Constructores y destructores con *new y delete.*

- ***Los Constructores*** son funciones miembro que sirven para inicializar un objeto.
- ***Los Destructores*** sirven para eliminar un objeto.
  - Tienen el mismo nombre que la clase con el símbolo ~ delante, no retornan valor y no se heredan.
  - No puede sobrecargarse porque no admite argumentos.

# Ejemplo constructores y destructores

```
class cadena {
    public:
        cadena(); // Constructor por defecto
        cadena(char *c); // Constructor desde cadena c
        cadena(int n); // Constructor para cadena de n caracteres
        cadena(const cadena &); // Constructor copia de Objeto cadena
        ~cadena(); // Destructor
        void Asignar(char *dest);
        char *Leer(char *);
    private:
        char *cad; // Puntero a char: cadena de caracteres
};
cadena::cadena() { cad=NULL;};
cadena::cadena(char *c)
{   cad = new char[strlen(c)+1]; // Reserva memoria para la cadena
    strcpy(cad, c); // Almacena la cadena
}
cadena::cadena(int n)
{   cad = new char[n+1]; // Reserva memoria para n caracteres
  cad[0] = 0; // Cadena vacía
}
cadena::cadena(const cadena &Cad) // Reservamos memoria para la nueva y la almacenamos
{ cad = new char[strlen(Cad.cad)+1]; // Reserva memoria para cadena
  strcpy(cad, Cad.cad); // Almacena la cadena
}
cadena::~cadena()
{ delete[] cad; // Libera la memoria reservada a cad
}
```

# Ejemplo constructores y destructores

```
void cadena::Asignar(char *dest)
{  delete[] cad;     // Eliminamos la cadena actual:
   cad = new char[strlen(dest)+1]; // Reserva memoria para la cadena
   strcpy(cad, dest); // Almacena la cadena
}
char *cadena::Leer(char *c)
{  strcpy(c, cad);
   return c;
}

int main(int argc, char *argv[])
{  cadena Cadena1("Cadena de prueba Nro 1");
   cadena Cadena2(Cadena1); // Cadena2 es copia de Cadena1
   cadena *Cadena3; // Cadena3 es un puntero
   char c[256];

   Cadena1.Asignar("Otra cadena diferente en Cad1"); // Modificamos Cadena1
   Cadena3 = new cadena("Cadena de prueba n° 3"); // Creamos Cadena3
   cout << "Cadena 1: " << Cadena1.Leer(c) << endl;
   cout << "Cadena 2: " << Cadena2.Leer(c) << endl;
   cout << "Cadena 3: " << Cadena3->Leer(c) << endl;
   delete Cadena3; // Destruir Cadena3.
   return 0;
} // Cadena1 y Cadena2 se destruyen automáticamente
```

**SALIDAS**

Cadena 1: Otra cadena diferente en Cad1
Cadena 2: Cadena de prueba Nro 1
Cadena 3: Cadena de prueba n° 3

# Clases abstractas

- Una clase abstracta es aquella que posee al menos una función virtual pura.
- No es posible crear objetos de una clase abstracta, estas clases sólo se usan como clases base para la declaración de clases derivadas.
- Las funciones virtuales puras serán aquellas que siempre se definirán en las clases derivadas, de modo que no será necesario definirlas en la clase base.
- A menudo se mencionan las clases abstractas como tipos de datos abstractos, en inglés: Abstract Data Type, o resumido ADT.
- Hay varias reglas a tener en cuenta con las clases abstractas:
  - No está permitido crear objetos de una clase abstracta.
  - Siempre hay que definir todas las funciones virtuales de una clase abstracta en sus clases derivadas, no hacerlo así implica que la nueva clase derivada será también abstracta.
- Para crear un ejemplo de clases abstractas, recurriremos a una clase "Persona".
- Haremos que ésta clase sea abstracta. De hecho, en nuestros programas de
- ejemplo nunca hemos declarado un objeto "Persona". Veamos un ejemplo:

# Ejemplo Clases Abstractas

```
class Persona {
    public:
        void SetPersona(char *n) {strcpy(nombre, n);}
        virtual void Mostrar() = 0; //función virtual pura;
    protected:char nombre[30];
};
class Empleado:public Persona {
    public:
        Empleado(char *n, int s);
        void Mostrar();
        int LeeSalario() const {return salario;}
        void ModificaSalario(int s) {salario = s;}
    protected: int salario;
};
class Estudiante:public Persona {
    public:
        Estudiante(char *n, float no);
        void Mostrar();
        float LeeNota() const {return nota;}
        void ModificaNota(float no) {nota = no;}
    protected:float nota;
};
```

# Ejemplo Clases Abstractas

```
Empleado::Empleado(char *n, int s)
{   SetPersona(n);  salario=s; }

void Empleado::Mostrar()
{   cout << "Empleado: " << nombre << ", Salario: " << salario << endl; }

Estudiante::Estudiante(char *n, float no)
{     SetPersona(n); nota=no; }

void Estudiante::Mostrar()
{   cout << "Estudiante: " << nombre << ", Nota: " << nota << endl; }

int main()
{     Persona *Mati = new Empleado("Matias", 1000);
      Persona *Bri = new Estudiante("Brianna", 7.56);
      char n[30];
      Mati->Mostrar();
      Bri->Mostrar();
}
```

**SALIDAS**

Empleado: Matias, Salario: 1000
Estudiante: Brianna, Nota: 7.56

# Plantillas

- Hay veces que repetimos las mismas estructuras/procesos.
  - Arrays o listas, pilas, colas, árboles, etc.
  - El código es similar, pero ciertas funciones dependen del tipo o clase de objeto que almacena…. (*Poliformismo)*
- Las ***plantillas (templates)*** permiten parametrizar clases o funciones para adaptarlas a cualquier tipo de dato.
- **Sintaxis:**
  - ***Plantilla de función:*** símil cualquier función. Se añade al principio una presentación de la clase que se usará como referencia:

```
template <class <id>[,...]>
<tipo_retorno> <identificador_de_función>(<lista_de_parámetros>)
{// Declaración de función};
```

  - ***plantilla de clase:*** símil cualquier clase. Se añade al principio una presentación de la clase que usará como referencia en la plantilla:

```
template <class <id>[,...]>
class <identificador_de_plantilla>
{// Declaración de funciones y datos miembro de la plantilla};
```

# Ejemplo Template

```
template <class T>
class Tabla {
  public:
      Tabla(int nElem);
      ~Tabla();
      bool FijarValor(int Elemento, T valor);
      bool LeerValor(int Elemento, T & valor) const;
  private:
      T * pElem;
      int CantElementos;
};
template <class T>
Tabla<T>::Tabla(int nElem)
{  CantElementos=nElem;
  pElem = new T[CantElementos];
};
template <class T>
Tabla<T>::~Tabla()
{   if (pElem) delete[] pElem;
};
```

# Ejemplo Template

```
template <class T>
bool Tabla<T>::LeerValor(int Elem, T & valor) const
{  if (Elem>-1 && Elem<CantElementos)
   {      valor=pElem[Elem];
          return true;
   }else
          return false;
};

template <class T>
bool Tabla<T>::FijarValor(int Elem, T valor)
{  if (Elem>-1 && Elem<CantElementos)
   {      pElem[Elem]=valor;
          return true;
   }else
          return false;
};
```

# Ejemplo Template

```
int main(void)
{   Tabla <int> Vint(10);
    Tabla <float> Vfloat(10);
    Tabla <char> Vch(10);
    int Xint, i;
    char Xchar;
    float Xfloat;
    for(i = 0; i < 10; i++)  Vch.FijarValor(i,97+i);
    for(i = 0; i < 10; i++)
    {   Vch.LeerValor(i, Xchar);
        cout << Xchar << endl;
    }
    for(i = 0; i < 10; i++)  Vint.FijarValor(i,i);
    for(i = 0; i < 10; i++)
    {   Vint.LeerValor(i, Xint);
        cout << Xint << endl;
    }
    for(i = 0; i < 10; i++)
    {   Xfloat=(float) (11.1)/(i+1);
        Vfloat.FijarValor(i,Xfloat);
    }
    for(i = 0; i < 10; i++)
    {   Vfloat.LeerValor(i, Xfloat);
        cout << Xfloat << endl;
    }
    return 0;
}
```

| SALIDAS |
|---|
| a |
| b |
| c |
| d |
| e |
| f |
| g |
| h |
| i |
| j |
| 0 |
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |
| 7 |
| 8 |
| 9 |

| SALIDAS |
|---|
| 11,1 |
| 5,55 |
| 3,7 |
| 2,775 |
| 2,22 |
| 1,85 |
| 1,58571 |
| 1,3875 |
| 1,23333 |
| 1,11 |

# Bibliografía & Licencia


- *Como programar en C++, 9na Ed*, Deitel, H.M. y Deitel, P.J., Pearson
- *Programación en C++, Un enfoque práctico*, Joyanes Aguilar, L., McGraw-Hill.
- *Thinking in C++, 2da Ed*, Bruce Eckel, Prentice Hall PTR.




- Autor:


Matías E. García

Prof. & Tec. en Informática Aplicada

www.profmatiasgarcia.com.ar

info@profmatiasgarcia.com.ar