

Copy Constructor

Sobrecarga de la asignación

Objetos como parámetros

Cuando un objeto se pasa a una función como un “parámetro por valor”, **NO SE EJECUTA EL CONSTRUCTOR** ya que se hace una copia bit*bit para crear un duplicado idéntico. Al final de la función **SE EJECUTA EL DESTRUCTOR** de la copia.

NOTA: Esto puede ocasionar problemas cuando el objeto tiene atributos dinámicos por lo que es importante crear una función constructor especial llamada **copy constructor**.

```
void Funcion ( unstring sS )
{
    // instrucciones
}

int main()
{
    unstring Obj1("palabra");
    Funcion(Obj1);
    //Otras Instrucciones
    return 0;
}
```

Función que regresa objetos

Cuando una función regresa un objeto, un objeto temporal es creado para mantener el valor de retorno de la función.

Después de regresar el valor, se ejecuta el destructor para este objeto.

Nota: Si el destructor libera memoria dinámica, se ocasionan problemas por lo que es importante crear un **copy constructor** y sobrecargar el **operador de asignación**.

```
unstring Funcion(char cX)
{
    unstring sS;
    //otras instrucciones
    return s;
}
int main()
{
    unstring Obj1("palabra"), Obj2;
    Obj2 = Funcion('*');
    //Otras Instrucciones
    return 0;
}
```

Inicializar un objeto con otro

Un objeto al ser declarado puede ser inicializado con el valor de otro objeto.

Nota: Esto puede ocasionar problemas cuando el objeto tiene atributos dinámicos por lo que es importante crear una función constructor especial llamada **copy constructor**.

```
int main()
{
    char cArrX[] = "palabra";
    unstring Obj1(cArrX), Obj2 = Obj1;
    Obj1.show();
    Obj2.show();
}
```

Copy Constructor

Es un constructor que recibe
un objeto de la misma clase

```
// Copy Constructor
unstring (const unstring& sO)
{
    int iN = strlen(sO.sS);
    sS = new char [iN+1];
    strcpy(sS, sO.sS);
}
```

Sobrecarga de la asignación

```
unstring operator= (const unstring &sO)
```

```
{
```

```
    if ( this != &sO )
```

```
{
```

```
        delete [] sS;
```

```
        if ( sO.sS != NULL )
```

```
{
```

```
            inti N = strlen(sO.sS);
```

```
            sS = new char [iN+1];
```

```
            strcpy(sS, sO.sS);
```

```
} else sS = NULL;
```

```
}
```

```
return *this;
```

```
}
```

Evita que se borren los valores cuando se asigna un objeto a sí mismo: A = A;

Libera el área actual del objeto. Esto es importante si el this tiene áreas dinámicas.

Ej. unstring Obj2, Obj1(x);
Obj2 = Obj2;

Hace una copia del objeto sólo si tiene elementos.

Regresa el objeto. Importante para que funcione correctamente las asignaciones múltiples.

Ejemplo: A = B = C;

Ejemplo

```
using namespace std;
class unstring
{
private:
    char *sS;
public:
    // Constructores
    unstring () { sS = NULL; }
    unstring (const char *sOtro)
    {
        int iN = strlen(sOtro);
        sS = new char [iN+1];
        strcpy(sS, sOtro);
    }
    // Destructor
    ~unstring(){ delete [] sS; }
    void show ()
    { cout << s << endl; }
```

```
// Copy Constructor
unstring (const unstring& sO)
{
    int iN = strlen(o.s);
    sS = new char [iN+1];
    strcpy(sS, sO.sS);
}
// Sobrecarga del =
unstring operator= (const unstring &sO)
{
    if ( this != &sO )
    { delete [] sS;
        if ( sO.sS != NULL )
        {
            int iN = strlen(sO.sS);
            sS = new char [iN+1];
            strcpy(sS, sO.sS);
        }
        else sS = NULL;
    }
    return *this;
};
```

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

unstring copia(unstring sA)
{
    unstring sB;
    sB = sA;
    return sB;
}

int main()
{   char cArrX[] = "palabra";
    unstring Obj1(x), Obj2 = Obj1;
    Obj1.show();
    Obj2.show();
    char cArrY[] = "prueba";
    unstring sUno(cArrY), sDos;
    sDos = copia(sUno);
    sUno.show();
    sDos.show();
    return 0;
}
```