# Guía 9: Sobrecarga de operadores.

## Objetivos

- **Comprender la sobrecarga de operadores en c++.**
- **Utilizar de manera correcta sobrecarga de operadores para resolver problemas.**

## Operadores en C++:

Los operadores son un tipo de Token (Componente lexico) que desarrollan ciertas operaciones que se han descrito.

Por ejemplo:

```
int a=5; //asignación del numero 5 a un int a
int b=5; //asignación del numero 5 a un int b
int c=a+b;//asignación y suma de int a e int b
```

Sabemos que el ejemplo contiene dos operadores; el de suma (+) y el de asignación (=), estos operadores actúan sobre objetos, en este caso de tipo int, además que sus reglas de uso y su significado están definidos por cada lenguaje.

En C++ podemos redefinir las acciones de cada uno de estos operadores, permitiendo aceptar otro tipo de dato y seguir otro comportamiento, los operadores trabajan en dos tipos de expresiones siendo: unarios y binarios.

## Operadores Unarios:

Se conocen también como operadores unitarios, estos actúan sobre un único operando.

Ejemplo:

```
b++;//operador unario ++ (incremento)
```

**Operadores unarios que se pueden sobrecargar:**

| Operador | Descripción |
|---|---|
| ! | Operador NOT lógico |
| & | Referencia, dirección de |
| ~ | Complemento a uno |
| * | Desreferencia de puntero |
| + | Unario más |
| - | Negación unaria |

| | |
|---|---|
| ++ | Incremento |
| -- | Decremento |
| New | Asignación dinámica de memoria |
| Delete | Eliminación dinámica de memoria |

## Operadores Binarios:

Un operador binario tiene dos operandos o argumentos, uno a cada lado del operador.

```
b = 4;//operador binario (asignacion)
```

### Operadores binarios que se pueden sobrecargar:

| Operador | Descripción |
|---|---|
| + | Suma |
| - | Resta |
| * | Multiplicación |
| / | División |
| % | Modulo |
| ^ | OR exclusivo |
| & | AND bit a bit |
| && | AND Lógico |
| \| | OR inclusivo bit a bit |
| \|\| | OR Lógico |
| != | Desigualdad |
| > | Mayor que |
| < | Menor que |
| >= | Mayor igual que |
| <= | Menor igual que |
| += | Suma y asignación |
| *= | Multiplicación y asignación |
| -= | Resta y asignación |
| /= | División y asignación |
| << | Desplazamiento a la izquierda |
| >> | Desplazamiento a la derecha |
| = | Asignación |
| == | Igualdad |
| -> | Selección miembro |
| ->* | Selección de puntero a miembro |

## Implementación de sobrecarga en c++:

La sobrecarga de operadores la podemos realizar siguiendo la siguiente sintaxis:

```
<tipo de dato><nombre de clase> operator <#>(lista de parametros){
      //instrucciones que forman el cuerpo del operador
}
```

Donde:

- tipo de dato: indica el tipo de dato que produce el operador sobrecargado, por lo general es la referencia al objeto.
- Nombre de clase: es el nombre de la clase a la que estamos aplicando sobrecarga.
- Operator: palabra reservada utilizada para la sobrecarga.
- # : operador que estamos sobrecargando
- Lista de parámetros: son los datos (en referencia ) o variables que procesaran la sobrecarga, por lo general si la operación es unaria no tendrá parámetros, si la operación es binaria tendrá uno o dos parámetros.

### Nota:

Cuando el operador que se desea sobrecargar pertenece o se implementa por una clase externa, debemos declarar el tipo de función **friend** para acceder a los miembros privados de la clase, y además de respetar la declaración de argumentos.

```
friend <tipo de dato><nombre de clase> operator <#>(lista de parametros){
      //instrucciones que forman el cuerpo del operador
}
```

## Ejemplo:

Cuando programamos habitualmente deseamos concatenar distintos tipos de datos, como por ejemplo concatenar un int a un string, etc, lo cual no se puede hacer de manera sencilla en este caso en c++ por la tipificación de datos.

Si ejecutamos el código siguiente veremos que los resultados no son los deseados:

```
int numero=45;
string cadena = "1";
cadena+=numero;//se esperaria que concatenara el numero
cout << "resultado " << cadena << endl;
//la salida es 1- el signo (-) es equivalente a 45 en codigo ascii
```

Por lo que el resultado no era el deseado, no se concateno los números 45 para formar 145.

## Planteamiento de la Solución:

La solución plantea crear una clase que tenga un comportamiento similar a la clase string pero con nuevos atributos, como concatenar números para formar una cadena de texto a partir de ellos.

Para ello se creara una clase llamada String y se sobrecargara los operadores +=,+, =, << y >>.

## Desarrollo:

Crear un nuevo proyecto en Code::Block usando C++, bajo el nombre de guía 9 agregar una clase llamada **String,** generarla sin el archivo de implementación (.cpp).

**Contenido String.h**

```
#include <iostream>
#include <string.h>
#include <sstream>

using namespace std;

class String
{

   private:
      int length;
      char *ptrString;
   public:

      //constructor vacio o con una cadena
      String(char cadena[]=""){
         length = strlen(cadena);//longitud de la cadena
         ptrString = new char[length];//asignando espacio de memoria

         if(ptrString==NULL){
            cout << "\nNo hay memoria"<<endl;
            exit(0);//salida del programa en ejecucion
         }
         strcpy(ptrString,cadena);

      }

      //longitud de la cadena
      int Length(){
         return strlen(ptrString);
      }
      //destructor
```

```
~String(){

   length=0;
   delete [] ptrString;//libera memoria

}
/// Sobrecarga del operador + para que permita concatenar Strings
String& operator + (String &s){//operador binario recibe como parametro una referencia

   char *temp = ptrString;        //guardamos en un puntero temporal nuestra cadena
   length = Length();
   length += s.Length();          //reservamos espacio para concatenar la cadena s
   ptrString = new char[length];

   if(ptrString==NULL){
      cout << "\nNo hay memoria"<<endl;
      exit(0);//salida del programa en ejecucion
   }
   strcpy(ptrString,temp);        //se copia a ptrString lo de temp
   strcat(ptrString,s.ptrString);  //se concatena la cadena s

   delete [] temp;              //liberar memoria
   return *this;              //se retorna un puntero this

}

/// Sobrecarga del operador >> para salida de datos
/// Se declara la funcion tipo friend para acceder a los campos privados
friend ostream& operator << (ostream & salida, String &s){
   //se le pasa como parametro el ofstream y un String  (ambos como referencia)
   salida << s.ptrString;
   return salida;

}

/// Sobrecarga del operador >> para entrada de datos
friend istream& operator >> (istream & entrada,String &s){

   entrada>> s.ptrString;
   return entrada;

}

/// Sobrecarga del operador = para asignacion de un puntero char
String&  operator = (char *s){

   this->~String();//liberar memoria
```

```
    ptrString = new char[strlen(s)];
    if(ptrString==NULL){
       cout << "\nNo hay memoria"<<endl;
       exit(0);//salida del programa en ejecucion
    }
    strcpy(ptrString,s);
    return *this;              //se retorna un puntero this

}
/// Sobrecarga del operador += para que permita concatenar String
String& operator += (String &s){

    char *temp = ptrString;        //guardamos en un puntero temporal nuestra cadena
    length += s.Length();
    ptrString = new char[length+1];//reservar espacio para la nueva cadena concatenada

    if(ptrString==NULL){
       cout << "\nNo hay memoria"<<endl;
       exit(0);//salida del programa en ejecucion
    }
    strcpy(ptrString,temp);        //se copia a ptrString lo de temp
    strcat(ptrString,s.ptrString);  //se concatena la cadena s

    delete [] temp;              //liberar memoria
    return *this;              //se retorna un puntero this
}

/// Sobrecarga del operador += para concatenar un puntero char
String& operator += (char *s){

    char *temp = ptrString;        //guardamos en un puntero temporal nuestra cadena
    length = Length();
    length += strlen(s);          //reservamos espacio para concatenar la cadena s
    ptrString = new char[length+1];

    if(ptrString==NULL){
       cout << "\nNo hay memoria"<<endl;
       exit(0);//salida del programa en ejecucion
    }
    strcpy(ptrString,temp);        //se copia a ptrString lo de temp
    strcat(ptrString,s);  //se concatena la cadena s

    delete [] temp;              //liberar memoria
    return *this;              //se retorna un puntero this

}
```

```
    /// Sobrecarga del operador += para concatenar un int
    String& operator += (int s){

       ostringstream stream;//se declara un tipo ostringstream para poder concatenar
       stream << ptrString << s;// se concatena
       string result = stream.str();//se almacena en un string
       length = result.size();

       ptrString = new char[length];
       strcpy(ptrString,result.c_str());
       return *this;            //se retorna un puntero this

    }

    /// Sobrecarga del operador = para asignar un int
    String& operator = (int s){

       this->~String();
       ostringstream stream;
       stream << ptrString << s;
       string result = stream.str();
       length = result.size();
       ptrString = new char[length];
       strcpy(ptrString,result.c_str());
       return *this;            //se retorna un puntero this

    }

};
```

**Contenido main.cpp**

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

  //manejo de sobrecarga

  String a=String("01234");
  String b=String("5");
  String c=String("6");
  cin >> c ;
  a+= b+c;
  a+= "7";
  a+=8;
```

```
    a+=c;
    cout << "\nMi dato ["<<a << "] longitud : "<< a.Length();
    return 0;
}
```

Observamos que la sobrecarga está presente en los operadores descritos anteriormente, con esto ya se pueden concatenar y asignar números enteros a nuestro objeto String, teniendo un comportamiento similar al tipo string proporcionado por c++.

## Ejercicios:

- Sobrecargar los operadores = y += para permitir el tipo de dato float.
- Solucionar el error generado al ejecutar estas línea de código en el main.cpp

```
int main()
{
    //manejo de sobrecarga
    String a=String("01234");
    String b=String("5");
    String c=String("6");
    a=b+c;
    a=c;
    a=0;
    a=c;
    cout << "\nMi dato ["<<a << "] longitud : "<< a.Length();
    return 0;
}
```

**Nota:**

Debe sobrecargar el operador = con un parámetro de referencia String.

- Sobrecargar el operador == para permitir comparar diferentes String

```
bool operator== (String &s){

      //contenido de la funcion

      return false;
}
```