Sobrecarga de operadores

Como todos los lenguajes de programación C++ tiene un conjunto de operadores (matemáticos, relacionales, lógicos: ver detalle en materiales de los cursos anteriores) para construir las instrucciones y con éstas los algoritmos. Estos operadores cumplen funciones específicas, y están diseñados para ser utilizados con los tipos de datos integrados o primitivos del lenguaje.

Lo anterior significa que el operador de suma (+) puede ser utilizado para obtener la suma entre números enteros, o float, y no entre dos objetos de tipo Fecha por ejemplo. Lo mismo podría decirse de la comparación entre 2 objetos de tipo Fecha, o de cualquier otro tipo. Sin embargo, entendiendo que la extensión del uso de los operadores hacia los tipos de datos no primitivos puede simplificar mucho el desarrollo del código, C++ nos da la posibilidad de agregarle funcionalidad a los operadores existentes. Este mecanismo se conoce como **sobrecarga de operadores**.

Veamos un ejemplo:

Dadas dos fechas, informar si son o no iguales.

```
class Fecha{
  private:
    int dia, mes, anio;
  public:
    int getDia(){return dia;}
    int getMes(){return mes;}
    int getAnio(){return anio;}
    void setDia(int d){dia=d;}
    void setMes(int m){mes=m;}
    void setAnio(int a){anio=a;}
    void Cargar();
    void Mostrar();
```

```
  };


void Fecha::Cargar(){
  cout<<"DIA: ";
  cin>>dia;
  cout<<"MES: ";
  cin>>mes;
  cout<<"ANIO: ";
  cin>>anio;
  }

void Fecha::Mostrar(){
  cout<<"DIA: ";
  cout<<dia<<"/";
  cout<<mes<<"/";
  cout<<anio;
  }

int main(){
      Fecha a, b;
      a.Cargar();
      b.Cargar();
      if(a.getDia()==b.getDia() && a.getMes()==b.getMes() &&
            a.getAnio()==b.getAnio()) cout<<”IGUALES”;
      else cout<<”DISTINTOS”;
      return 0;
}
```

Como no podemos preguntarnos de manera directa a==b, tenemos que analizar cada una de las propiedades de los objetos mediante los gets(). Pero como se indico más arriba C++ permite sobrecargar los operadores para hacerlo. **En las clases la sobrecarga de cada**

operador tiene que hacerse mediante el agregado de un método dentro de la clase.
Vemos el caso del == para la clase Fecha del ejemplo anterior:

```
bool Fecha::operator==(Fecha aux){
  if(dia!=aux.dia)return false;
  if(mes!=aux.mes)return false;
  if(anio!=aux.anio)return false;
  return true;
  }
```

El ejemplo anterior queda entonces de la siguiente manera:

```
int main(){
      Fecha a, b;
      a.Cargar();
      b.Cargar();
      if(a==b)    cout<<"IGUALES";
      else        cout<<"DISTINTOS";
      return 0;
}
```

En el ejemplo el objeto que llama a la función es a, y el que se recibe como parámetro es b: siempre el objeto que llama es el de la izquierda, y el parámetro es el valor de la derecha. Es como si pusieramos la siguiente instrucción

```
      if(a.operator==(b))
```

Si resulta de utilidad, se pueden hacer varias sobrecargas para un mismo operador, ya que C++ permite sobrecargar las funciones (esto significa tener funciones distintas con el mismo nombre, diferenciadas por los parámetros que reciben). Supongamos que querramos saber si una fecha pertenece a un mes en particular, contenida en un vector de caracteres

```
bool Fecha::operator==(const char *nombreMes){
  char meses[12][12];
```

```
  strcpy(meses[0],"ENERO");
  strcpy(meses[1],"FEBRERO");
  strcpy(meses[2],"MARZO");
  strcpy(meses[3],"ABRIL");
  strcpy(meses[4],"MAYO");
  strcpy(meses[5],"JUNIO");//FALTA COMPLETAR PARA EL RESTO DE LOS MESES
  if(strcmp(meses[mes-1],nombreMes)==0)
    return true;
  return false;
  }
```

En este caso, el parámetro es una cadena constante (por eso se la recibe como const char*), y lo que se compara es ese valor con el valor contenido en la matriz meses, en la que se copió previamente el nombre de cada mes, en la instrucción

```
if(strcmp(meses[mes-1],nombreMes)==0)
```

mes es el valor de esa propiedad en el objeto que llama a la función.

El programa permitiría entonces hacer lo siguiente:

```
int main(){
      Fecha a, b;
      a.Cargar();
      b.Cargar();
      if(a==”ENERO”)    cout<<”IGUALES”;
      else        cout<<”DISTINTOS”;
      return 0;
}
```

Se pueden sobrecargar la mayoría de los operadores existentes, pero no inventar nuevos. Como puede verse la sobrecarga consiste en agregar el método en la clase, por lo que el comportamiento del operador puede ser el que se le ocurra al programador.

En todos los casos, la sintaxis es:

valor_devuelto operator operador_a_sobrecargar(parámetro)

Links para profundizar sobre el tema

https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_C%2B%2B/Sobrecarga_de_Operadores

http://c.conclase.net/curso/?cap=035