```
/* SOBRECARGA DE OPERADORES.
1.- Objetivos
    Permitir escribir expresiones mas naturales
2.- Reglas y resticciones
    - Operadores con comportamiento por defecto, el unico es el operador =. Con memoria
    dinamica, si no se hace implementacion al final ambos objetos apuntarian a la misma memoria
    - NO SE PUEDEN SOBRECARGAR:
        * .
        * .*
        * ::
        * ?:
        * sizeof
    - SI SE PUEDEN SOBRECARGAR:
        * new[]
        * delete[]
        * ->
        * []
        * ()

    -Reglas:
        * La sobrecarda de un operador no implica la de operadores relacionados
        * No se cambia la precedenvia
        * No se cambia la asofiacitividad
        * No se puede cambiar el caracter unario/binario
        * No se pueden crear nuevos operadores
        * La semantica debs ser natural
3.- Mecanismos
Tanto los operadores Unarios como Binarios permiten dos modos de creacion
    - Metodo
    - Funcion externa

El objeto que define cual de los dos se hace es el de la izquierda de la operacion, si es un
objeto de la clase, puede hacerse como metodo (y como funcion, aunque si existe la opcion de
usar metodo se recomienda).
Si el objeto de la izquierda es de otra clase se necesita de una funcion externa.



REQUISITOS CLASE MINIMA: (con memoria dincamica)
- Constructor de copia
- Destructor
- Operador de asignacion

*/


Class Array{
    private:
        int size;
        int *ptr;
    public:
    Arrat(int 10);
    Array(const Array & otro);
    ~Array();
    inline int getSize(){return size;}
    friend ostream & operator<<(ostream &, const &array);
    friend istream& operator>>(ostream &, Array &);
    const Array& operator=(const Array&otro);
    Array & operator+(const Array&otro);
    const Array &operator+=(const Arrat &otro);
    bool operator==(const Array &otro);
    inline bool operator!=(const Array &otro){return !(this==otro);};
}




Array::Array(int size=10){
    this->size=size;
    prt=new int[size];
    for(int i=0;i<size;i++){
        ptr[i]=0;
    }
```

```
}
Array::Array(const Array & otro){
    size=otro.size;
    ptr=new int[size];
    for(int i=0;i<size;i++)
        ptr[i]=otro[i];
}
Array::~Array(){
    if(ptr!=0)
        delete []ptr;
    size=0;
    ptr=0;
}


//**********************************
//*************UTILS.CPP*************
//**********************************
friend ostream & operator<<(ostream &output, const & Array array){
    output<<endl;
    for(int i=0;i<array.size;i++)
        output<<array.ptr[i]<<" ";
    output<<endl;
return output;
}
friend istream& operator>>(ostream &input,Array &array){
    for(int i=0;i<array.size;i++)
        input>>array.ptr[i];
    return input
}


//**********************************
//*************UTILS.CPP*************
//**********************************


//**********************************
//*************ARRAY.CPP*************
//**********************************
Array & Array::operator=(const Array&otro){
    if(this!=otro){
        if(size!=otro.size){
            delete[] ptr;
            size=otro.size;
            ptr=new int[size];
        }
        for(int i=0;i<size;i++){
            ptr[i]=otro.ptr[i];
        }
    }
    return *this;
}

Array & Array::operator+(const Array & otro){
        Array * resultado = new Array(size+otro.size);
        //Copiar this
        for(int i=0;isize;i++){
            resultado->ptr[i]=ptr[i]
        }
        for(int i=0;i<otro.size;i++){
            resultado->prt[i+size]=otro.ptr[i];
        }
        return *resultado;
}

const Array & Array::operator+=(const Array & otro){
    int * ptrNuevo = new int[size+otro.size];
            //Copiar this
        for(int i=0;isize;i++){
            ptrNuevo[i]=ptr[i]
        }
        for(int i=0;i<otro.size;i++){
```

```
            ptrNuevo[i+size]=otro.ptr[i];
        }
        delete[]ptr;
        size+=otro.size;
        ptr=ptrNuevo;
        return *this;
}
const Array & Array::operator==(const Array & otro){
    bool resultado=false;
    if(this==&otro)
        resultado=true
    else{
        if(size==otro.size){
            resultado=true;
            for(int i=0;i<size&&resultado;i++){
                if(ptr[i]!=otro.prt[i])
                    resultado=false;
            }
        }
    }

    return resultado
}

//R value, se utiliza si el arrayt es constante a la hora de declararlo
int Array::operator(int indice){
    if(indice>0&&indice<size)
        return ptr[indice];
}

//L value, se utiliza en cualquier otro caso
int & Array::operator(int indice){
    if(indice>0&&indice<size)
        return ptr[indice];
}
//**********************************
//*************ARRAY.CPP************
//**********************************
```