```cpp
// *
// * File:   main.cpp
// * Author: J. Miguel Guanira E.
// ********************************************************************
// * Ejemplo sencillo de operador unario sobrecargado (++  prefijo)
// * El operador incrementa en un segundo el reloj
// ********************************************************************
// * Creado el 17 de junio de 2009, 10:55 PM
// * Corregido el 04 de octubre del 2023-2
// *
// *
#include <iostream>
#include <iomanip>
using namespace std;
#include "Reloj.h"

int main(void) {
    class Reloj reloj(10, 59, 56);

    for (int i=0; i<10; ++i,++reloj){
        // En cada ciclo del for:
        // i++ incremente en 1 el valor de i
        // ++reloj incrementa en 1 segundo el reloj
        cout<<"    "<<setfill('0');
        cout <<setw(2)<< reloj.GetHora()<<':'
             <<setw(2)<< reloj.GetMinuto()<<':'
             <<setw(2)<< reloj.GetSegundo() << endl;
        cout<<setfill(' ');
    }
    return 0;
}

// Al ejecutar este programa se muestra:
//    10:59:56
//    10:59:57
//    10:59:58
//    10:59:59
//    11:00:00
//    11:00:01
//    11:00:02
//    11:00:03
//    11:00:04
//    11:00:05

// *
// * File:   Reloj.h
// * Author: J. Miguel Guanira E.
// *
// * Created on 17 de junio de 2009, 11:03 PM
// * Corregido el 04 de octubre del 2023-2
// *

#ifndef _RELOJ_H
#define _RELOJ_H
class Reloj{
private:
    int hora;
    int minuto;
    int segundo;
public:
    Reloj(int h=0, int m=0, int s=0);
    // Los argumentos por defecto evitan sobrecargar muchas
    // vecese el constructor. Se puede invocar como:
    // Reloj reloj; <=== 0:0:0
    // Reloj reloj(3); <=== 3:0:0
    // Reloj reloj(3,25); <=== 3:25:0
```

```
67        // Reloj reloj(3,25,12); <=== 3:25:12
68
69        void SetSegundo(int segundo);
70        int GetSegundo() const;
71        void SetMinuto(int minuto);
72        int GetMinuto() const;
73        void SetHora(int hora);
74        int GetHora() const;
75        // La sobrecarga incrementa en un segundo el reloj
76        void operator ++ (void); //Operador como prefijo
77    };
78    #endif  /* _RELOJ_H */
79
80    // *
81    // * File:   Reloj.cpp
82    // * Author: J. Miguel Guanira E.
83    // *
84    // * Created on 17 de junio de 2009, 11:03 PM
85    // * Corregido el 04 de octubre del 2023-2
86    // *
87
88    #include "Reloj.h"
89
90    Reloj::Reloj(int h, int m, int s){
91        hora=h;
92        minuto=m;
93        segundo=s;
94    }
95
96    void Reloj::SetSegundo(int segundo) {
97        this->segundo = segundo;
98    }
99
100   int Reloj::GetSegundo() const {
101       return segundo;
102   }
103
104   void Reloj::SetMinuto(int minuto) {
105       this->minuto = minuto;
106   }
107
108   int Reloj::GetMinuto() const {
109       return minuto;
110   }
111
112   void Reloj::SetHora(int hora) {
113       this->hora = hora;
114   }
115
116   int Reloj::GetHora() const {
117       return hora;
118   }
119
120   void Reloj::operator ++(void){
121       segundo++;
122       if (segundo > 59){
123           segundo = 0;
124           ++minuto;
125       }
126       if (minuto > 59){
127           minuto = 0;
128           ++hora;
129       }
130       if (hora > 23)  hora = 0;
131   }
132
```

```cpp
// *
// * File:   main.cpp
// * Author: J. Miguel Guanira E.
// *
// * Created on 19 de junio de 2009, 09:15 PM
// * Corregido el 04 de octubre del 2023-2
// *
// * Ejemplo sencillo de operador sobrecargado binario (*)
// * Se sobrecarga dos veces el operador, la primra forma
// * devuelve el producto escalar de dos vectores (sólo un valor real),
// * la segunda multiplica un vector por un factor (escala)
// *
// *
#include <iostream>
#include <iomanip>
using namespace std;
#include "Vector.h"

int main(void){
    class Vector v1(10,20), v2(5,5);
    double prodEsc;
    cout.precision(2);
    cout<<fixed;

    // Producto escalar:
    prodEsc = v1*v2;
    cout << "Producto escalar = " << setw(8)<< prodEsc<< endl ;

    // Escalar un vector
    v1*5;
    cout << "Resultado de escalar el vector:"<<endl
         << "x = " <<setw(8)<< v1.GetX()<<endl
         << "y = " <<setw(8)<< v1.GetY()<<endl;
    return 0;
}

//   Al ejecutar este programa se muestra:
//   Producto escalar =   150.00
//   Resultado de escalar el vector:
//   x =    50.00
//   y =   100.00
//

// *
// * File:   Vector.h
// * Author: J. Miguel Guanira E.
// *
// * Created on 19 de junio de 2009, 09:17 PM
// * Corregido el 04 de octubre del 2023-2
// *
// *

#ifndef _VECTOR_H
#define _VECTOR_H
class Vector {
private:
    double x;
    double y;
public:
    Vector(double cx=0, double cy=0);
    void SetY(double y);
    double GetY() const;
    void SetX(double x);
    double GetX() const;
    double operator * (const class Vector &); //producto escalar
    void operator * (double); // un factor multiplica al vector
```

```
67  };
68  #endif  /* _VECTOR_H */
69
70  // *
71  // * File:   Vector.cpp
72  // * Author: J. Miguel Guanira E.
73  // *
74  // * Created on 19 de junio de 2009, 09:17 PM
75  // * Corregido el 04 de octubre del 2023-2
76  // *
77  // *
78
79  #include "Vector.h"
80
81  void Vector::SetY(double y) {
82      this->y = y;
83  }
84
85  double Vector::GetY() const {
86      return y;
87  }
88
89  void Vector::SetX(double x) {
90      this->x = x;
91  }
92
93  double Vector::GetX() const {
94      return x;
95  }
96
97  Vector::Vector(double cx, double cy){
98      x=cx;   y=cy;
99  }
100
101 double Vector::operator * (const class Vector &v){
102     return x*v.x + y*v.y;
103 // Pede hacerse también;  return x*v.GetX() + y*v.GetY();
104 }
105 void Vector::operator * (double f){
106     x*=f;         y*=f;
107 }
108
```

```
1   // *
2   // * File:   main.cpp
3   // * Author: J. Miguel Guanira E.
4   // *
5   // * Created on 19 de junio de 2009, 09:38 PM
6   // * Corregido el 04 de octubre del 2023-2
7   // *
8   // * Ejemplo sencillo de operador sobrecargado binario (+)
9   // * permite concatenar una cadena atributo de un objeto
10  // * no devuelve valor.
11  //*
12
13  #include <stdlib.h>
14  #include "Cadena.h"
15
16  int main(void){
17      class Cadena c1,c2;
18
19      c1.setCad("Juan");
20      c2.setCad("Lopez");
21
22      c1 + c2;
23      c1.imprimeCad();
24      return 0;
25  }
26
27  /* Al ejecutar este programa se muestra:
28
29      Cadena = JuanLopez
30
31  */
32
33  // *
34  // * File:   Cadena.h
35  // * Author: J. Miguel Guanira E.
36  // *
37  // * Created on 19 de junio de 2009, 09:39 PM
38  // * Corregido eel 04 de octubre del 2023-2
39  // *
40
41  #ifndef _CADENA_H
42  #define _CADENA_H
43  class Cadena{
44  private:
45      char *cad;
46  public:
47      Cadena();
48      void setCad(const char* cad);
49      void getCad(char*) const;
50      void imprimeCad(void);
51      void operator + (const class Cadena&);
52  };
53  #endif  /* _CADENA_H */
54
55  // *
56  // * File:   Cadena.cpp
57  // * Author: J. Miguel Guanira E.
58  // *
59  // * Created on 19 de junio de 2009, 09:39 PM
60  // * Corregido el 04 de octubre del 2023-2
61  // *
62
63  #include <iostream>
64  #include <iomanip>
65  using namespace std;
66  #include <string.h>
```

```
67  #include "Cadena.h"
68
69  Cadena::Cadena(){
70      cad=nullptr;
71  }
72
73  void Cadena::setCad(const char* c) {
74      if (cad != nullptr) delete cad;
75      cad = new char[strlen(c)+1];
76      strcpy(cad,c);
77  }
78
79  void Cadena::getCad(char*c) const {
80      if(cad == nullptr) c[0] = 0; //Cadena vacía
81      else strcpy(c, cad);
82  }
83
84  void Cadena::imprimeCad(void){
85      cout << "Cadena = " << cad << endl;
86  }
87
88  void Cadena::operator +(const class Cadena &c){
89      char aux[500], buffC[100];
90      c.getCad(buffC);
91      if (cad != nullptr){
92          strcpy(aux,cad);
93          strcat(aux, buffC);
94          setCad(aux);
95      }
96      else setCad(buffC);
97  }
98
```

```
1   // *
2   // * File:   main.cpp
3   // * Author: Miguel Guanira
4   // *
5   // *******************************************************
6   // * Ejemplo de operador sobrecargado binario (+, - y +=)
7   // * aplicado a numeros complejos: a + bi
8   // * devuelve un objeto y permite definir expresiones
9   // *
10  // * Created on 19 de junio de 2009, 09:54 PM
11  // * Corregido el 04 de octubre del 2023-2
12  // *******************************************************
13
14  #include <iostream>
15  #include <iomanip>
16  using namespace std;
17  #include "Complejo.h"
18
19  int main(void) {
20      class Complejo a(1.5,2.3), b(5.2,4.7), c(1.1,0.4), d;
21      cout.precision(2);
22      cout.fixed;
23
24      d = a + b - c;
25      cout << "D = " <<setw(7) << d.GetReal()
26           << " + " << setw(7) << d.GetImag() << "i" <<endl;
27
28      d = a += c;
29      cout << "A = " <<setw(7) << a.GetReal()
30           << " + " << setw(7) << a.GetImag() << "i" <<endl;
31      cout << "D = " <<setw(7) << d.GetReal()
32           << " + " << setw(7) << d.GetImag() << "i" <<endl;
33
34      return 0;
35  }
36
37  /* Al ejecutar este programa se muestra:
38      D =    5.60 +    6.60 i
39      A =    2.60 +    2.70 i
40      D =    2.60 +    2.70 i
41  */
42
43  /*
44   * File:   Complejo.h
45   * Author: J. Miguel Guanira e.
46   *
47   * Created on 19 de junio de 2009, 09:57 PM
48   * Corregido el 04 de octubre del 2023-2
49   *
50  */
51
52  #ifndef _COMPLEJO_H
53  #define _COMPLEJO_H
54  class Complejo{
55  private:
56      double real;
57      double imag;
58  public:
59      Complejo(double r=0, double i=0);
60      void SetImag(double imag);
61      double GetImag() const;
62      void SetReal(double real);
63      double GetReal() const;
64      class Complejo operator + (const class Complejo &);
65      class Complejo operator - (const class Complejo &);
66      class Complejo operator += (const class Complejo &);
```

```
67  };
68  #endif  /* _COMPLEJO_H */
69
70  /*
71   * File:   Complejo.cpp
72   * Author: J. Miguel Guanira e.
73   *
74   * Created on 19 de junio de 2009, 09:57 PM
75   * Corregido 04 de octubre del 2023-2
76  */
77  #include "Complejo.h"
78
79  Complejo::Complejo(double r, double i){
80      real=r;  imag=i;
81  }
82
83  void Complejo::SetImag(double imag) {
84      this->imag = imag;
85  }
86
87  double Complejo::GetImag() const {
88      return imag;
89  }
90
91  void Complejo::SetReal(double real) {
92      this->real = real;
93  }
94
95  double Complejo::GetReal() const {
96      return real;
97  }
98
99  ////   Si no hay constructor
100 //class Complejo Complejo::operator + (const class Complejo &c){
101 //    Complejo aux;
102 //    aux.real = real + c.GetReal();
103 //    aux.imag = imag + c.GetImag();
104 //
105 //    return aux;
106 //}
107
108 //  Habiendo un constructor:
109 class Complejo Complejo::operator + (const class Complejo &c){
110     return Complejo(real+c.GetReal() , imag+c.GetImag());
111 }
112
113 class Complejo Complejo::operator - (const class Complejo &c){
114     return Complejo(real-c.GetReal() , imag-c.GetImag());
115 }
116
117 class Complejo Complejo::operator += (const class Complejo &c){
118     real += c.GetReal();
119     imag += c.GetImag();
120     return *this;
121 }
122
```

```
1   // *
2   // * File:   main.cpp
3   // * Author: J. Miguel Guanira E.
4   // *
5   // ***********************************************
6   // * Ejemplo de operador sobrecargado unario (++)
7   // * devuelve un objeto de la clase en la que
8   // * se define. La sobrecarga implica su uso como
9   // * prefijo y como sufijo
10  // ***********************************************
11  // * Creatdo el 19 de junio de 2009, 10:16 PM
12  // * Corregido el 04 de octubre del 2023-2
13  // *
14  // *
15
16  #include <iostream>
17  #include <iomanip>
18  using namespace std;
19  #include "A.h"
20
21  int main(void){
22      class A a1(2), a2(3), a3;
23
24      cout.precision(2);
25      cout<<fixed;
26
27      cout << "Inicialmente : " << endl;
28      cout << "A1 : " << setw(7) << a1.GetX() <<endl;
29      cout << "A2 : " << setw(7) << a2.GetX() <<endl << endl;
30
31      a3 = ++a1;
32      cout << "Como prefijo A3 = ++A1 :" << endl;
33      cout << "A1 : " << setw(7) << a1.GetX() << endl;
34      cout << "A3 : " << setw(7) << a3.GetX() << endl << endl;
35
36      a3 = a2++;
37      cout << "Como sufijo  A3 = A2++" << endl;
38      cout << "A2 : " << setw(7) << a2.GetX() << endl;
39      cout << "A3 : " << setw(7) << a3.GetX() << endl;
40      return 0;
41  }
42  /* Al ejecutar este programa se muestra:
43      Inicialmente :
44      A1 :    2.00
45      A2 :    3.00
46
47      Como prefijo A3 = ++A1 :
48      A1 :    4.00
49      A3 :    4.00
50
51      Como sufijo  A3 = A2++:
52      A2 :    9.00
53      A3 :    3.00
54  */
55
56  // *
57  // * File:   A.h
58  // * Author: Miguel Guanira
59  // *
60  // * Created on 19 de junio de 2009, 10:16 PM
61  // * Corregido el 04 de octubre del 2023-2
62  // *
63
64  #ifndef _A_H
65  #define _A_H
66  class A{
```

```
67  private:
68      double x;
69  public:
70      A(double cx=0);
71      void SetX(double x);
72      double GetX() const;
73      class A operator ++ (void); // como prefijo
74      class A operator ++ (int);  // como sufijo el parámetro int es obligatorio
75  };
76  #endif  /* _A_H */
77
78  // *
79  // * File:   A.h
80  // * Author: Miguel Guanira
81  // *
82  // * Created on 19 de junio de 2009, 10:16 PM
83  // * Corregido el 04 de octubre del 2023-2
84  // *
85
86  #include "A.h"
87
88  A::A(double cx){
89      x=cx;
90  }
91
92  void A::SetX(double x) {
93      this->x = x;
94  }
95
96  double A::GetX() const {
97      return x;
98  }
99
100 class A A::operator ++ (void){  //como prefijo
101     x=x*x;
102     return *this;
103 }
104
105 class A A::operator ++ (int){  //como sufijo
106     class A aux=*this;
107     x=x*x;
108     return aux;
109 }
110
```

```cpp
// *
// * Archivo: main.cpp
// * Autor:   J. Miguel Guanira E.
// *
// * Creado el 21 de junio de 2009, 10:06 PM
// * Corregido el 04 de octubre del 2023-2
// *
// * Objeto "Quebrado"
// *

#include <iostream>
#include <iomanip>
using namespace std;

#include <stdlib.h>
#include "Quebrado.h"

int main(void) {
    class Quebrado a(2,3), b(5,6), c, d;
    int p = 3;

    cout<<"A = ";
    a.muestra();
    cout<<endl;

    cout<<"B = ";
    b.muestra();
    cout<<endl;

    d = c = a;

    cout<<"C = ";
    c.muestra();
    cout<<endl;

    cout<<"D = ";
    d.muestra();
    cout<<endl;

    c = a + b;
    cout<<"C = A + B = ";
    c.simplifica();
    c.muestra();
    cout<<endl;

    d += b;
    cout<<"D += B = ";
    d.muestra();
    cout<<endl;

    d = a + Quebrado(p);
    //d = a + p;
    cout<<"D = A + p = ";
    d.muestra();
    cout<<endl;

    d = Quebrado(p) + a; // también: d = Quebrado(3) + a;
    //d = p + a;
    cout<<"D = p + A = ";
    d.muestra();
    cout<<endl;

    double f;
    f = double(a);
    cout<<"f = "<<f<<endl;

```

```
67          return 0;
68      }
69
70      // Al ejecutar el programa se muestra:
71
72      //  A = 2/3
73      //  B = 5/6
74      //  C = 2/3
75      //  D = 2/3
76      //  C = A + B = 3/2
77      //  D += B = 3/2
78      //  D = A + p = 11/3
79      //  D = p + A = 11/3
80      //  f = 0.666667
81
82      // *
83      // * File:   Quebrado.h
84      // * Author: Miguel Guanira
85      // *
86      // * Created on 21 de junio de 2009, 10:07 PM
87      // * Corregido 04 de octubre del 2023-2
88      // *
89      // *
90
91      #ifndef _QUEBRADO_H
92      #define _QUEBRADO_H
93      class Quebrado{
94      private:
95          int numerador;
96          int denominador;
97      public:
98          Quebrado(int = 0, int = 1);
99          void setNum(int);
100         void setDen(int);
101         int getNum(void) const;
102         int getDen(void) const;
103         void muestra(void)const;
104         class Quebrado simplifica(void);
105         class Quebrado operator = (const class Quebrado&);
106         class Quebrado operator += (const class Quebrado&);
107         class Quebrado operator -= (const class Quebrado&);
108         class Quebrado operator + (const class Quebrado&);
109         class Quebrado operator - (const class Quebrado&);
110
111         operator double();
112     };
113
114     #endif  /* _QUEBRADO_H */
115
116     // *
117     // * File:   Quebrado.cpp
118     // * Author: Miguel Guanira
119     // *
120     // * Created on 21 de junio de 2009, 10:07 PM
121     // * Corregido el 04 de octubre del 2023-2
122     // *
123     // *
124     #include <iostream>
125     #include <iomanip>
126     using namespace std;
127     #include "Quebrado.h"
128
129     Quebrado::Quebrado(int n, int d){
130         numerador = n;
131         denominador = d;
132     }
```

```
133
134 void Quebrado::setNum(int n){
135     numerador = n;
136 }
137
138 void Quebrado::setDen(int d){
139     denominador = d;
140 }
141
142 int Quebrado::getNum(void)const{
143     return numerador;
144 }
145
146 int Quebrado::getDen(void)const{
147     return denominador;
148 }
149
150 void Quebrado::muestra(void)const{
151     cout<<numerador<<'/'<<denominador<<endl;
152 }
153
154 class Quebrado Quebrado::simplifica(void){
155     int mcd, temp, resto;
156
157     //Algoritmo de Euclides
158     mcd = (numerador>= 0) ? numerador : -numerador;
159     temp = (denominador>= 0) ? denominador : -denominador;
160
161     while (temp > 0){
162         resto = mcd % temp;
163         mcd = temp;
164         temp = resto;
165     }
166     if (mcd > 1){
167         numerador /= mcd;
168         denominador /= mcd;
169     }
170     return *this;
171 }
172
173 // En el siguiente método, como q entra como constante
174 // para poder usar algunos de sus métodos como getNum,
175 // éste debe haber sido definido también con la cláusula
176 // const: int getNum(void) const;
177
178 class Quebrado Quebrado::operator = (const class Quebrado  &q){
179     numerador = q.getNum();
180     denominador = q.getDen();
181     return *this;
182 }
183
184 class Quebrado Quebrado::operator += (const class Quebrado  &q){
185     numerador = numerador*q.denominador + q.numerador*denominador;
186     denominador = denominador*q.denominador;
187     return this->simplifica();
188 }
189
190 class Quebrado Quebrado::operator -= (const class Quebrado  &q){
191     numerador = numerador*q.denominador - q.numerador*denominador;
192     denominador = denominador*q.denominador;
193     return this->simplifica();
194 }
195
196 Quebrado::operator double(){
197     return double(numerador)/denominador;
198 }
```

```
199
200  class Quebrado Quebrado::operator + ( const class Quebrado  &q2){
201      return Quebrado(numerador * q2.denominador + q2.numerador * denominador,
202                      denominador * q2.denominador);
203  }
204
205  class Quebrado Quebrado::operator - (const class Quebrado  &q2){
206      return Quebrado(numerador * q2.denominador - q2.numerador * denominador,
207                      denominador * q2.denominador);
208  }
209
210  //// Qizá esta forma se erntienda más
211  //class Quebrado Quebrado::operator - (class Quebrado q2){
212  //    Quebrado Aux;
213  //    Aux.numerador = numerador * q2.denominador -
214  //                    q2.numerador * denominador ;
215  //
216  //    Aux.denominador = denominador * q2.denominador;
217  //
218  //    return Aux;
219  //}
220
```

```cpp
// *
// * File:   main.cpp
// * Author: J. Miguel Guanira E.
// *
// * Created on 22 de junio de 2009, 03:55 PM
// * Corregido el 04 de octubre del 2023-2
// *
// * Ejemplo de operador sobrecargadO [] y ()
// *
#include <iostream>
#include <fstream>
#include <iomanip>
using namespace std;
#include "B.h"

int main(void) {
    class B obj;
    ifstream arch("datos.txt", ios::in); //Archivo d eenteros
    if(not arch.is_open()){
        cout<<"ERROR: No se pudo abrir el archivo datos.txt"<<endl;
        exit(1);
    }
    while (1){
        int p;
        arch >> p;
        if (arch.eof()) break;
        obj.ingresa(p);
    }

    cout<<"Datos leidos : "<<endl;
    // Por efecto de la sobrecarga el objeto parece un arreglo
    for (int i=0; i<obj.getCant(); i++)
        cout << setw(6)<< obj[i];
    cout<<endl;

    // Por efecto de la sobrecarga el objeto parece una función
    cout <<"Sobrecarga del (): "<< obj(3,5)<<endl;
    return 0;
}

// Al ejecutar el programa se obtiene :
//Datos leidos :
//    23    65   121     3    72   190     2    32
//Sobrecarga del (): 29

// *
// * File:   A.h
// * Author: J. Miguel Guanira E.
// *
// * Created on 22 de junio de 2009, 03:56 PM
// * Corregido el 04 de octubre del 2023-2
// *

#ifndef _A_H
#define _A_H
class A{
private:
    int a;
public:
    A(int = 0);
    void setA(int);
    int getA(void);
};
#endif  /* _A_H */

// *
```

```
67  // * File:   A.cpp
68  // * Author: J. Miguel Guanira E.
69  // *
70  // * Created on 22 de junio de 2009, 03:56 PM
71  // * Corregido el 04 de octubre del 2023-2
72  // *
73  #include "A.h"
74
75  A::A(int x){
76   a = x;
77  }
78
79  void A::setA(int x){
80      a = x;
81  }
82
83  int A::getA(void){
84      return a;
85  }
86
87  // *
88  // * File:   B.h
89  // * Author: Miguel Guanira
90  // *
91  // * Created on 22 de junio de 2009, 04:03 PM
92  // * Corregido el 04 de octubre del 2023-2
93  // *
94  // *
95
96  #include "A.h"
97
98  #ifndef _B_H
99  #define _B_H
100 class B    { //la clase B es una agregación
101 private:
102     int b;
103     int cant;
104     class A m[20];
105 public:
106     B(void);
107     void setB(int);
108     int getB(void);
109     int getCant(void);
110     void ingresa(int);
111     int operator[](int);
112     int operator () (int, int);
113  };
114 #endif  /* _B_H */
115
116 // *
117 // * File:   B.cpp
118 // * Author: Miguel Guanira
119 // *
120 // *
121 // * Created on 22 de junio de 2009, 04:03 PM
122 // * Corregido el 04 de octubre del 2023-2
123 // *
124 // *
125
126 #include "A.h"
127 #include "B.h"
128
129 B::B(void)
130 { cant = 0;
131 }
132
```

```
133  void B::setB(int x){
134      b = x;
135  }
136
137  int B::getB(void){
138      return b;
139  }
140
141  int B::getCant(void){
142      return cant;
143  }
144
145  void B::ingresa(int x){
146      m[cant++].setA(x);
147  }
148
149  int B::operator[](int i) {
150      return m[i].getA();
151  }
152
153  int B::operator () (int p, int q){
154      return cant*p + q;
155  }
156
```

```cpp
// *
// * Archivo: main.cpp
// * Autor:   J. Miguel Guanira E.
// *
// * Created on 15 de noviembre de 2010, 07:13 PM
// * Corregido el 04 de octubre del 2023-2
// *
// * Ejemplo de operador sobrecargado de inserción y
// * extracción de flujo: >> y <<
// *


#include <iostream>
#include <iomanip>
#include <fstream>
using namespace std;
#include <cstdlib>
#include "Persona.h"


int main(void) {
    class Persona per;
    // Desde la entrada estándar
//    cin >> per;
//    cout << per;

    // Desde Archivos de texto
    ifstream archPer("personal.csv",ios::in);
    if(not archPer.is_open()){
        cerr<<"No se pudo abrir el archivo personal.csv"<<endl;
        exit(1);
    }
    ofstream archRep("reporte.txt",ios::out);
    if(not archPer.is_open()){
        cerr<<"No se pudo abrir el archivo reporte.txt"<<endl;
        exit(1);
    }
    while (1){
        archPer>>per;
        if(archPer.eof())break;
        archRep<<per;
    }
    return 0;
}

// *
// * File:   Persona.h
// * Autor:  J. Miguel Guanira E.
// *
// * Created on 15 de noviembre de 2010, 07:13 PM
// * Corregido el 04 de octubre del 2023-2
// *
// *

#ifndef PERSONA_H
#define PERSONA_H
#include <ostream>
using namespace std;

class Persona{
private:
    int dni;
    char *nombre;
    double sueldo;
public:
    Persona(void);
```

```
67          void SetSueldo(double sueldo);
68          double GetSueldo() const;
69          void SetNombre(const char* nombre);
70          void GetNombre(char*) const;
71          void SetDni(int dni);
72          int GetDni() const;
73      };
74
75      // La sobrecarga de los operadores << y >> se deben desarrollar
76      // fuera de la clase
77
78      //Entrada estándar
79      istream & operator >>(istream &, class Persona &);
80      ostream & operator <<(ostream &, const class Persona &);
81
82      //Archivos de texto
83      ifstream & operator >>(ifstream &, class Persona &);
84      ofstream & operator <<(ofstream &, const class Persona &);
85
86      #endif  /* PERSONA_H */
87
88      // *
89      // * File:   Persona.cpp
90      // * Autor:  J. Miguel Guanira E.
91      // *
92      // * Created on 15 de noviembre de 2010, 07:13 PM
93      // * Corregido el 04 de octubre del 2023-2
94      // *
95      // *
96
97      #include <iostream>
98      #include <fstream>
99      #include <iomanip>
100     using namespace std;
101     #include <cstring>
102     #include "Persona.h"
103
104     Persona::Persona() {
105         nombre = NULL;
106     }
107
108     void Persona::SetSueldo(double sueldo) {
109         this->sueldo = sueldo;
110     }
111
112     double Persona::GetSueldo() const {
113         return sueldo;
114     }
115
116     void Persona::SetNombre(const char* nomb) {
117         if (nombre!=NULL) delete []nombre;
118         nombre = new char[strlen(nomb)+1];
119         strcpy(nombre,nomb);
120     }
121
122     void Persona::GetNombre(char*nomb) const {
123         strcpy(nomb,nombre);
124     }
125
126     void Persona::SetDni(int dni) {
127         this->dni = dni;
128     }
129
130     int Persona::GetDni() const {
131         return dni;
132     }
```

```cpp


// La sobrecarga de los operadores << y >> se deben desarrollar
// fuera de la clase

istream & operator >>(istream & in, class Persona &p){
    char nomb[200],c;
    int dni;
    double sueldo;

    cout<<"DNI: ";
    in >> dni;

    cout<<"NOMBRE: ";
    in>>ws;
    in.getline(nomb,200);
    cout<<"SUELDO: ";
    in>>sueldo;

    p.SetDni(dni);
    p.SetNombre(nomb);
    p.SetSueldo(sueldo);

    return in;
}

ostream & operator << (ostream & out, const class Persona &p){
    out.precision(2);
    out<<fixed;
    char nomb[200];

    p.GetNombre(nomb);
    out <<left << setw(10)<<p.GetDni()<<setw(40)<<nomb<<setw(15)
        <<right<<p.GetSueldo()<<endl;;

    return out;
}

ifstream & operator >>(ifstream & inArch, class Persona &p){
    char nomb[200],c;
    int dni;
    double sueldo;

    inArch >> dni;
    if(inArch.eof())return inArch;
    inArch.get();  //sacamos la coma
    inArch.getline(nomb,200,',');
    inArch >> sueldo;

    p.SetDni(dni);
    p.SetNombre(nomb);
    p.SetSueldo(sueldo);
    return inArch;
}

ofstream & operator << (ofstream & outArch, const class Persona &p){
    outArch.precision(2);
    outArch<<fixed;
    char nomb[200];

    p.GetNombre(nomb);
    outArch <<left << setw(10)<<p.GetDni()<<setw(40)<<nomb<<setw(15)
            <<right<<p.GetSueldo()<<endl;

    return outArch;
}
```