MP 20-21 Tema 1
Sobrecàrrega d'operadors
Sessió 9: Programació orientada a objectes

Sobrecàrrega d'operadors

Recuperem la classe `Complex` que vam utilitzar en un exercici anterior:

```
class Complex
{
public:
   Complex() { m_real = 0; m_img = 0; }
   void llegeix();
   void mostra() const;
   Complex suma(const Complex &c) const;
   Complex resta(const Complex &c) const;
   Complex multiplica(const Complex &c) const;
private:
   float m_real, m_img;
};
```

```
...
void Complex::mostra() const
{
   cout << m_real << "+" << m_img << "i";
}

Complex Complex::suma(const Complex &c) const
{
   Complex resultat;

   resultat.m_real = m_real + c.m_real;
   resultat.m_img = m_img + c.m_img;
   return resultat;
}
...
```

Sobrecàrrega d'operadors

Fem un programa per operar amb números complexos:

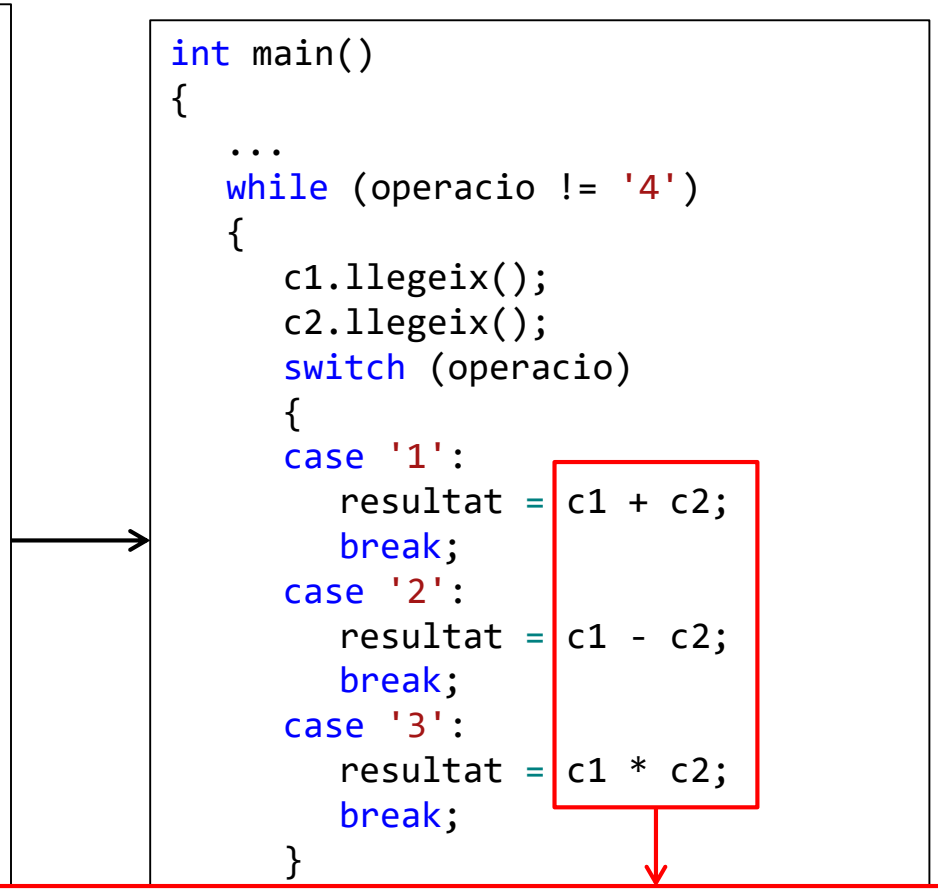
```
int main()
{
   Complex c1, c2, resultat;
   char operacio;
   cout << "Introdueix operacio (1. suma, 2. resta, 3. multiplica, 4. sortir): ";
   cin >> operacio;
   while (operacio != '4')
   {
      c1.llegeix();
      c2.llegeix();
      switch (operacio)
      {
      case '1':
         resultat = c1.suma(c2);
         break;
      case '2':
         resultat = c1.resta(c2);
         break;
      case '3':
         resultat = c1.multiplica(c2);
         break;
      }
      resultat.mostra();
      cout << endl << "Introdueix operació: ";
      cin >> operacio;
   }
}
```

```
class Complex
{
public:
   Complex() { m_real = 0; m_img = 0; }
   void llegeix();
   void mostra() const;
   Complex suma(const Complex &c) const;
   Complex resta(const Complex &c) const;
   Complex multiplica(const Complex &c) const;
private:
   float m_real, m_img;
};
```

Sobrecàrrega d'operadors

Fem un programa per operar amb números complexos:

```
int main()
{
   ...
   while (operacio != '4')
   {
      c1.llegeix();
      c2.llegeix();
      switch (operacio)
      {
      case '1':
         resultat = c1.suma(c2);
         break;
      case '2':
         resultat = c1.resta(c2);
         break;
      case '3':
         resultat = c1.multiplica(c2);
         break;
      }
      resultat.mostra();
      cin >> operacio;
   }
   ...
```



Sobrecàrrega d'operadors: redefinició dels operadors estàndar per poder-los utilitzar amb objectes de la classe utilitzant la sintaxi habitual dels operadors.

Sobrecàrrega d'operadors

```
class Complex
{
public:
   ...
   Complex suma(const Complex &c) const;
   Complex resta(const Complex &c) const;
   Complex multiplica(const Complex &c) const;
   ...
};
```

```
class Complex
{
public:
   ...
   Complex operator+(const Complex &c) const;
   Complex operator-(const Complex &c) const;
   Complex operator*(const Complex &c) const;
   ...
};
```

Sobrecàrrega d'operadors

```
Complex Complex::suma(const Complex &c) const
{
   Complex resultat;

   resultat.m_real = m_real + c.m_real;
   resultat.m_img = m_img + c.m_img;
   return resultat;
}
```

```
Complex c1, c2, resultat;

resultat = c1.suma(c2);
```

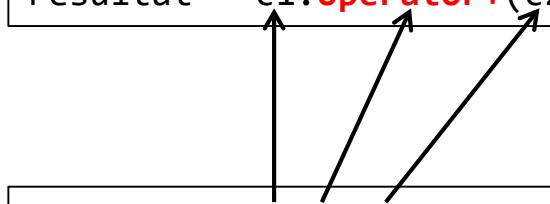


```
Complex Complex::operator+(const Complex &c) const
{
   Complex resultat;

   resultat.m_real = m_real + c. m_real;
   resultat.m_img = m_img + c. m_img;
   return resultat;
}
```

```
Complex c1, c2, resultat;

resultat = c1.operator+(c2);
```



```
resultat = c1 + c2;
```

Sobrecàrrega d'operadors

Alguns operadors que es poden sobrecarregar:

Aritmètics

+	+=	++
-	-=	--
*	*=	
/	/=	
%	%=	

Relacionals

>	<=
<	==
>=	!=

Lògics

&&
||
!

Entrada/Sortida

>>
<<

Exercici

Suposem que tenim aquesta classe `Data` per guardar dates:

```
class Data
{
public:
   Data() { m_dia = 0; m_mes = 0; m_any = 0; }
   Data(int dia, int mes, int any) : m_dia(dia), m_mes(mes), m_any(any) { }
   void setDia(int dia) { m_dia = dia; }
   void setMes(int mes) { m_mes = mes; }
   void setAny(int any) { m_any = any; }
   void llegeix();
   int getDia() const { return m_dia; }
   int getMes() const { return m_mes; }
   int getAny() const { return m_any; }
   bool dataValida() const;
private:
   int m_dia, m_mes, m_any;
   static const int N_MESOS = 12;
   const int nDiesMes[N_MESOS] = { 31,28,31,30,31,30,31,31,30,31,30,31 };
};
```

Exercici

1. Sobrecarregueu l'operador < per poder comparar dues dates i saber si una data és més petita que una altra.
2. Feu un programa principal que llegeixi per teclat dues dates i ens mostri per pantalla quina de les dues és més petita.

Exercici: solució

```
class Data
{
public:
   Data() { m_dia = 0; m_mes = 0; m_any = 0; }
   Data(int dia, int mes, int any) : m_dia(dia), m_mes(mes), m_any(any) { }
   void setDia(int dia) { m_dia = dia; }
   void setMes(int mes) { m_mes = mes; }
   void setAny(int any) { m_any = any; }
   void llegeix();
   int getDia() const { return m_dia; }
   int getMes() const { return m_mes; }
   int getAny() const { return m_any; }
   bool operator<(const Data& data) const;
private:
   int m_dia, m_mes, m_any;
   static const int N_MESOS = 12;
   const int nDiesMes[N_MESOS] = { 31,28,31,30,31,30,31,31,30,31,30,31 };
};
```

Exercici: solució

```
bool Data::operator<(const Data& data) const
{
   return ((m_any < data.m_any) ||
      ((m_any == data.m_any) && (m_mes < data.m_mes)) ||
      ((m_any == data.m_any) && (m_mes == data.m_mes) && (m_dia < data.m_dia)));
}
```

```
int main()
{
   Data data1;
   data1.llegeix();
   Data data2;
   data2.llegeix();
   if (data1 < data2)
      cout << "Data 1 mes petita" << endl;
   else
      cout << "Data 2 mes petita" << endl;
   return 0;
}
```

Sobrecàrrega d'operadors entrada/sortida

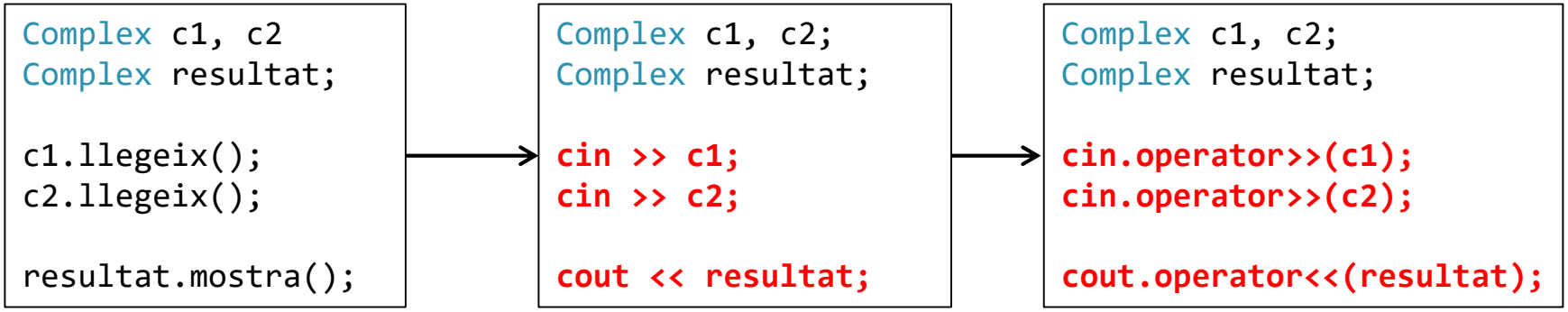
Fem un programa per operar amb números complexos:

```
int main()
{
   ...
   while (operacio != '4')
   {
      c1.llegeix();
      c2.llegeix();
      switch (operacio)
      {
      case '1':
         resultat = c1.suma(c2);
         break;
      case '2':
         resultat = c1.resta(c2);
         break;
      case '3':
         resultat = c1.multiplica(c2);
         break;
      }
      resultat.mostra();
      cout << endl;
   }
   ...
```

```
int main()
{
   ...
   while (operacio != '4')
   {
      cin >> c1;
      cin >> c2;
      switch (operacio)
      {
      case '1':
         resultat = c1 + c2;
         break;
      case '3':
         resultat = c1 * c2;
         break;
      }
      cout << resultat;
      cout << endl;
   }
   ...
```

Redefinició dels operadors d'entrada/sortida per poder llegir i escriure directament objectes de la classe `Complex`

Sobrecàrrega d'operadors entrada/sortida



- `cin` és un objecte de la classe `istream` i `cout` és un objecte de la classe `ostream` definits a la llibreria estàndard
- Hem de sobrecarregar l'operador `>>` de la classe `istream` i l'operador `<<` de la classe `ostream`
- Com que estan definides a la llibreria estàndard no podem modificar el codi d'aquestes classes per sobrecarregar aquests operadors.
  - Hem de fer la sobrecàrrega com a funcions globals

Sobrecàrrega d'operadors entrada/sortida

```
c1.llegeix();
```

```
void Complex::llegeix()
{
   cout << "Part real: ";
   cin >> m_real;
   cout << "Part imaginaria: ";
   cin >> m_img;
}
```

- Estem fora de l'àmbit de la classe Complex
- Si hem d'accedir a la part privada (atributs) ho hem de fer amb els getters i setters

Definició com a funció global, fora de l'àmbit de la classe Complex

```
cin >> c1;
```

```
operator>>(cin, c1)
```

```
istream& operator>>(istream& input, Complex& c)
{
   float real, img;

   cout << "Part real: ";
   input >> real;
   c.setReal(real);
   cout << "Part imaginaria: ";
   input >> img;
   c.setImg(img);

   return input;
}
```

Retornem l'objecte `istream`

Sobrecàrrega d'operadors entrada/sortida

```
resultat.mostra();
```

```
void Complex::mostra()
{
   cout << m_real << "+";
   cout << m_img << "i";
}
```

```
cout << resultat;
```

```
operator<<(cout, resultat)
```

```
ostream& operator<<(ostream& output, Complex& c)
{
   output << c.getReal() << "+" << c.getImg() << "i";

return output;

}
```

Retornem l'objecte ostream

- Estem fora de l'àmbit de la classe Complex
- Si hem d'accedir a la part privada (atributs) ho hem de fer amb els getters i setters

Exercici sessió 9 voluntari pujar nota

Utilitzant la classe `Data` de l'exercici anterior:

1. Sobrecarregueu l'operador == perquè comprovi si dues dates són iguals.
2. Sobrecarregueu l'operador + perquè retorni el resultat de sumar un nombre de dies determinat a una data.
3. Sobrecarregueu els operadors << i >> per poder llegir i escriure els valors de la data per pantalla. Per llegir una data s'ha de demanar per teclat el dia, el mes i l'any. Per escriure, s'ha de mostrar la data en format DD/MM/YYYY
4. Feu una funció que es digui `comprovaTermini` que, utilitzant els operadors anteriors, llegeixi per teclat una data inicial, la data actual i un nº de dies, i retorni un booleà que indiqui si la data actual introduïda és més petita o igual que el resultat de sumar el nº de dies introduït a la data inicial.

Exercici: solució

```
class Data
{
public:
   Data() { m_dia = 0; m_mes = 0; m_any = 0; }
   Data(int dia, int mes, int any) : m_dia(dia), m_mes(mes), m_any(any) { }
   void setDia(int dia) { m_dia = dia; }
   void setMes(int mes) { m_mes = mes; }
   void setAny(int any) { m_any = any; }
   void llegeix();
   int getDia() const { return m_dia; }
   int getMes() const { return m_mes; }
   int getAny() const { return m_any; }
   bool operator<(const Data& data) const;
   bool operator==(const Data& data) const;
   bool operator<(const Data& data) const;
private:
   int m_dia, m_mes, m_any;
   static const int N_MESOS = 12;
   const int nDiesMes[N_MESOS] = { 31,28,31,30,31,30,31,31,30,31,30,31 };
};

istream& operator>>(istream& input, Data& d);
ostream& operator<<(ostream& input, Data& d);
```

Exercici: solució

```
bool Data::operator==(const Data& data) const
{
   return ((m_dia == data.m_dia) && (m_mes == data.m_mes) && (m_any == data.m_any));
}
```

```
istream& operator>>(istream& input, Data& d)
{
   int dia, mes, any;

   cout << "Entra el dia: ";
   input >> dia;
   cout << "Entra el mes: ";
   input >> mes;
   cout << "Entra l'any: ";
   input >> any;
   d.setDia(dia);
   d.setMes(mes);
   d.setAny(any);
   return input;
}
```

```
ostream& operator<<(ostream& output, Data& d)
{
   output << d.getDia() << "/" << d.getMes() << "/" << d.getAny();
   return output;
}
```

Exercici: solució

```
Data Data::operator+(int nDies) const
{
   Data dataNova(m_dia, m_mes, m_any);
   while (nDies > 0)
   {
      int diaAux = dataNova.m_dia + nDies;
      if (diaAux > nDiesMes[dataNova.m_mes-1])
      {
         nDies -= (nDiesMes[dataNova.m_mes-1] - dataNova.m_dia + 1);
         dataNova.m_dia = 1;
         dataNova.m_mes++;
         if (dataNova.m_mes > N_MESOS)
         {
            dataNova.m_any++;
            dataNova.m_mes = 1;
         }
      }
      else
      {
         dataNova.m_dia = diaAux;
         nDies = 0;
      }
   }
   return dataNova;
}
```

Exercici: solució

```
bool comprovaTermini()
{
   Data diaOriginal, dataActual;
   int nDies;

   cin >> diaOriginal;
   cin >> dataActual;
   cout << "Introdueix n. dies: ";
   cin >> nDies;
   bool valid = false;
   Data diaTermini = diaOriginal + nDies;
   if ((dataActual < diaTermini) || (diaTermini == dataActual))
      valid = true;
   return valid;
}
```