Objectes dinàmics
Tema 2: Estructures de dades dinàmiques

Memòria dinàmica

```
Complex *llegeixComplex()
{
   Complex *c = new Complex;
   float real, img;

   cout << "Introdueix part real: ";
   cin >> real;
   c->setReal(real);
   cout << "Introdueix part imaginaria: ";
   cin >> img;
   c->setImg(img);
   return c;
}

int main()
{
   Complex *c1 = llegeixComplex();
   Complex *c2 = llegeixComplex();
   cout << "Suma: " << *c1 + *c2 << endl;

   delete c1;
   delete c2;
   return 0;
}
```

- Què fa aquest programa?

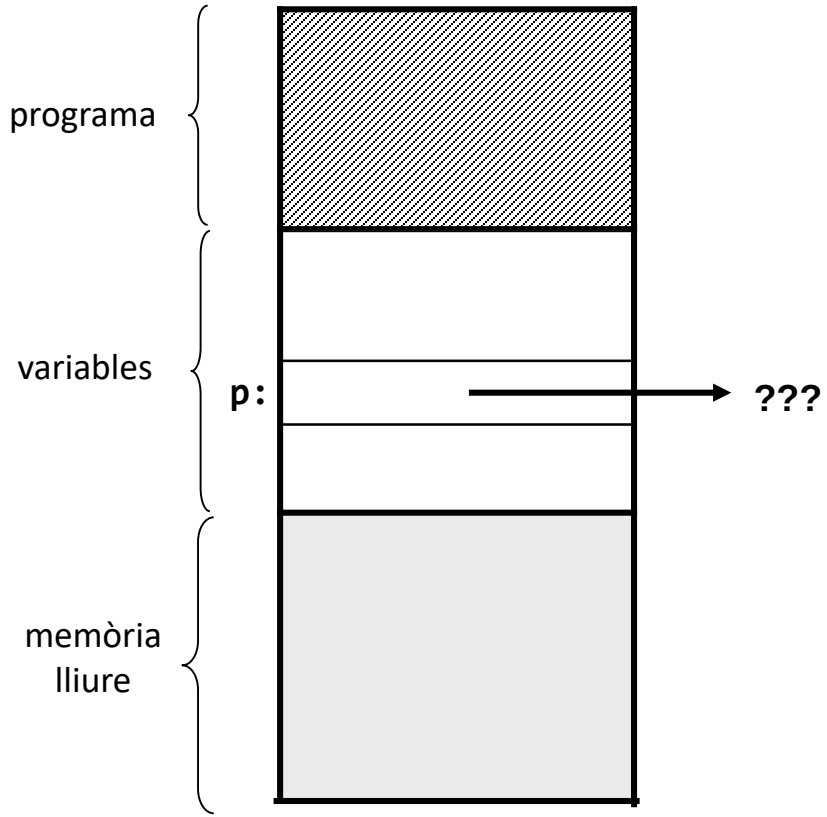
Creació d'objectes dinàmics

Destrucció d'objectes dinàmics

Memòria dinàmica

Creació d'objectes dinàmics: l'operador `new`

`<tipus>* p`: declaració d'un apuntador de tipus **`<tipus>`**

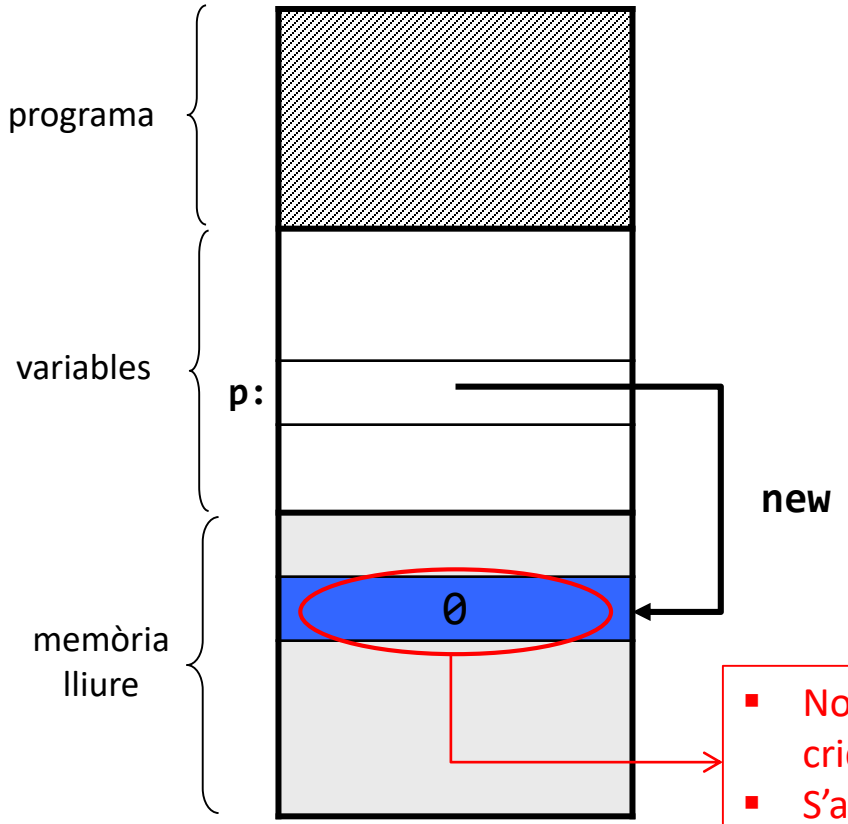



```
int* p;
```

Memòria dinàmica

Creació d'objectes dinàmics: l'operador new

`p = new <tipus>:` creació d'un nou objecte (dinàmic) del **tipus** apuntat per p



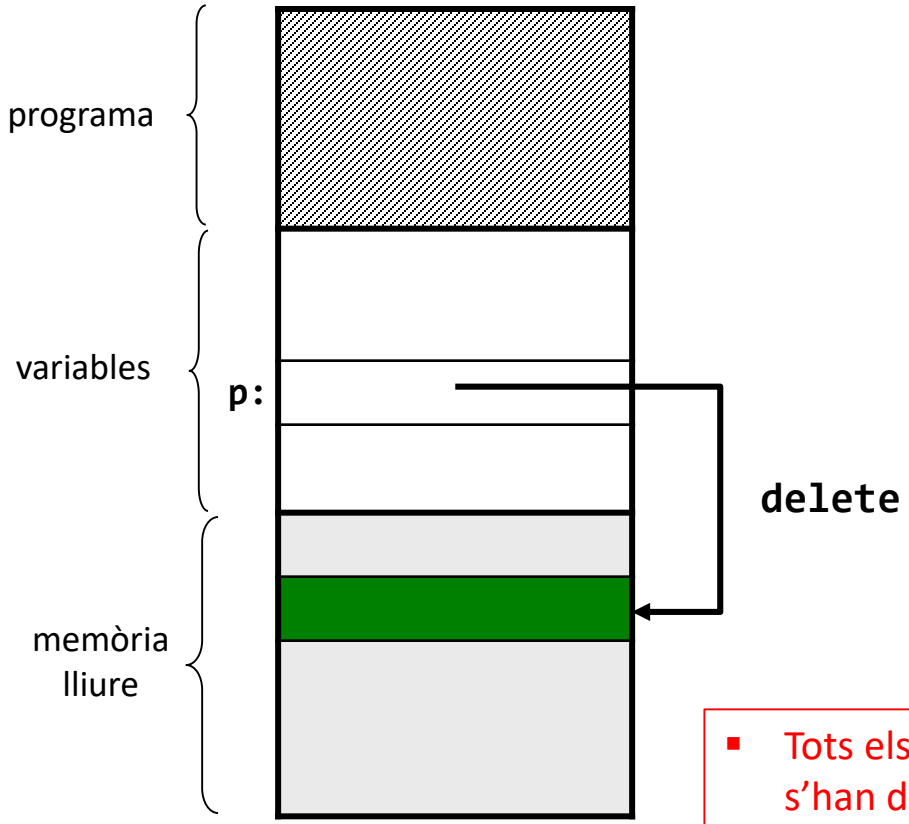

```
int* p;

p = new int;

*p = 0;
```

- Nova variable dinàmica (es crea quan es crida a l'operador new) de tipus `int`
- S'assigna un espai de memòria lliure suficient per guardar un valor del tipus de l'apuntador
- Només accessible amb l'apuntador p

Memòria dinàmica

Destrucció d'objectes dinàmics: l'operador `delete`

`delete` p: destrucció de l'objecte dinàmic apuntat per p



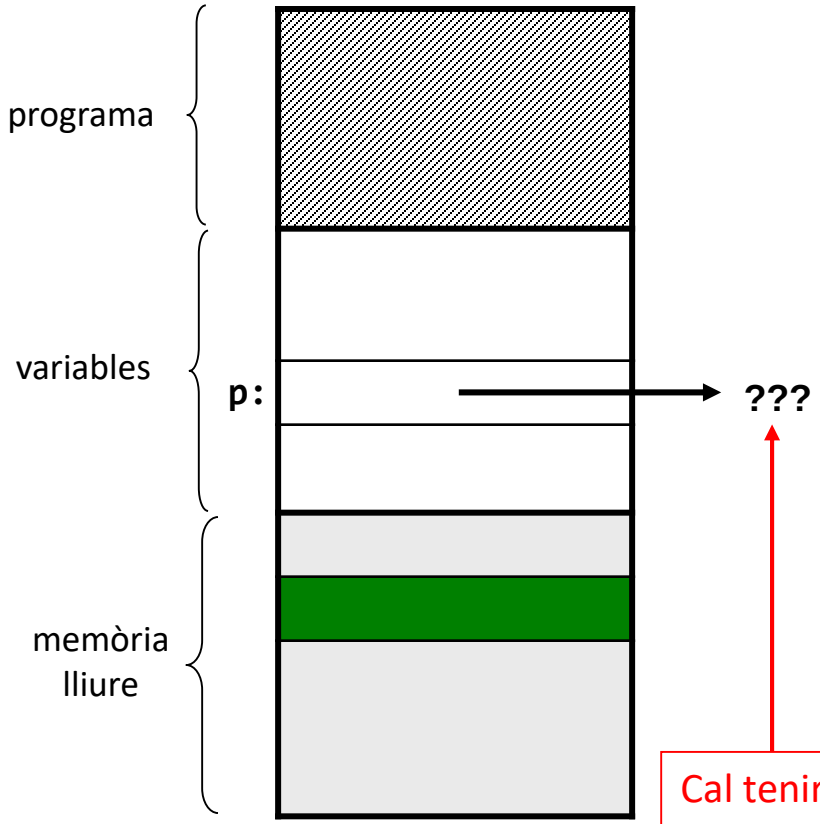

```
int* p;

p = new int;

*p = 0;

...

delete p;
```

- Tots els objectes dinàmics que es creen s'han d'alliberar (operador `delete`) quan ja no es necessiten
- Retorna l'espai de memòria ocupat per la variable dinàmica a la memòria lliure

Memòria dinàmica

Destrucció d'objectes dinàmics: l'operador `delete`

`delete` p: destrucció de l'objecte dinàmic apuntat per p



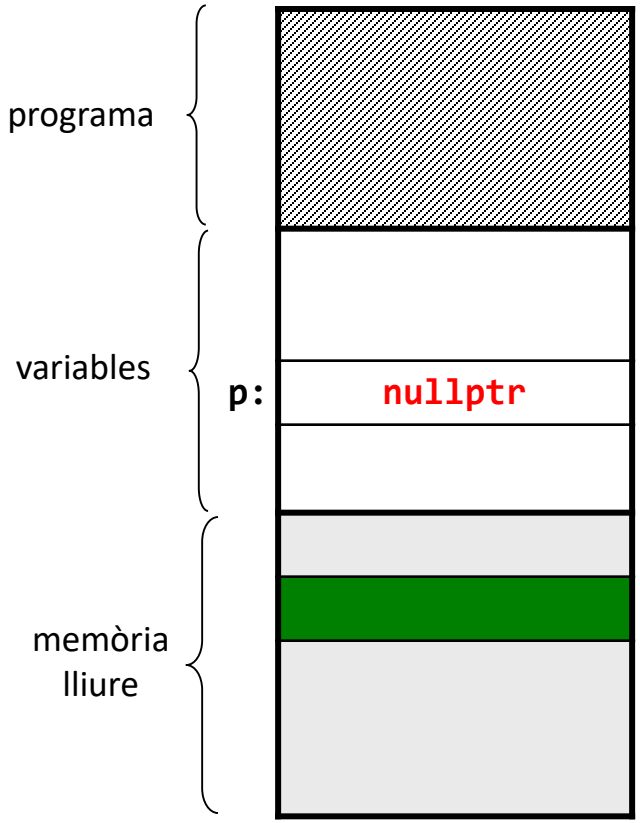

```
int* p;

p = new int;

*p = 0;

...

delete p;
```

Cal tenir en compte que després d'alliberar un apuntador, el seu valor és INDEFINIT

Memòria dinàmica

Destrucció d'objectes dinàmics: l'operador `delete`

`delete` p: destrucció de l'objecte dinàmic apuntat per p




```
int* p;

p = new int;

*p = 0;

...

delete p;
p = nullptr;
```

IMPORTANT: El valor `null` és la manera correcta de determinar que un apuntador no apunta a res

Memòria dinàmica

```
Complex *llegeixComplex()
{
   Complex *c = new Complex;
   float real, img;

   cout << "Introdueix part real: ";
   cin >> real;
   c->setReal(real);
   cout << "Introdueix part imaginaria: ";
   cin >> img;
   c->setImg(img);
   return c;
}

int main()
{
   Complex *c1 = llegeixComplex();
   Complex *c2 = llegeixComplex();
   cout << "Suma: " << *c1 + *c2 << endl;

   delete c1;
   delete c2;
   return 0;
}
```

- Què fa aquest programa?

Objectes dinàmics, constructors i destructors

```
class Complex
{
public:
   Complex() {};
   Complex(float real, float img) : m_real(real), m_img(img) {}
   ~Complex() {};
   ...
};
```

Creació de l'objecte

```
Complex *llegeixComplex()
{
   Complex *c = new Complex;
   ...
```

crida al constructor per defecte

```
Complex *llegeixComplex()
{
   ...
   cin >> real;
   cin >> img;
   Complex *c = new Complex(real, img);
   return c;
}
```

crida al constructor amb paràmetres

Destrucció de l'objecte

```
int main()
{
   Complex *c1 = llegeixComplex();
   ...
   delete c1;
}
```

crida al destructor

Exercici

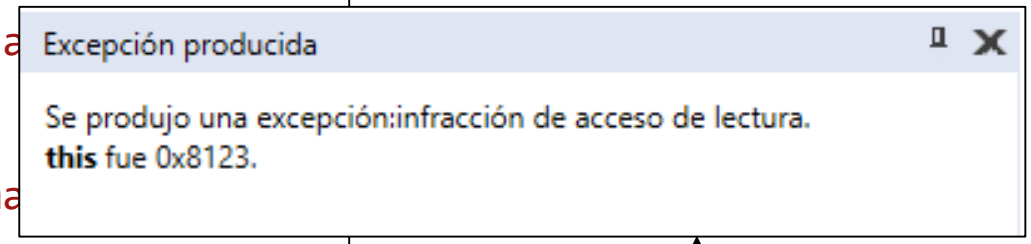
- Quina instrucció és incorrecta en el codi de la funció `llegeixComplex`?

```
Complex *llegeixComplex()
{
   Complex *c = new Complex;
   float real, img;

   cout << "Introdueix part real: ";
   cin >> real;
   c->setReal(real);
   cout << "Introdueix part imaginaria: ";
   cin >> img;
   c->setImg(img);
   delete c;
   return c;
}
```

```
int main()
{
   Complex *c1 = llegeixComplex();
   Complex *c2 = llegeixComplex();
   cout << "Suma: " << *c1 + *c2 << endl;
   delete c1;
   delete c2;

   return 0;
}
```

Exercici: solució

```
Complex *llegeixComplex()
{
   Complex *c = new Complex;
   float real, img;

   cout << "Introdueix part rea
   cin >> real;
   c->setReal(real);
   cout << "Introdueix part ima
   cin >> img;
   c->setImg(img);
   delete c;
   return c;
}
```




```
int main()
{
   Complex *c1 = llegeixComplex();
   Complex *c2 = llegeixComplex();
   cout << "Suma: " << *c1 + *c2 << endl;
   ...
```

- No podem alliberar variables dinàmiques que encara s'han d'utilizar (aquesta variable s'utilitza al `main`)
- Les variables dinàmiques s'han d'alliberar quan ja no es faran servir més, encara que sigui en una funció diferent a on s'han creat.

Exercici

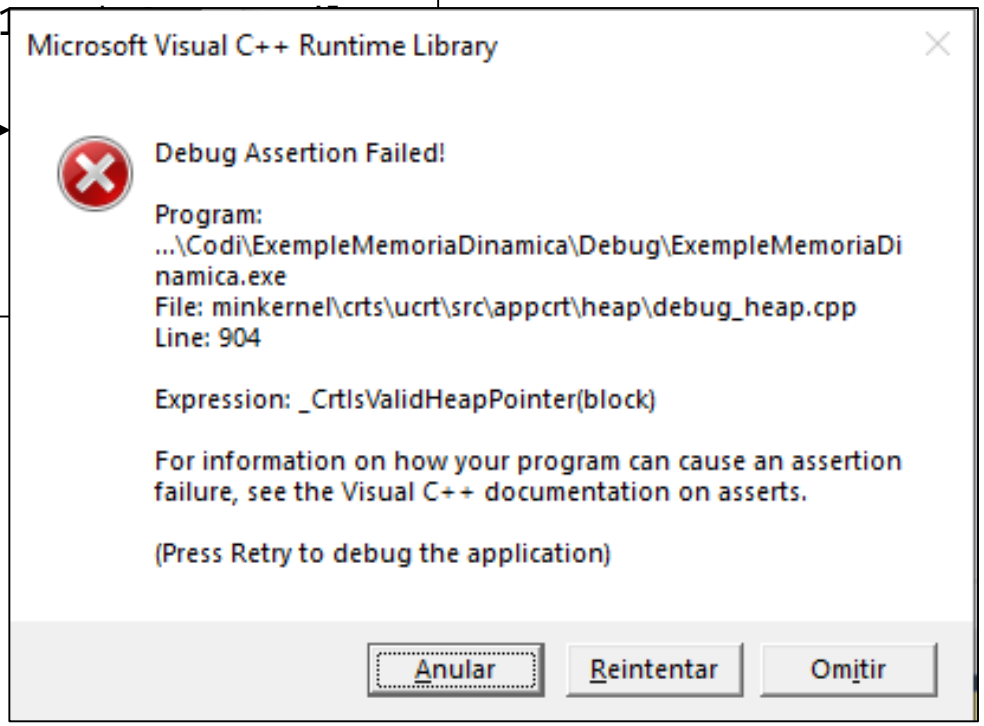
- Quina instrucció és incorrecta en el codi del `main`?

```
Complex *llegeixComplex()
{
   Complex *c = new Complex;
   float real, img;

   cout << "Introdueix part real: ";
   cin >> real;
   c->setReal(real);
   cout << "Introdueix part imaginaria: ";
   cin >> img;
   c->setImg(img);

   return c;
}
```

```
int main()
{
   Complex *c1 = llegeixComplex();
   Complex c2 = Complex(1, 1);
   cout << "Suma: " << *c1 + *c2 << endl;
   delete c1;
   delete c2;

   return 0;
}
```

Exercici: solució

```
int main()
{
   Complex *c1 = llegeixComplex();
   Complex c2 = Complex(1, 1);
   cout << "Suma: " << *c1
   delete c1;
   delete c2;

   return 0;
}
```



- No podem alliberar apuntadors que apuntin a variables estàtiques

Exercici

- Quines instruccions falten en el codi del `main`?

```
Complex *llegeixComplex()
{
   Complex *c = new Complex;
   float real, img;

   cout << "Introdueix part real: ";
   cin >> real;
   c->setReal(real);
   cout << "Introdueix part im
   cin >> img;
   c->setImg(img);

   return c;
}
```

```
int main()
{
   Complex *c1 = llegeixComplex();
   Complex *c2 = llegeixComplex();
   cout << "Suma: " << *c1 + *c2 << endl;

   c1 = llegeixComplex();
   c2 = llegeixComplex();
   cout << "Resta: " << *c1 - *c2 << endl;

   delete c1;
   delete c2;
   return 0;
}
```

Exercici: solució

```
int main()
{
   Complex *c1 = llegeixComplex();
   Complex *c2 = llegeixComplex();
   cout << "Suma: " << *c1 + *c2 << endl;

   c1 = llegeixComplex();
   c2 = llegeixComplex();
   cout << "Resta: " << *c1 - *c2 << endl;

   delete c1;
   delete c2;
   return 0;
}
```

- Només alliberem les variables dinàmiques que s'han llegit per fer la resta.
- Què passa amb les variables dinàmiques que s'han llegit per fer la suma?

Exercici: solució

```
int main()
{
   Complex *c1 = llegeixComplex();
   Complex *c2 = llegeixComplex();
   cout << "Suma: " << *c1 + *c2 << endl;
   delete c1;
   delete c2;

   c1 = llegeixComplex();
   c2 = llegeixComplex();
   cout << "Resta: " << *c1 - *c2 << endl;

   delete c1;
   delete c2;
   return 0;
}
```

Exercici

Volem crear un conjunt de classes que ens permetin fer una gestió simplificada de les targetes d'embarcament d'un vol d'avió. Per guardar les dades (dni i nom) dels passatgers que viatgen a l'avió tenim ja creada una classe `Passatger`, amb aquesta declaració:

```
class Passatger
{
public:
   Passatger() {}
   Passatger(const string& dni, const string& nom): m_dni(dni), m_nom(nom) {}
   void setDni(const string& dni) { m_dni = dni; }
   void setNom(const string& nom) { m_nom = nom; }
   const string& getDni() const { return m_dni; }
   const string& getNom() const { return m_nom; }
private:
   string m_dni;
   string m_nom;
};
```

Exercici

- Tenim també creada una classe `Seient` que permet guardar les dades de cadascun dels seients que té l'avió. Per cada seient guardem un codi que l'identifica i les dades del passatger que ha reservat aquest seient.
- Les dades del passatger les guardem amb un apuntador a un objecte dinàmic de la classe `Passatger`.
- Si l'apuntador val `nullptr` vol dir que el seient està lliure. Si té un valor diferent de `nullptr`, el contingut de l'apuntador és un objecte dinàmic amb les dades del passatger que ha reservat aquest seient.

```
class Seient
{
public:
   Seient();
   Seient(const string& codi);
   ~Seient();
   void setCodi(const string& codi) { m_codi = codi; }
   void setPassatger(Passatger* p) { m_passatger = p; }
   bool assignaPassatger(const string& dni, const string& nom);
   bool eliminaPassatger();
   const string& getCodi () const { return m_codi; }
   Passatger* getPassatger() { return m_passatger; }
private:
   string m_codi;
   Passatger* m_passatger;
};
```

Exercici

En aquesta classe heu d'implementar els mètodes següents:

- Els dos constructors de la classe, tenint en compte que l'apuntador al passatger ha de quedar correctament inicialitzat a `nullptr`.

- El destructor, tenint en compte d'alliberar, si està creat, l'objecte dinàmic de la classe `Passatger`.

- El mètode `assignaPassatger`, que assigna al seient el passatger amb les dades que es passen com a paràmetre. Heu de tenir en compte de crear correctament l'objecte dinàmic de la classe `Passatger` per poder-hi guardar les dades. Si el seient ja estava reservat a un altre passatger no s'ha de fer res i s'ha de retornar false.

- El mètode `eliminaPassatger`, que elimina el passatger que tenia reservat aquest seient. Heu de tenir en compte d'alliberar correctament l'objecte dinàmic de la classe `Passatger` que guardava les dades del passatger. Si el seient no estava reservat no s'ha de fer res i s'ha de retornar false.

Exercici

Finalment, per guardar totes les dades d'un vol, amb tots els seus seients i passatgers, hem fet la declaració de la classe `Vol`:

```
class Vol
{
public:
   Vol() { m_nSeients = 0; }
   Vol(const string& codi, const string& origen, const string& desti,
     const string& data, const string& hora) :
     m_codi(codi), m_origen(origen), m_desti(desti), m_data(data), m_hora(hora)
         { m_nSeients = 0; }
   void afegeixSeients(string* codiSeients, int nSeients);
   bool afegeixPassatger(const string& codiSeient, const string& dni, const string& nom);
   Passatger* recuperaPassatger(const string& codiSeient);
   bool cancelaReserva(const string& codiSeient);
   bool modificaReserva(const string& codiSeientOriginal, const string& codiSeientNou);
private:
   static const int MAX_SEIENTS = 100;
   string m_codi;
   string m_origen;
   string m_desti;
   string m_data;
   string m_hora;
   Seient m_seients[MAX_SEIENTS];
   int m_nSeients;
};
```

Exercici

Apart de guardar les dades bàsiques del vol (origen, destí, dia i hora), aquesta classe guarda un array amb les dades de tots els seients disponibles a l'avió. Cada seient es guarda utilitzant un objecte de la classe `Seient` que hem descrit anteriorment. En aquesta classe heu d'implementar els mètodes següents:

- El mètode `afegeixPassatger`, que assigna el passatger amb les dades (dni i nom) que es passen com a paràmetre al seient amb el codi que també es passa com a paràmetre. Aquest mètode haurà d'utilitzar el mètode `assignaPassatger` de la classe `Seient`. Si el codi de seient no existeix dins de l'array o el seient ja estava reservat a un altre passatger, no s'ha de fer res i retorna false.
- El mètode `cancelaReserva`, que elimina el passatger del seient amb codi que es passa com a paràmetre. Aquest mètode haurà d'utilitzar el mètode `eliminaPassatger` de la classe `Seient`. Si el codi de seient no existeix dins d el'array o el seient no tenia cap reserva feta, no s'ha de fer res i retorna false.
- El mètode `modificaReserva`, que canvia la reserva de seient que havia fet anteriorment un passatger. El mètode rep com a paràmetres el codi del seient de la reserva inicial que es vol anul·lar i el codi del seient on es vol fer la nova reserva. S'ha d'eliminar la reserva del seient inicial i passar-la al nou seient. Si algun dels dos codis de seient no existeixen dins de l'array o bé el seient inicial no estava reservat o bé el seient nou ja està reservat no es pot fer el canvi i s'ha de retornar false.
- El mètode `recuperaPassatger`, que retorna un apuntador a les dades del passatger que té reservat el seient amb el codi que es passa com a paràmetre. Si el codi de seient no existeix dins de l'array o el seient no està reservat es retorna `nullptr`.