MP 20-21 Tema 1
Constructors i destructors
Sessió 7: Programació Orientada a Objectes

Inicialització/destrucció objectes: constructors/destructors

```
Punt p1;
p1.setX(0.0);
p1.setY(0.0);
```

- Quin valor té p1 just després de la declaració?
- És important inicialitzar correctament totes les variables (per tant, també els objectes) abans d'utilitzar-los
- Podem inicialitzar p1 sense necessitat de cridar explícitament als `setters` de la classe?

Constructors i destructors

```
class Punt
{
public:
    Punt();
    ~Punt();
...
```

Constructor:

- Mètode que existeix sempre per defecte a qualsevol classe
- Es defineix sense valor de retorn
- Es crida per defecte sempre que es crea un objecte (variable) de la classe
- Es pot utilitzar per inicialitzar la classe (valors dels atributs), per crear memòria dinàmica quan calgui (tema 2), …

Destructor:

- Mètode que existeix sempre per defecte a qualsevol classe
- Es defineix sense valor de retorn
- Es crida per defecte sempre que es destrueix un objecte (variable) de la classe
- Serveix per fer qualsevol tractament final de l'objecte, per allliberar memòria dinàmica quan calgui (tema 2), …

Inicialització/destrucció objectes: constructors/destructors

```
class Punt
{
public:
    Punt();
    ~Punt();
...
```

```
Punt::Punt()
{
   m_x = 0;
   m_y = 0;
}
```

Constructor de la classe Punt : inicialitza les coordenades a 0

```
Punt::~Punt()
{
}
```

Destructor de la classe Punt : no ha de fer res

```
int main ()
{
   Punt p1;

    ...

}
```

Creació de l'objecte
Crida al constructor

Destrucció de l'objecte
Crida al destructor

p1

m_x: 0
m_y: 0

- Ja no cal inicialitzar l'objecte explícitament amb els setters
- Queda inicialitzat de forma automàtica cridant al constructor

Inicialització/destrucció objectes: constructors/destructors

Alternatives per implementar constructors i destructors

1. Declaració al fitxer `.h` i implementació al fitxer `.cpp`

```
class Punt
{
public:
    Punt();
    ~Punt();
```

```
Punt::Punt()
{
   m_x = 0;
   m_y = 0;
}
```

```
Punt::~Punt()
{
}
```

2. Implementació *inline*

```
class Punt
{
public:
   Punt() { m_x = 0; m_y = 0; }
   ~Punt() { }
```

3. Inicialització directa dels atributs

```
class Punt
{
public:
   Punt(): m_x(0), m_y(0) {  }
   ~Punt() { }
```

Sobrecàrrega constructors: constructors amb paràmetres

```
Punt p1;
p1.setX(2.0);
p1.setY(3.0);
```

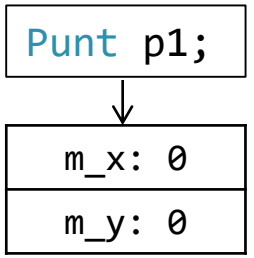
- Crida al constructor per defecte. Inicialització sempre amb els mateixos valors
- Podem inicialitzar p1 a un valor diferent sense necessitat de cridar als setters?

Constructors amb paràmetres

```
class Punt
{
public:
   Punt(): m_x(0), m_y(0) {  }
   Punt(float x, float y) : m_x(x), m_y(y) {}
```

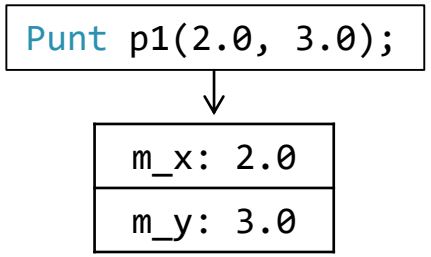
Constructor per defecte:

- Inicialització sempre igual a valors per defecte



Constructor amb paràmetres:

- Inicialització amb els valors indicats com a paràmetres quan es crea l'objecte
- Sobrecàrrega del constructor
  - Sobrecàrrega: redefinició de la mateixa funció canviant els paràmetres i/o el valor de retorn
  - Sabem quina versió de la funció s'ha d'utilitzar pels paràmetres indicats en el moment de fer la crida



Ja no cal cridar als `setters`

Exercici voluntari pujar nota

Afegiu a la classe `Recta` els constructors següents:

- Un constructor per defecte que inicialitzi els paràmetres a la recta amb equació $x = 0$.
- Un constructor per inicialitzar els paràmetres de la recta amb els valors que es passen com a paràmetre:

```
Recta(float a, float b, float c);
```

- Un constructor per inicialitzar els paràmetres de la recta de forma que passi pels dos punts que es passen com a paràmetre:

```
Recta(Punt& pt1, Punt& pt2);
```

Suposant que (x_1, y_1) i (x_2, y_2) són els dos punts que es passen com a paràmetres, la recta (si no és vertical) queda definida per aquestes expressions:

$$A = \frac{y_2 - y_1}{x_2 - x_1}$$

$$B = -1.0$$

$$C = -Ax_1 + y_1$$

Si la recta és vertical $A = 1.0, B = 0$ i $C = -x_1$.

Exercici voluntari pujar nota

Utilitzeu la classe `Recta` per implementar les funcions següents:

- Una funció que permeti calcular totes les distàncies d'una recta a un conjunt de punts, amb la capçalera següent:

```
void distancia(Punt& pt1, Punt& pt2, Punt llistaPunts[],
               int nPunts, float distancies[]);
```

Primer de tot la funció ha de crear la recta que passa pels dos primers punts que es passen com a paràmetres. Després ha de calcular la distància de tots els punts de l'array `llistaPunts` a la recta que s'acaba de crear i guardar totes les distàncies a l'array `distancies` que es passa com a paràmetre. `nPunts` és el nº de punts total de l'array `llistaPunts`.

Exercici voluntari pujar nota

- Una funció que permeti calcular la intersecció d'una recta a un conjunt de rectes, amb la capçalera següent:

```
void interseccio(Recta& r, Punt llistaPunts1[], Punt llistaPunts2[],
                 int nPunts, bool interseccio[], Punt ptInterseccio[]);
```

La funció ha de trobar el punt d'intersecció entre la recta que es passa com a paràmetre i totes les rectes definides pels punts que es passen als arrays `llistaPunts1` i `llistaPunts2`. Cada recta està definida per un punt de `llistaPunts1` i el punt corresponent de `llistaPunts2`. `nPunts` és el nº de punts total de l'array `llistaPunts`. A l'array `interseccio` s'ha de retornar si existeix o no intersecció per cadascuna de les rectes. A l'array `ptInterseccio` s'ha de retornar el punt d'intersecció per cadascuna de les rectes, si existeix, i un valor indefinit si no existeix.

Exercici: solució

```
class Recta
{
public:
   Recta() { m_a = 1; m_b = 0; m_c = 0; }
   Recta(float a, float b, float c) : m_a(a), m_b(b), m_c(c) {}



Recta::Recta(const Punt& pt1, const Punt& pt2)
{
   if (pt2.getX() != pt1.getX())
   {
      m_a = (pt2.getY() - pt1.getY()) / (pt2.getX() - pt1.getX());
      m_b = -1.0;
      m_c = -m_a * pt1.getX() + pt1.getY();
   }
   else
   {
      m_a = 1.0;
      m_b = 0.0;
      m_c = -pt1.getX();
   }
}
```

Exercici: solució

```
void distancia(Punt& pt1, Punt& pt2, Punt llistaPunts[], int nPunts,
                  float distancies[])
{
   Recta r(pt1, pt2);
   for (int i = 0; i < nPunts; i++)
      distancies[i] = r.distancia(llistaPunts[i]);
}

void interseccio(Recta& r, Punt llistaPunts1[], Punt llistaPunts2[], int nPunts,
                  bool interseccio[], Punt ptInterseccio[])
{
   for (int i = 0; i < nPunts; i++)
   {
      Recta r2(llistaPunts1[i], llistaPunts2[i]);
      interseccio[i] = r.interseccio(r2, ptInterseccio[i]);
   }
}
```