# Examen Parcial Programació 2 (24 Març 2015)

1. Crea una classe "*Projectile*" que fent servir la classe Punt2D i coneixent una velocitat pugui calcular la posició del punt donat un temps **t** (2 punts)
2. Afegeix un mètode "*removeWastedMemory"* al Array Dinàmic que elimini tota la memòria que no es faci servir. Ha de retornar els espais de memòria alliberades. (2 punts)
3. Afegeix un mètode *"delNodes"* a la classe de llista doblement enllaçada per poder esborrar qualsevol nombre de nodes des d'una posició qualsevol. Ha de retornar la quantitat de nodes eliminats. (3 punts)
4. Afegeix dos mètodes "*prefix*" a la classe cadena per poder afegir un altre classe cadena o cadena de c al principi. Ha de retornar a si mateixa. (3 punts)

Criteris de correcció:
- Si no compila o el unit test no passa el exercici no es corregira
- Minimitzar crides als new i delete
- Codi *const-correct*
- Codi clar i ben organitzat

## Unit Test a passar (feu copiar i enganxar)

```
// Projectile ----------------------------------------
TEST_METHOD(Projectile_test)
{
	Projectile p;

	p.point.x = 10.0f;
	p.point.y = 10.0f;

	p.speed.x = 2.0f;
	p.speed.y = 0.0f;

	p2Point<float> current = p.GetCurrentPosition(3.0f);

	Assert::AreEqual((float)16.0f, current.x, 0.00001f);
	Assert::AreEqual((float)10.0f, current.y, 0.00001f);
}

// ArrDyn remove wasted memory ----------------------------------------
TEST_METHOD(ArrDyn_optimizeMem)
{
	p2DynArray<int> array(10);

	array.PushBack(1);
	array.PushBack(2);
	array.PushBack(3);

	Assert::AreEqual((unsigned int)10, array.GetCapacity());

	unsigned int wasted = array.removeWastedMemory();
```

```
	Assert::AreEqual((unsigned int)3, array.GetCapacity());
	Assert::AreEqual((unsigned int)7, wasted);
	Assert::AreEqual((int)1, array[0]);
	Assert::AreEqual((int)2, array[1]);
	Assert::AreEqual((int)3, array[2]);
}

// P2List delete few nodes ----------------------------------------
TEST_METHOD(p2List_delNodes_mid)
{
	p2List<int> l;

	l.add(1);
	l.add(2);
	l.add(3);
	l.add(4);

	l.delNodes(1, 2);

	Assert::AreEqual((int)1, l.start->data);
	Assert::AreEqual((int)4, l.end->data);
	Assert::AreEqual((unsigned int)2, l.count());
}

// P2List delete few nodes ----------------------------------------
TEST_METHOD(p2List_delNodes_begin)
{
	p2List<int> l;

	l.add(1);
	l.add(2);
	l.add(3);
	l.add(4);

	l.delNodes(0, 3);

	Assert::AreEqual((int)4, l.start->data);
	Assert::AreEqual((int)4, l.end->data);
	Assert::AreEqual((unsigned int)1, l.count());
}

// P2List delete few nodes ----------------------------------------
TEST_METHOD(p2List_delNodes_end)
{
	p2List<int> l;

	l.add(1);
	l.add(2);
	l.add(3);
	l.add(4);

	l.delNodes(2, 100);

	Assert::AreEqual((int)1, l.start->data);
	Assert::AreEqual((int)2, l.end->data);
	Assert::AreEqual((unsigned int)2, l.count());
```

```
}

// String prefix ---------------------------------------
TEST_METHOD(String_prefix)
{
	p2SString a("1234567890");
	p2SString b(50);
	b = "hola";

	a.prefix(b);
	b.prefix("1234567890");

	Assert::AreEqual(strcmp(a.GetString(), "hola1234567890"), 0);
	Assert::AreEqual(strcmp(b.GetString(), "1234567890hola"), 0);
}
```