Seminarios de C# (Primera Parte)

# Problema 1 - Clausura

Ejecute el siguiente código:

var actions = new Action[10];

for (int x = 0; x < actions.Length; x++)

{

int y = x;

actions[x] = () =>

{

int z = x;

Console.WriteLine("{0}, {1}, {2}\n", x, y, z);

};

}

foreach (var action in actions)

{

action();

}

¿Por qué el resultado que sale en pantalla no es el esperado? Explicar el concepto de **clausura** (*closure*) y la forma (antinatural) en que C# captura las variables en la clausura.

Apoyar la explicación con el código IL generado (use Reflector). Explique por qué el siguiente código lanza excepción:

var actions = new List<Action>();

string[] urls =

{

"http://www.url.com",

"http://www.someurl.com",

"http://www.someotherurl.com",

"http://www.yetanotherurl.com"

};

for (int i = 0; i < urls.Length; i++)

{

actions.Add(() => Console.WriteLine(urls[i]));

}

# Problema 2 - LINQ

Brinde una implementación eficiente y simple del siguiente método de extensión y analice el costo operacional para el caso peor:

**public** **static** IEnumerable<[IGrouping](ms-help://MS.MSDNQTR.v90.en/fxref_system.core/html/bba4962d-c071-c465-5c4e-75754a7ad9bb.htm)<TKey, TSource>> GroupBy<TSource, TKey>(

**this** IEnumerable<TSource> *source*,

[Func](ms-help://MS.MSDNQTR.v90.en/fxref_system.core/html/a0ab867a-da51-dd82-2e1a-e87b93712102.htm)<TSource, TKey> *keySelector*

)

Una aplicación útil de este método de extensión sería:

var estudiantes = new List<Estudiante>();

// ...Algún código de inicialización...

var Grupos = estudiantes.GroupBy(estudiante => estudiante.Grupo);

1. ¿Se explotaría en su totalidad una implementación “*Lazy*” del GroupBy? ¿o el costo de las operaciones para el caso peor es el mismo independientemente de si se hace un Take(k)?
2. Rescriba el siguiente código de forma tal que siga manteniendo el while(true) pero que permita “parar” la ejecución del método para un momento dado:

static List<int> GetPrimes()

{

var primes = new List<int>();

int i = 1;

while (true)

{

if (IsPrime(i)) primes.Add(i);

i++;

}

return primes;

}

1. ¿Por qué la siguiente sentencia no bloquea el programa?

GetPrimes().Where(prime => prime.ToString().StartsWith("2")).Take(10);

1. Convierta el siguiente código Haskell a C#:

Four :: Integer -> Integer  
Four x = 4

Infinity :: Integer  
Infinity = 1 + Infinity

1. ¿Cuál es el resultado de evaluar *Infinity* en *Four*?
2. ¿Son equivalentes los siguientes códigos?

|  |  |
| --- | --- |
| if (Cond1() || Cond2())  {  Console.WriteLine(true);  }  else  {  Console.WriteLine(false);  } | if (Cond1() | Cond2())  {  Console.WriteLine(true);  }  else  {  Console.WriteLine(false);  } |

**Objetivos:** LINQ, iteradores *lazy*,

# Problema 3 - Varianza, Covarianza, Herencia y Polimorfismo

En la Universidad, una persona (que se identifica por su Nombre) puede representar diferentes roles:

* + Estudiante (Acción: RecibirClase())
  + Profesor (Acción: ImpartirClase())
  + Alumno Ayudante (Estudiante que no es profesor pero actúa como tal en un momento dado, es decir, puede realizar ImpartirClase())
  + Trabajador (no todo trabajador es profesor, pero sí todos los profesores son trabajadores. Acción: CobrarSalario())

1. Diseñe una jerarquía en C# que represente/modele los roles anteriores y sus relaciones. Utilice alguna herramienta para ilustrar dicho modelo (Ejemplo: diseñador de clases de Visual Studio)
2. ¿Es posible utilizar el siguiente código para imprimir una lista genérica de profesores? Haga los arreglos que crea necesario para que ejecute en caso de que su respuesta sea NO. Explique el funcionamiento de las características del lenguaje utilizadas.

void PrintPeople(IEnumerable<Person> people) {

for(var p in people)

Console.WriteLine(p.Name);

}

1. En la secretaría de la Facultad, usualmente se imprimen listados de estudiantes dado algún criterio (por nombre, por nota, etc). El algoritmo es el siguiente:

static void PrintStudents(IEnumerable<Student> students,

IComparer<Student> comparer) {

foreach (var student in students.OrderBy(x => x, comparer))

Console.WriteLine(student.Name);

}

Implemente un comparador que permita utilizar el código anterior para imprimir los estudiantes ordenados por nombre, pero que dicho comparador se pueda reutilizar luego para profesores, trabajadores y alumnos ayudantes. Explique las características del lenguaje utilizadas.

1. Explique e Ilustre el funcionamiento del siguiente código:

static void PrintByConsole(Action<Action<Person>> person) {

person(x => Console.WriteLine(x.Name));

}

...

PrintByConsole(x=>new Student() {Name = "Pedro"});

1. El siguiente código recibe una colección de personas que pueden ejercer cualquier rol pero se quieren imprimir sólo los que son estudiantes. Complete el espacio para cumplir dicho objetivo.

static void PrintStudentsOnly(IEnumerable<object> people) {

foreach (var student in people.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

Console.WriteLine(student.Name);

}

**Objetivos:** covarianza, contravarianza, herencia y polimorfismo, composición y encapsulamiento (*wrapper*)

# Problema 4 - Operadores y conversores

Impelente en C# una clase Complex que represente a los números complejos. Dicha clase debe cumplir que:

* + Console.WriteLine(c) debe imprimir el número complejo en formato
  + Complex c = 35.0; debe compilar y representa el número .
  + El tipo debe soportar las operaciones comunes (mediante la redefinición de operadores) en los complejos: suma (), resta (), multiplicación (), división (), comparación, igualdad, potencia (^).
  + Implemente propiedades que puedan ser útiles. (Real, Imaginaria, Rho, Theta, Inverso, Conjugado).
  + Un *array* de números complejos tiene que poder ordenarse con Array.Sort(). Suponga un posible orden para el dominio de los complejos primero ordenado según su parte real, y luego por su parte imaginaria.
  + double real = (double)complex; debe asignar a la variable “real” la parte real del número complejo, pero double real = complex; debe dar error en compilación.
  + Proponga una forma de construcción clara, legible y que no dé lugar a ambigüedades.

**Objetivos:** operadores, conversores, delegados