Introducción

Tipos de Datos Abstractos y Estructuras de Datos

Verónica E. Arriola-Rios

Facultad de Ciencias, UNAM

6 de agosto de 2024

Tipos

Temas

Tipo de datos

- Un *tipo de datos* especifica la interpretación semántica de la información introducida en la computadora, es decir, su significado y, por ende, cómo operar con ella.

Código: Tipo de dato

```
1 int x = 5, y = 8;
2 int res = x + y;
3
4 string a = "Tipo ";
5 string g = "de dato";
6 string concat = a + g;
```

Tipos de datos en C#

- Tipos base en `.NET`: espacio de nombres `System`

  https://learn.microsoft.com/en-us/dotnet/standard/class-library-overview

Ejemplos:

Categoría	Clase	C#	Descripción
Entero	Int32	int	Entero de 32 bits con signo
	UInt32	uint	Entero de 32 bits sin signo
Lógico	Boolean	bool	Valor booleano `true` o `false`
Otros	Object	object	La raíz de la jerarquía de objetos
	String	string	Secuencia inmutable de caracteres unicode

Ejemplo en C#

Creación de un proyecto:

```
1 mkdir MiProyecto
2 cd MiProyecto
3 dotnet new console
```

En el archivo `Program.cs` escribir:

```
1  Console.WriteLine("¿Cómo te llamas?");
2  string n = Console.ReadLine();                    // Tipo cadena
3  int x;
4  while(true)
5  {
6    try
7    {
8      Console.WriteLine("Escribe un entero:");
9      x = Convert.ToInt32(Console.ReadLine());     // Tipo entero
10     break;
11   }
12   catch (FormatException fe)
13   {
14     Console.WriteLine("Ese no fue un entero.");
15   }
16 }
17 Console.WriteLine($"{n} me diste el número {x}");
```

Ejecutar con:

```
dotnet run
```

Si sólo se quiere compilar usar:

```
dotnet build
```

Temas

Tipos abstracto de datos (TAD)

- Un *tipo abstracto de datos* es una especificación formal algebraica de:
  - Un conjunto de datos
  - Las operaciones que pueden realizarse con ellos con:
    1. Las *precondiciones* que se deben cumplir para poder realizar la operación.
    2. Las *postcondiciones* o relaciones que se satisfarán tras haber ejecutado la operación.
- No indica cómo serán representados estos datos en ningún sistema en particular, ni los detalles de cómo se llevarán a cabo las operaciones. Por ello son **abstractos**.

Crear un proyecto:

```
mkdir Números
cd Números
dotnet new console
rm -f Program.cs        # Usaremos otro Main
```

Código: API (Interfaz de programación de aplicaciones). Matemáticas/IComplejo.cs

```
1  namespace Matemáticas;
2
3  public interface IComplejo
4  {
5      /// DATOS z = x + yi
6      // Propiedades
7      Real Real { get; set; }
8      Real Imaginaria { get; set; }
9
10     // OPERACIONES
11     /// Norma = raíz cuadrada de (x*x + y*y)
12     Real Norma();
13     /// conjugado(z) = x - yi
14     Complejo Conjugado();
15 }
```

Temas

Estructuras de datos

"Las estructuras de datos son las formas de representación interna de datos de la computadora, mediante las que se representa cualquier situación en la computadora, es decir, son los tipos de datos que maneja la máquina.

Por ejemplo, podemos representar a un trabajador mediante los datos nombre del empleado, número de horas trabajadas, cuota por hora, etcétera." López Román 2011

Código: Estructura de datos. Matemáticas/Real.cs

```
1  namespace Matemáticas;
2
3  public class Real
4  {
5    /// DATOS
6    // Propiedad
7    public double Valor { get; }
8
9    // OPERACIONES
10   public Real(double valor)
11   {
12       Valor = valor;
13   }
14
15   public static Real operator *(Real a, Real b)
16       => new Real(a.Valor * b.Valor);
17   public static Real operator +(Real a, Real b)
18       => new Real(a.Valor + b.Valor);
19   public static Real operator -(Real a)
20       => new Real(-a.Valor);
21   public override string ToString() => $"{Valor}";
22 }
```

Código: Estructura de datos. Matemáticas/Complejo.cs

```
1  namespace Matemáticas;
2
3  public class Complejo : IComplejo
4  {
5    // DATOS z = x + yi
6    // Atributos
7    private Real _x;
8    private Real _y;
9
10   // Propiedades
11   public Real Real { get => _x; set => _x = value; }
12   public Real Imaginaria { get => _y; set => _y = value; }
13
14   // OPERACIONES
15   public Complejo(Real x, Real y) { this._x = x; this._y = y; }
16   /// <summary> Norma = raíz cuadrada de (x*x + y*y) </summary>
17   public Real Norma()
18   {
19     return new Real(Math.Sqrt((_x * _x + _y * _y).Valor));
20   }
21   /// <summary> conjugado(z) = x - yi </summary>
22   public Complejo Conjugado()
23   {
24     return new Complejo(_x, -_y);
25   }
26   public override string ToString() => $"{Real} + {Imaginaria} i";
27 }
```

Código: Estructura de datos. Matemáticas/Programa.cs

```
1  namespace Matemáticas;
2
3  class Programa
4  {
5      static void Main(string[] args)
6      {
7          var c = new Complejo(new Real(3.0), new Real(-2.5));
8          Console.WriteLine(c);
9      }
10 }
```

Ejecutar con:

```
1  dotnet run
```

Estructuras de Datos como contenedores

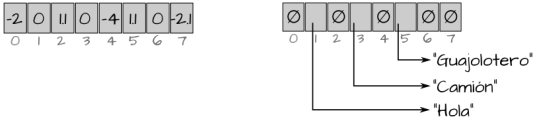
- Las *estructuras de datos* son particularmente importantes también para el **almacenamiento** eficiente de otros datos.
- Estructuras como *listas*, *árboles* y *diccionarios* permiten almacenar objetos de otros tipos, recuperándolos de forma eficiente.
- Se definen TDAs con las políticas de almacenamiento, manejo y recuperación dentro de estas estructuras.
- Cada TDA puede ser implementado en más de una forma. Algunas variantes serán más o menos eficientes dependiendo del contexto donde se utilizará a la estructura.

Temas

Bloques constructores

Podemos identificar dos técnicas básicas para construir estructuras que funcionen como almacen:

- *Arreglos*. El espacio apartado para los objetos almacenados son direcciones contiguas en la memoria.

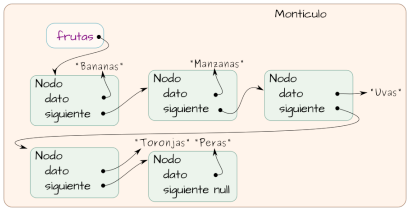


Bloques constructores

Podemos identificar dos técnicas básicas para construir estructuras que funcionen como almacen:

- *Arreglos*. El espacio apartado para los objetos almacenados son direcciones contiguas en la memoria.

- *Referencias* Un objeto o *nodo* contiene la dirección de otros objetos o nodos, que contienen a los datos de interés, generando una topología que permite accederlos según es requerido.



Bibliografía

Bibliografía I

López Román, Leobardo (2011). *Programación estructurada y orientada a objetos. Un enfoque algorítmico.* 3.ª ed. Alfaomega.

Licencia

Creative Commons
Atribución-No Comercial-Compartir Igual