Strutture Software 1

Docente: Fabio Solari
Tel.: 010-3532059
E-mail: fabio.solari@unige.it

Tutor: Manuela Chessa
Tel.: 010-3532794
E-mail: manuela@dibe.unige.it

URL: http://www.pspc.dibe.unige.it

OBIETTIVI FORMATIVI

- Presentare i principali metodi utilizzati per organizzare e rappresentare l'*informazione* (le strutture dati) al fine di ottenerne una *elaborazione* efficiente (gli algoritmi).

- Gli algoritmi vengono presentati dal punto di vista delle strutture dati.
- Le strutture dati vengono presentate in un ambiente orientato agli oggetti.

OBIETTIVI FORMATIVI

- Le strutture dati sono introdotte prima come tipi di dati astratti, definite mediante la loro interfaccia.

- In seguito si passa alla loro realizzazione concreta in un linguaggio di programmazione.

- Infine si mostrano alcuni esempi significativi di un loro utilizzo in *ambito ingegneristico*.

ARGOMENTI TRATTATI

- Richiami di programmazione: I/O dati, array, ricorsione.
- Analisi della complessità degli algoritmi: notazione O-grande.
- Ordinamento e ricerca: algoritmi elementari e algoritmi efficienti.
- Liste : specifica e implementazione.
- Pile e code: specifica e implementazione.
- Insiemi e tabelle hash : specifica e implementazione.
- Librerie di collezioni: `System.Collections`.

INDICAZIONI SULL'ESAME

- *Capacità operative*: Risolvere problemi impiegando le strutture dati e gli algoritmi più opportuni, utilizzando le tecniche di programmazione presentate.
- *Forme didattiche*: Lezioni ed esercitazioni a calcolatore. Ogni studente dovrà documentare lo svolgimento delle esercitazioni su un *quaderno di laboratorio*.
- *Tipologia dell'esame*: Valutazione delle esercitazioni e prova orale.

RIFERIMENTI BIBLIOGRAFICI

- Dispense fornite a lezione.
- Testo di consultazione:
  - P. Crescenzi, G. Gambosi, R. Grossi. *Strutture di dati e algoritmi*. Addison Wesley, 2006.
- Per i linguaggi *C* e *C#* si fa riferimento agli insegnamenti di "Informatica 1" e "Programmazione ad oggetti per sistemi elettronici 1".

SOMMARIO: richiami di programmazione

- I/O dati:
  - Tastiera e file.
  - Conversione tra stringhe e numeri.
  - Argomenti alla linea di commando.
- Generazione di numeri pseudo-casuali.
- Classi e oggetti.
- Array (struttura dati):
  - Monodimensionale (1D) di dati primitivi e oggetti. Array e metodi.
  - Bidimensionale (2D): matrice.

I/O

- In generale, per avere un'applicazione reale, i programmi devono avere uno scambio di dati con l'esterno: in modo interattivo con la tastiera e il monitor, oppure attraverso file.
- Vediamo alcuni esempi in modo da *uniformare* le modalità di gestione dell'input/output di dati.
- Il primo esempio riguarda la lettura da tastiera di due valori e la loro visualizzazione a monitor.

I/O

- Viene definita una classe (`Tester`) che contiene il `Main()` (*program entry point*).
- Dal `Main()` vengono chiamati i diversi *metodi statici* (i test) che implementano *ciò che si vuole verificare*.

```
class Tester
{
    static void Main(string[] args)
    {
        TestIOTastiera();
    }
    public static void TestIOTastiera()
    {
        //...
    }
}
```


I/O INTERATTIVO

```
public static void TestIOTastiera()
{
    double f;
    string str;
    Console.Write("Inserisci un double:");
    str = Console.ReadLine();

    f = Convert.ToDouble(str);
    Console.WriteLine("Hai inserito: {0}\n", f);

    Console.Write("Inserisci una stringa:");
    str = Console.ReadLine();
    Console.WriteLine("Hai inserito: {0}", str);
}
```

Converte la rappresentazione letterale (stringa) di un numero nel valore numerico corrispondente.

Una possibile uscita

```
Inserisci un double:-1.2e2
Hai inserito: -120

Inserisci una stringa:prova
Hai inserito: prova
```


I/O CON FILE

- Leggere un file (`file1.txt`) contenete un elenco di articoli e relativi prezzi e scrivere un nuovo file (`file2.txt`) con i prezzi maggiorati del 20%.

`file1.txt`

```
Articolo1
1.2
Articolo2
4.6
Articolo3
10.0
```

`file2.txt`

```
Articolo1
1.44
Articolo2
5.52
Articolo3
12.0
```


I/O CON FILE

```
using System.IO;
public static void TestFile()
{
    StreamReader fin = new StreamReader("file1.txt");
    StreamWriter fout = new StreamWriter("file2.txt");

    string sInput1,sInput2;
    double prezzo;

    while ((sInput1 = fin.ReadLine()) != null &&
                    (sInput2 = fin.ReadLine()) != null)
    {
        fout.WriteLine(sInput1);
        prezzo = Convert.ToDouble(sInput2) * 1.2;
        fout.WriteLine(prezzo);
    }
    fin.Close();
    fout.Close();
}
```

SPLIT DI STRINGHE

- Se la linea letta con `readLine()` contenesse diversi elementi, si posso separare come nel seguente esempio.

```
public static void TestStringhe()
{
    string str = "Una stringa composta da 6 parti";
    string[] strv;
    char spl = ' ';
    strv = str.Split(spl);
    for (int i = 0; i < strv.Length; i++)
        Console.WriteLine(strv[i]);
}
```

Una possibile uscita

```
Una
stringa
composta
da
6
parti
```


ARGOMENTI ALLA LINEA DI COMMANDO

- I dati di input vengono forniti direttamente al momento di lanciare l'applicazione da riga di commando.
- Un esempio in cui vengono inseriti due valori e poi si calcola la loro somma.

```
static void Main(string[] args)
{
  double x = Convert.ToDouble(args[0]) + Convert.ToDouble(args[1]);
  Console.WriteLine("La somma: {0}« , x);
}
```

Una possibile uscita

```
C:\temp>esempio.exe 10 -3
La somma: 7
```


GENERAZIONE DI NUMERI PSEUDO-CASUALI

- Utile per verificare algoritmi con diversi ingressi generici.
- Viene generata una sequenza di numeri pseudo-casuali identica se viene usato lo stesso seme per inizializzare l'oggetto.

- Un esempio di generazione di `int` tra 0 ed un valore assegnato (7) e di `float` tra 0.0 e 1.0.

GENERAZIONE DI NUMERI PSEUDO-CASUALI

```
public static void TestNumeriCasuali(){
    Random rnd = new Random();

    for (int i = 0; i < 10; i++)
        Console.Write("{0:F3} ", rnd.NextDouble());

    Console.Write("\n\n");

    for (int i = 0; i < 10; i++)
        Console.Write("{0} ", rnd.Next(7));
}
```

Una possibile uscita

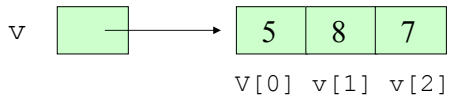
```
0.371 0.035 0.602 0.841 0.058 0.897 0.744 0.193 0.374 0.264

0 4 5 2 2 5 3 5 2 1
```

ARRAY 1D: dati primitivi

- La prima *struttura dati* che consideriamo è l'*array*: un insieme di valori dello stesso tipo disposti consecutivamente, ad ogni valore è associato un indice di posizione in modo univoco. Attraverso tali indici si accede ai dati memorizzati.

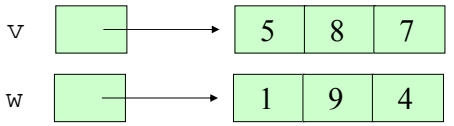
```
int[] v = new int[3];
v[0]=5; v[1]=8; v[2]=7;
```



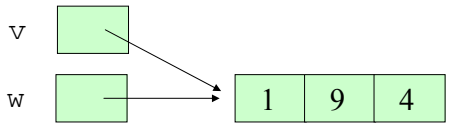
ARRAY 1D: dati primitivi

```
int[] v = new int[3];
v[0]=5; v[1]=8; v[2]=7;
int[] w = {1,9,4};
```

Gli array sono tipi per riferimento




```
v=w;
```



ARRAY 1D: esempio

- Trovare il valore massimo e minimo di un array la cui dimensione è specificata da tastiera, l'array viene inizializzato con valori casuali.

Una possibile uscita

```
Inserisci la dimensione: 10
13 10 61 34 75 3 60 83 85 66
(max,index) 85 8
(min,index) 3 5
```

Una possibile uscita

```
Inserisci la dimensione: 21
34 1 86 36 53 66 1 29 61 56 51 48 28 34 24 44 45 82 22 95 75
(max,index) 95 19
(min,index) 1 1
```


ARRAY 1D: esempio

```
public static void TestMinMax()
{
    Random rnd = new Random();

    Console.Write("Inserisci la dimensione: ");
    string str = Console.ReadLine();
    int n = int.Parse(str);

    int[] v = new int[n];

    for (int i = 0; i < v.Length; i++)
    {
        v[i] = rnd.Next(100);
        Console.Write("{0} ", v[i]);
    }
    Console.Write("\n");
```

La dimensione è specificata durante l'esecuzione

...

ARRAY 1D: esempio

```
...
    int max = v[0], min = v[0];
    int imax = 0, imin = 0;
    for (int i = 0; i < v.Length; i++)
    {
        if (v[i] > max)
        {
            max = v[i];
            imax = i;
        }
        if (v[i] < min)
        {
            min = v[i];
            imin = i;
        }
    }
    Console.WriteLine("(max,index) {0} {1}", max, imax);
    Console.WriteLine("(min,index) {0} {1}", min, imin);
}
```


ARRAY E METODI

- Poiché un *array* è un tipo per *riferimento*, sia il metodo chiamante sia il metodo chiamato modificano lo stesso *array* (*oggetto*).
- L'esempio mostra un metodo che azzera i valori dell'array passato come argomento.

- Un metodo può ritornare un riferimento ad un array creato al suo interno.
- L'esempio estrae una porzione dell'array passato come argomento e ritorna tale porzione come un nuovo array.

ARRAY E METODI

```
static void Main()
{
    int[] vett ={ 1, 2, 3, 4, 5 };
    for (int i = 0; i < vett.Length; i++)
        Console.Write("{0} ", vett[i]);
    Console.Write("\n");

    int [] subvett = sottoarray(vett,2,5);

    for (int i = 0; i < subvett.Length; i++)
        Console.Write("{0} ", subvett[i]);
    Console.Write("\n");

    azzera(vett);

    for (int i = 0; i < vett.Length; i++)
        Console.Write("{0} ", vett[i]);
    Console.Write("\n");
}
```


ARRAY E METODI

```
public static void azzera(int[] v)
{
    for (int i = 0; i < v.Length; i++)
        v[i] = 0;
}

public static int[] sottoarray(int[] v, int l, int h)
{
    int[] tmp = new int[h - l];
    for (int j = l, i = 0; j < h; j++, i++)
        tmp[i] = v[j];
    return tmp;
}
```

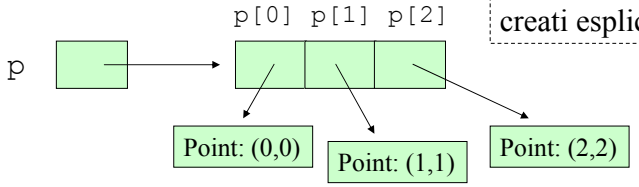
Una possibile uscita

```
1 2 3 4 5
3 4 5
0 0 0 0 0
```

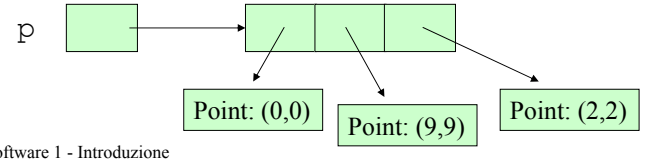
ARRAY 1D: oggetti

```
Point[] p = new Point[3];
p[0] = new Point(0,0);
p[1] = new Point(1,1);
p[2] = new Point(2,2);
```

Gli array di *oggetti* contengono riferimenti agli oggetti e i singoli elementi devono essere creati esplicitamente.




```
p[1].Set(9,9);
```



CLASSI E OGGETTI

- Quando si parla di *tipi definiti dall'utente*, cioè *classi*, si deve porre attenzione ad alcune considerazioni legate all'OOP (*object-oriented programming*):
  - È importante nascondere i dettagli di implementazione della classe (*encapsulation*) all'utente degli oggetti. Si semplifica la programmazione: si creano istanze e si inviano i messaggi appropriati.
  - Di conseguenza si fa riferimento alla nozione di protezione dei dati (*data protection*): lo stato di un oggetto deve essere dichiarato `private` e non `public`.
  - Solo i metodi che descrivono il comportamento/servizi forniti all'utente degli oggetti sono dichiarati `public`.

CLASSI E OGGETTI

- Vediamo una possibile implementazione di una classe che rappresenta i punti del piano. La classe `Point` è definita all'interno del file `Point.cs` e la classe che contiene il `Main()` nel file `PointTester.cs`.

```
public class Point
{
    private int x;
    private int y;

    public Point() { }

    public Point(int a, int b)
    {
        x = a;
        y = b;
    }
```

…

CLASSI E OGGETTI

…

```
public int X
{
    get
    {
        return x;
    }
    set
    {
        x = value;
    }
}
public int Y
{
    get
    {
        return y;
    }
    set
    {
        y = value;
    }
}
```

Al posto dei tradizionali metodi `Get()` e `Set()`, utilizzati per modificare lo stato dell'oggetto, si sono utilizzate le proprietà (*properties*), che *simulano* un accesso pubblico ai campi dei dati.

Metodo ereditato da `object` e modificato dalla classe derivata `Point`.

```
public override string ToString()
{
    string tmp;
    tmp = x + " " + y;
    return tmp;
}
```

}

CLASSI E OGGETTI

```
static void Main()
{
    Point p = new Point();
    Console.WriteLine(p);

    Point[] pv = new Point[3];
    for (int i = 0; i < pv.Length; i++)
        pv[i] = new Point(5+i, 5+i);

    pv[1].X = 9;
    pv[1].Y = 9;

    for (int i = 0; i < pv.Length; i++)
        Console.WriteLine(pv[i].ToString());
}
```

Costruttore di default.

Una possibile uscita

```
0 0
5 5
9 9
7 7
```


ARRAY 2D

- Si considerano gli *array multidimensionali* e in particolare gli *array rettangolari* (*matrici*): sono caratterizzati da un certo numero di righe (il primo indice) e per ogni riga lo stesso numero di colonne (il secondo indice).
- Si consideri, per esempio, una matrice di tre righe e cinque colonne inizializzata con valori interi consecutivi e crescenti.

Una possibile uscita

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14

ARRAY 2D

```
public static void TestMatrice()
{
    int r = 3, c = 5;

    int[,] matrice = new int[r, c];

    for (int i = 0; i < r; i++)
        for (int j = 0; j < c; j++)
            matrice[i,j] = i*5 + j;

    for (int i = 0; i < r; i++)
    {
        for (int j = 0; j < c; j++)
            Console.Write("{0} \t", matrice[i,j]);
        Console.Write("\n");
    }
}
```