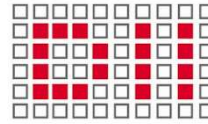




UNIVERSITÀ  
POLITECNICA  
DELLE MARCHE



Dipartimento di  
Ingegneria dell'Informazione

FACOLTÀ  
DI INGEGNERIA



# **LABORATORIO**

## **Algoritmi e Strutture Dati**

### **corso a.a. 2020**

**Dott. Andrea Sergiacomi**

[andrea.sergiacomi@regione.marche.it](mailto:andrea.sergiacomi@regione.marche.it)

# Programma #1

<https://www.mheducation.it/algoritmi-e-strutture-dati-9788838662621-italy#tab-label-product-description-title>

ALGORITMI E STRUTTURE DATI P. Foggia, M. Vento - McGraw-Hill



## Definizioni e Linguaggio C

esempi di algoritmi e strutture dati

interfaccia Pelles C

nozioni introduttive (sintassi, struttura dei programmi)

ricerca lineare e binaria



**CAP. 4**

***rif. CAP. 3***

## Algoritmi ricorsivi

il problema della Torre di Hanoi

calcolo del fattoriale (metodo iterativo vs ricorsione lineare)

ricorsione multipla: funzione di Fibonacci

ricorsione mutua: pari o dispari

ricorsione innestata: funzione di Ackermann



**CAP. 2**

***rif. CAP. 1***

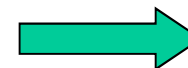
## Algoritmi di base

ordinamento :

selection sort

insertion sort

bubble sort (... iterativi)



**CAP. 4**

***rif. CAP. 3***

merge sort

quick sort (... ricorsivi)

# Programma #2

## Liste dinamiche

funzioni iterative e ricorsive per:

creare, riempire, stampare una lista  
cercare, cancellare un elemento da lista ordinata  
effettuare operazioni avanzate (ordinamento,  
ricerca minimo, conteggio e somma dei nodi,  
inversione, copia, fusione)

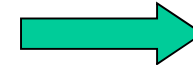


**CAP. 6**

*ref. CAP. 5*

## Alberi binari di ricerca

algoritmi ricorsivi BST : visita, minimo, altezza,  
operazioni di accumulazione (somma, conteggio)



**CAP. 8**

## Tabelle hash

ricerca e visita su tabella a indirizzamento chiuso



**CAP. 7**

# Listati di codice

- 01 **Definizioni e Linguaggio C** ..... (slide 1-14)  
helloworld.c, pointer.c, setupPELLESC9.exe,  
Neural-Style-Transfer/train\_TensorFlow.py
- 02 **Algoritmi di Ricerca** ..... (slide 15-24)  
linear\_search.c, binary\_search.c, binary\_search\_recursive.c
- 03 **Algoritmi ricorsivi** ..... (slide 25-39)  
Hanoi.c, Fattoriale\_ciclo.c, Fattoriale\_ricorsione.c,  
Fibonacci.c, Pari\_o\_Dispari.c, Ackermann.c
- 04 **Algoritmi ordinamento iterativi** ..... (slide 40-44)  
selection\_sort.c, insertion\_sort.c, bubble\_sort.c
- 05 **Algoritmi ordinamento ricorsivi** ..... (slide 45-47)  
merge\_sort2.c, quick\_sort.c
- 06 **Liste dinamiche** ..... (slide 48-63)  
list\_crea-riempi-stampa.c, list\_cerca-cancella.c,  
ordinaLISTA\_DEF.c, LISTE\_operazioni\_avanzate.c
- 07 **Alberi binari di ricerca** ..... (slide 64-67)  
binarytree\_min\_count\_sum\_depth.c
- 08 **Tabelle hash a indirizzamento chiuso** ..... (slide 68-72)  
chained\_hashtable.c

# Algoritmi? Strutture Dati? Strumenti



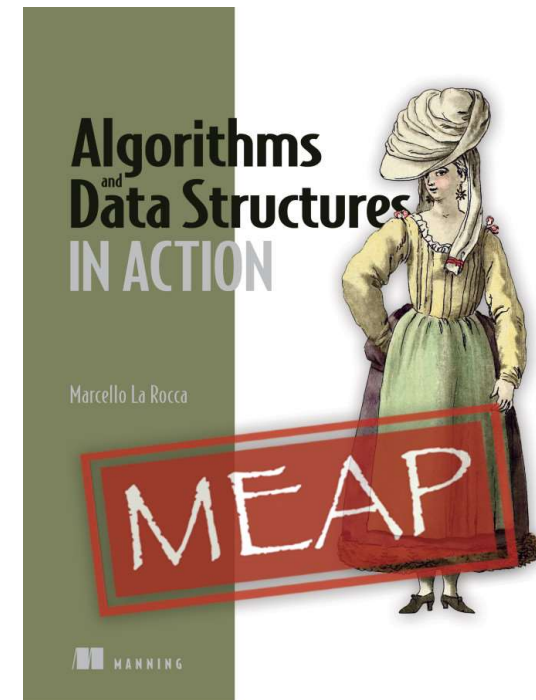
Concetti interconnessi e interdipendenti: metaforicamente le **strutture dati** sono come dei **nomi**, mentre gli **algoritmi** dei **verbi**.

Un algoritmo è un procedimento che risolve un determinato problema attraverso un numero **finito** di passi elementari, **chiari e non ambigui**.

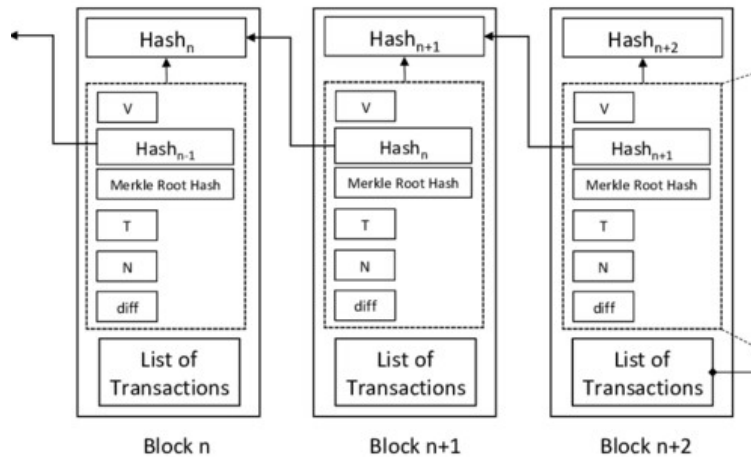
Data structures are the substrate, a way to organize an area of memory to **represent data**.

Algorithms are procedures, sequence of instructions aimed to **transform data**.

Every data structure, moreover, implicitly defines algorithms that can be performed on it: for instance, **methods** to add, retrieve and remove elements to/from the data structure.



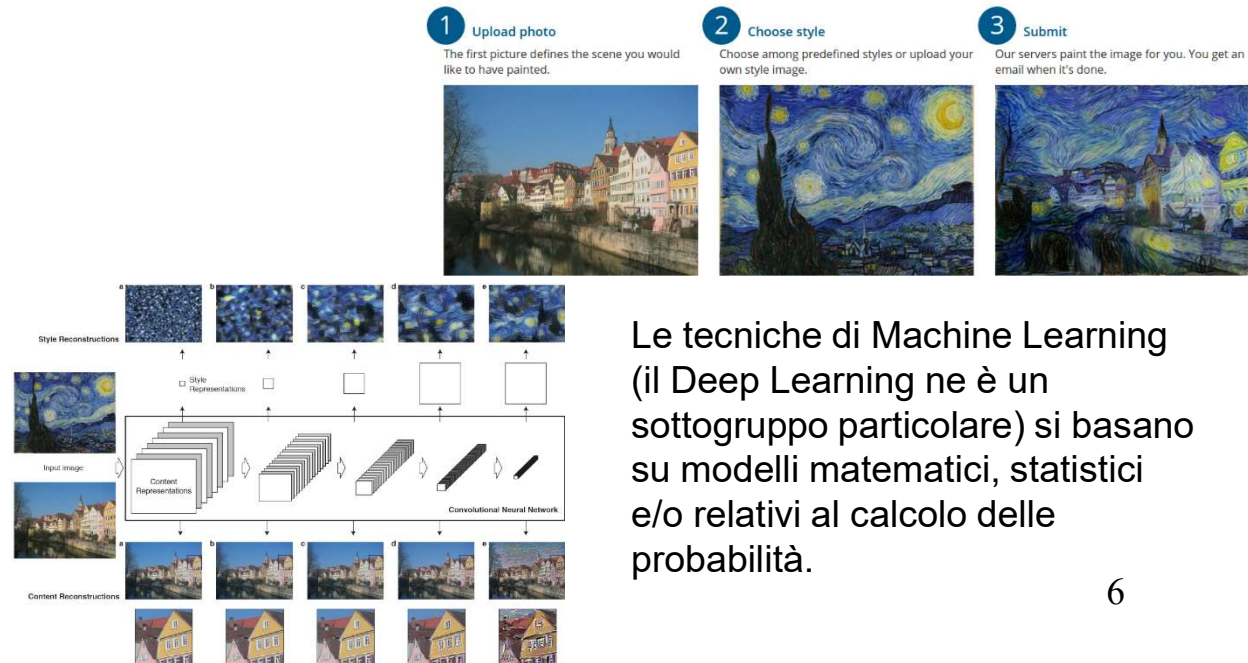
# Un paio di esempi attuali



Le **Blockchain** (catene di blocchi), architetture basate sulle DLT (Distributed Ledger Technologies = registri distribuiti), utilizzano una *struttura dati* condivisa e immutabile. I blocchi crittografati sono concatenati in ordine cronologico. La dimensione cresce nel tempo ma le transazioni precedenti, una volta approvate consensualmente dai nodi, non possono più essere manipolate (grazie al fatto che l'**hash** di ciascun blocco viene generato in base sia ai dati contenuti al suo interno che all'hash del blocco precedente). La non ripudiabilità è garantita indipendentemente dal fatto che i nodi coinvolti conoscano l'identità reciproca o si fidino l'uno dell'altro.

La web application

<https://deepart.io/> è costruita sopra un *algoritmo* che utilizza tecniche di **Deep Learning** (basate sulle reti neurali), applicate nel campo della computer vision e del digital image processing, in grado di trasferire lo stile di un'immagine ad un'altra, arrivando, ad esempio, a rendere un "Picasso" la foto di un profilo Facebook o a "VanGoghizzare" un paesaggio.



Le tecniche di Machine Learning (il Deep Learning ne è un sottogruppo particolare) si basano su modelli matematici, statistici e/o relativi al calcolo delle probabilità.



# Pelles C



**WIKI :** [http://it.wikipedia.org/wiki/Pelles\\_C](http://it.wikipedia.org/wiki/Pelles_C) (in italiano) +  
**Download vers. 9.0 Windows 10 32/64-bit:** <http://www.smorgasbordet.com/pellesc/>

```
/* Ricorsione annidata - funzione di Ackermann
 * dati due numeri positivi
 * A(m,n) = n+1          se m=0
 * A(m,n) = A(m-1,n)     se m>0 e n=0
 * A(m,n) = A(m-1,A(m,n-1)) se m>0 e n>0
 *
 * NOTA -> la funzione cresce molto rapidamente: già con A(4,1) si può avere stack
 */

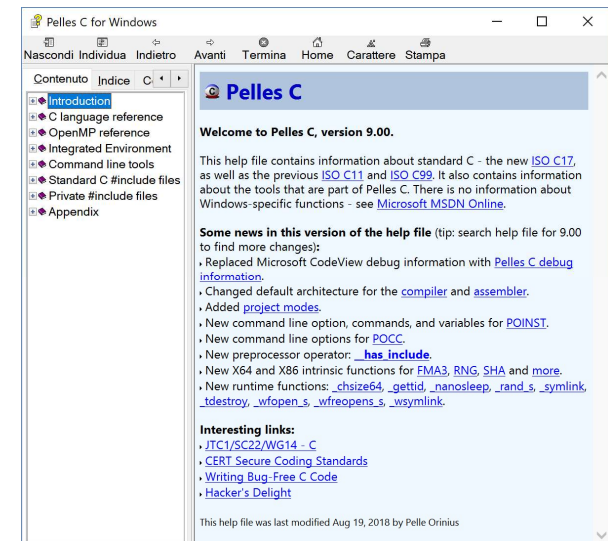
#include <stdio.h>

int ackermann(int m, int n)
{
    if (m < 0 || n < 0)
        return -1; /* la funzione non è definita per numeri negativi ! */

    if (m == 0)
        return n+1;
    else
        if (n == 0)
            return ackermann(m-1, 1);
        else
            return ackermann(m-1, ackermann(m, n-1));
}

void stampaRisultato(int M, int N)
{
}
```

DEMO Interfaccia  
(helloworld.c) +  
Help contestuale  
(click dx)







# Pelles C



Browse information


The system definitions database is used for code completion of symbols from standard include files. That is, all files found in the include folders listed in the Options dialog. If you change the list of include folders you may want to rebuild this database.

System definitions database

Name: C:\Users\andrea.sergiacomi\AppData\Roaming\Pelles C\sysdefs.tag

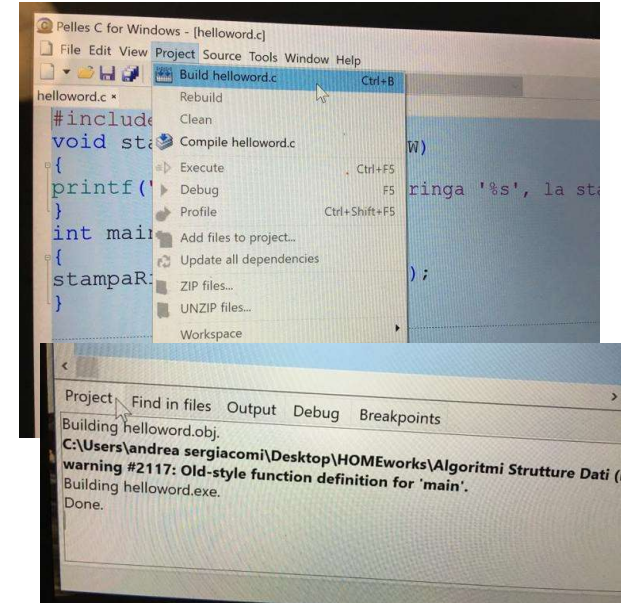
Size:

Created: Never saved

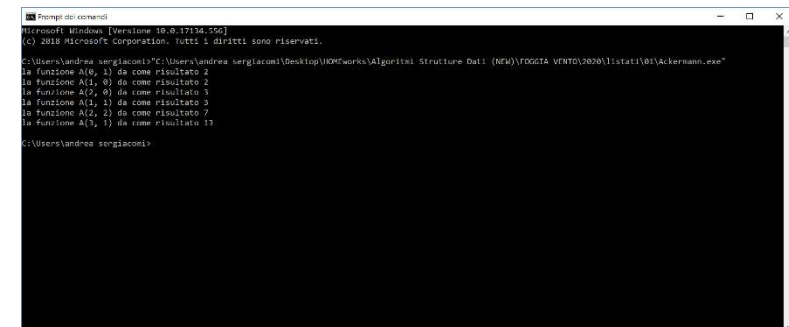
 The database is missing. You are recommended to build it now. Most users would want this. An average database is 25-100 MB.

New default project

This command requires an active project. Select a type, and click OK.



	Nome	Ultima modifica	Tipo	Dimensione
output	output	08/03/2020 14.55	Cartella di file	
Ackermann.c	Ackermann.c	08/03/2020 14.53	File C	1 KB
Ackermann.exe	Ackermann.exe	08/03/2020 14.55	Applicazione	39 KB
Ackermann.ppj	Ackermann.ppj	08/03/2020 14.55	Pelles C Project File	2 KB
Ackermann.ppx	Ackermann.ppx	08/03/2020 15.00	File PPX	1 KB
Ackermann.tag	Ackermann.tag	08/03/2020 14.55	File TAG	20 KB



## INSTALLA + COMPILA + ESEGUI



# Nozioni sul linguaggio C

- *sintassi*

*/\* commenti*

*\* altri commenti*

*ultima riga di commento \*/*

*// commento su singola riga*

*#include <NOME\_HEADER\_LIBRERIA\_FUNZIONI.h>*

*#define DIMENSIONE\_ARRAY\_COSTANTE (10)*

*costante const x – variabile x – puntatore \*x – indirizzo in memoria &x*

*tipi: void unsigned/signed int short long float double char enum array struct*

*! not || or && and ++ incremento 1 -- decremento 1*

*== verifica equivalenza = assegnazione*



<https://www.html.it/guide/guida-c/>

# Nozioni sul linguaggio C

- *struttura base di un programma ed output*

```
#include <stdio.h>
```

```
void stampaRisultato(char *HW)
```

```
{
```

```
printf("avuta in input la stringa '%s', la stampo a video:  
\n%s\n\n", HW, HW);
```

```
}
```

```
int main()
```

```
{
```

```
stampaRisultato("hello word");
```

```
}
```

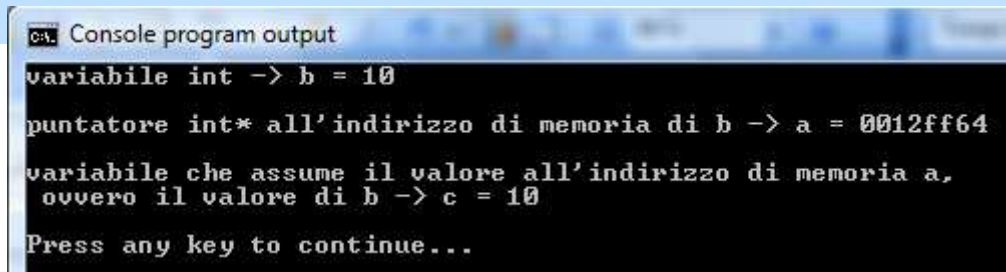


# Nozioni sul linguaggio C

- *esempio sull'utilizzo dei puntatori (pointer.c)*

```
#include <stdio.h>
int main()
{
    int b,c; /* definisco 2 interi */
    int *a; /* definisco a come puntatore a intero */
    b=10; /* assegno a b il valore 10 */
    a=&b; /* assegno ad a l'indirizzo di memoria di b */
    c=*a; /* assegno a c il valore contenuto all'indirizzo di memoria specificato da a
           -> siccome a contiene l'indirizzo in memoria di b c sara' uguale a b */
    printf("variabile int -> b = %d\n\n",b);
    printf("puntatore int* all'indirizzo di memoria di b -> a = %p\n\n",a);
    printf("variabile che assume il valore all'indirizzo di memoria a,\n ovvero il valore di b -> c = %d\n\n",c);
    return 0;
}
```

<https://www.devapp.it/wordpress/12-i-puntatori-parte-1/>



```
Console program output
variabile int -> b = 10
puntatore int* all'indirizzo di memoria di b -> a = 0012ff64
variabile che assume il valore all'indirizzo di memoria a,
ovvero il valore di b -> c = 10
Press any key to continue...
```

\* un indirizzo di memoria è rappresentabile con 8 o 16 caratteri (a seconda che si lavori a 32bit o a 64bit)

# Nozioni sul linguaggio C

- *utilizzo dei puntatori in strutture dinamiche (vettori vs liste)*

```
#define ARRAY_MAX 2
```

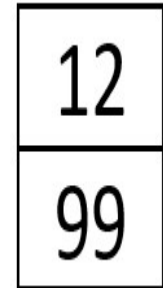
```
void main()
```

```
{ /* definizione vettore monodimensionale di numeri interi */
```

```
    int vet[ARRAY_MAX];
```

```
    vet[0]=12;
```

```
    vet[1]=99;}
```



```
#include <stdlib.h> // libreria funzioni di allocazione dinamica della memoria
```

```
/* definizione lista dinamica contenente numeri interi */
```

```
struct Snode {
```

```
    int info;           // elemento informativo
```

```
    struct SNode *link; // elemento puntatore al successivo nodo };
```

```
typedef struct SNode TNode;
```

```
typedef TNode *TList;
```

```
    // pseudocodice ----- CICLO listCreate() -----
```



```
Tnode *new = (TNode *)malloc(sizeof(TNode));
```

```
scanf("%d", & new->info);
```

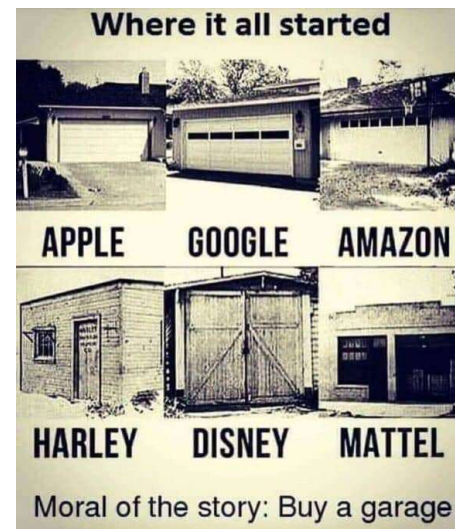
```
new->link = (TNode *)malloc(sizeof(TNode)); // NULL
```

# Nozioni sul linguaggio C

alcune specifiche per il formato dell'output nella funzione **printf**

<i><b>specifier</b></i>	<b>Output</b>	<b>Example</b>
<code>%c</code>	Character	a
<code>%d</code> or <code>%i</code>	Signed decimal integer	+392
<code>%f</code>	Decimal floating point	392.65
<code>%s</code>	String of characters	sample
<code>%u</code>	Unsigned decimal integer	7235
<code>%p</code>	pointer	0012ff64
<code>\n</code>	New Paragraph. Nothing printed.	

# Ma l'importante...



È che programmare (creare con logica) - o comunque trattare il digitale - vi diverta e, se possibile, diventi una vostra passione ed il vostro futuro lavoro 😊





# A proposito di ricerca (FB da BigData a Echo Chambers)

da Giacomo Papi <il censimento dei radical chic>  
«molti segni ci dicono che l'umanità sta imitando le spugne di mare, che non hanno un cervello perché non l'hanno voluto. Milioni di anni fa questi simpatici animali, così utili e piacevoli sotto la doccia, avrebbero potuto sviluppare un sistema nervoso, invece rinunciarono per ragioni evolutive: possederlo era inutile e richiedeva troppa fatica. Non avevano torto: se quello che devi fare nella vita è filtrare acqua alla ricerca di particelle di cibo a che cosa ti serve pensare e provare emozioni? È un dubbio che gli uomini di oggi possono riformulare senza punto di domanda: se quello che devi fare nella vita è **filtrare informazioni alla ricerca di ciò che confermerà quello in cui già credi**, cercare di comprendere come stanno davvero le cose è solo un fastidio.»



da Concita De Gregorio  
<nella notte>  
«quando è gratis, in vendita ci sei tu.»

# Search and problem solving #1

Ricerca Operativa: disciplina che tratta dello sviluppo e dell'applicazione di metodi scientifici per la soluzione ottimale di problemi di decisione

Many problems can be phrased as search problems. This requires that we start by formulating the alternative choices and their consequences.

**ELEMENTS OF AI** by Finland - University of Helsinki

<https://course.elementsofai.com/2/1> (*free online course*)

## **Toy problem: chicken crossing**

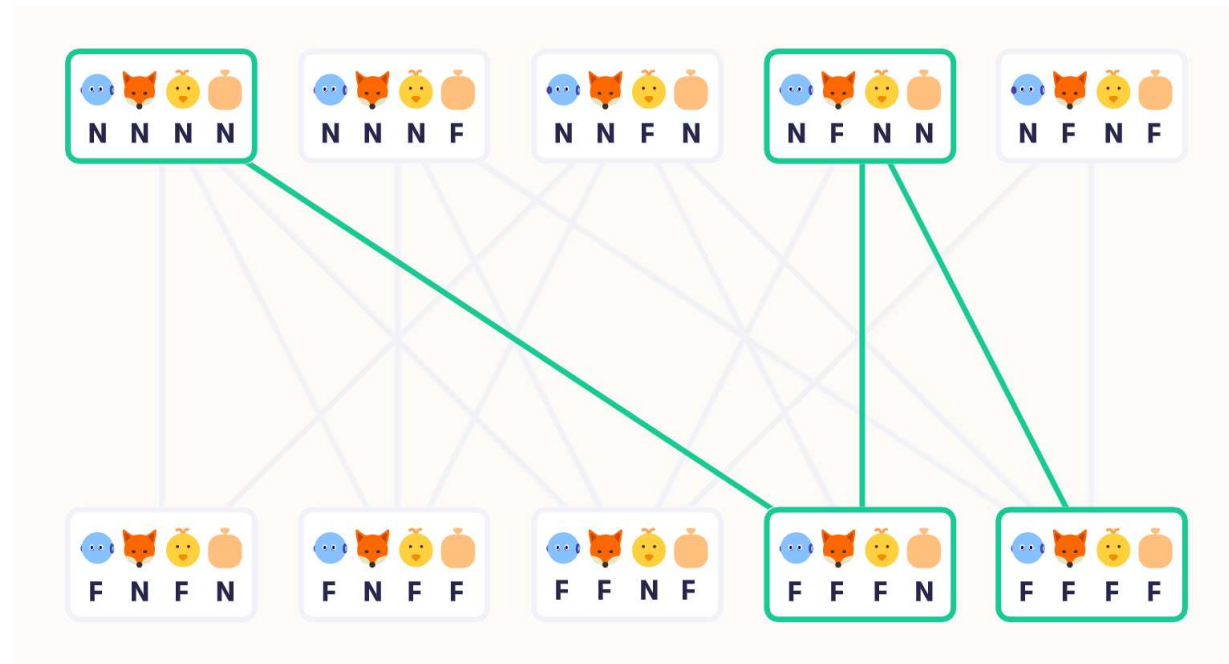
A robot on a rowboat needs to move three pieces of cargo across a river: a fox, a chicken, and a sack of chicken-feed. The fox will eat the chicken if it has the chance, and the chicken will eat the chicken-feed if it has the chance, and neither is a desirable outcome. The robot is capable of keeping the animals from doing harm when it is near them, but only the robot can operate the rowboat and only two of the pieces of cargo can fit on the rowboat together with the robot.

# Search and problem solving #2

## Toy problem: chicken crossing

STATE: N Near side (of the river) F Far side

There is one shortest paths that lead from the start NNNN to the goal FFFF. It is NNNN -> FFFN -> NFNN -> FFFF. Intuitively, the strategy is to move the fox on the other side first, and then go back get the chicken to save it from being eaten, finally taking the remaining objects (chicken and feed) from the near side to the far side, to reach the goal.

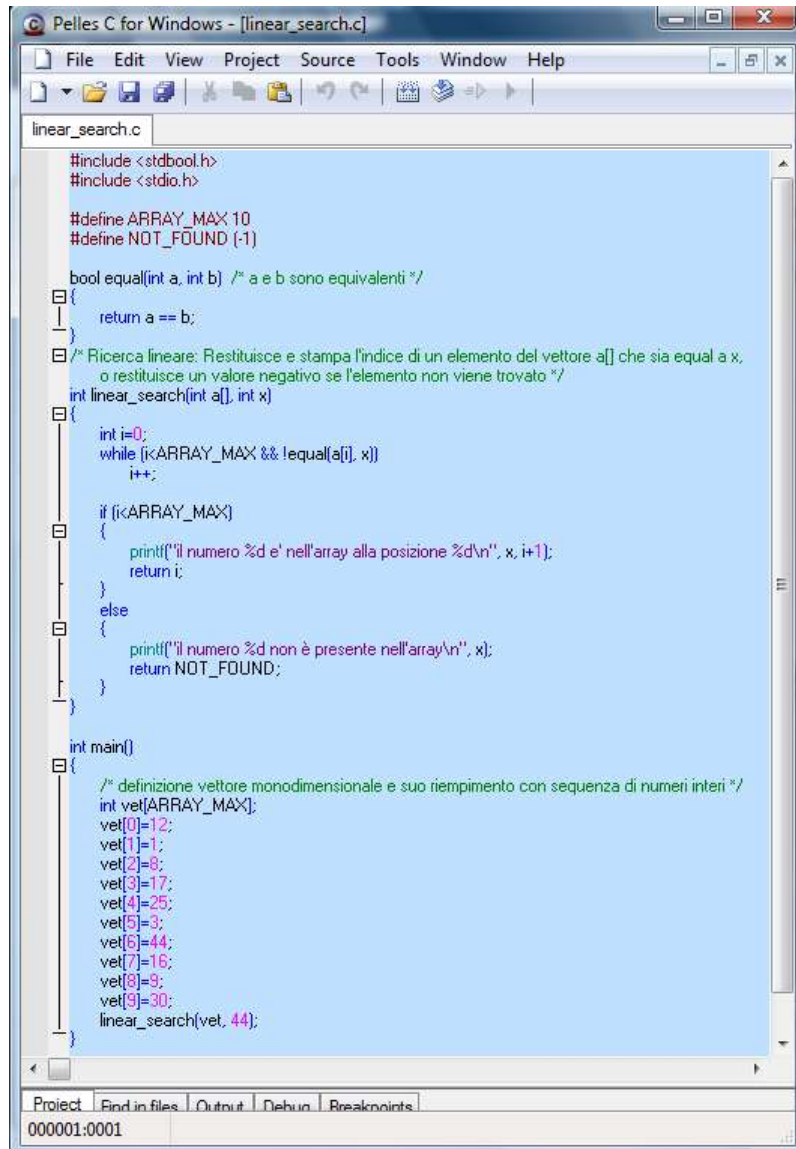


# Ricerca lineare o sequenziale

E' il più semplice algoritmo di ricerca:  
per cercare un determinato elemento in una  
struttura dati, esamina in sequenza tutti i  
singoli elementi.

Nell'esempio la struttura dati è un vettore e  
l'algoritmo è implementato in modo  
iterativo: il ciclo termina quando non ci sono  
più elementi da esaminare o quando si  
trova l'elemento desiderato (al primo  
successo).

# Ricerca lineare o sequenziale



```
#include <stdbool.h>
#include <stdio.h>

#define ARRAY_MAX 10
#define NOT_FOUND (-1)

bool equal(int a, int b) /* a e b sono equivalenti */
{
    return a == b;
}

/* Ricerca lineare: Restituisce e stampa l'indice di un elemento del vettore a[] che sia equal a x,
o restituisce un valore negativo se l'elemento non viene trovato */
int linear_search(int a[], int x)
{
    int i=0;
    while (i<ARRAY_MAX && !equal(a[i], x))
        i++;

    if (i<ARRAY_MAX)
    {
        printf("il numero %d e' nell'array alla posizione %d\n", x, i+1);
        return i;
    }
    else
    {
        printf("il numero %d non e' presente nell'array\n", x);
        return NOT_FOUND;
    }
}

int main()
{
    /* definizione vettore monodimensionale e suo riempimento con sequenza di numeri interi */
    int vet[ARRAY_MAX];
    vet[0]=12;
    vet[1]=1;
    vet[2]=8;
    vet[3]=17;
    vet[4]=25;
    vet[5]=3;
    vet[6]=44;
    vet[7]=16;
    vet[8]=9;
    vet[9]=30;
    linear_search(vet, 44);
}
```

LINEAR\_SEARCH.c -> IL  
LISTATO IN C CHE  
IMPLEMENTA  
L'ALGORITMO DI RICERCA  
SEQUENZIALE PER  
DETERMINARE **SE E IN  
CHE POSIZIONE**, IN UN  
VETTORE DI 10 NUMERI  
INTERI SIA PRESENTE  
(ALMENO 1 VOLTA) IL  
NUMERO 44.

# Ricerca binaria o dicotomica #1

Supponiamo di voler cercare un nominativo in un elenco telefonico. L'algoritmo di ricerca sequenziale ci costringerebbe a valutare tutti i cognomi che precedono quello che ci interessa, richiedendo, in media, un numero di operazioni proporzionale alla dimensione della struttura dati.

Per ottimizzare e velocizzare la ricerca utilizzeremo invece la ricerca dicotomica (si basa sullo stesso concetto con cui sfogliamo un vocabolario alla ricerca di una parola). Naturalmente è necessario assumere che la nostra struttura dati di riferimento sia ordinata in ordine alfabetico crescente.



# Ricerca binaria o dicotomica #2

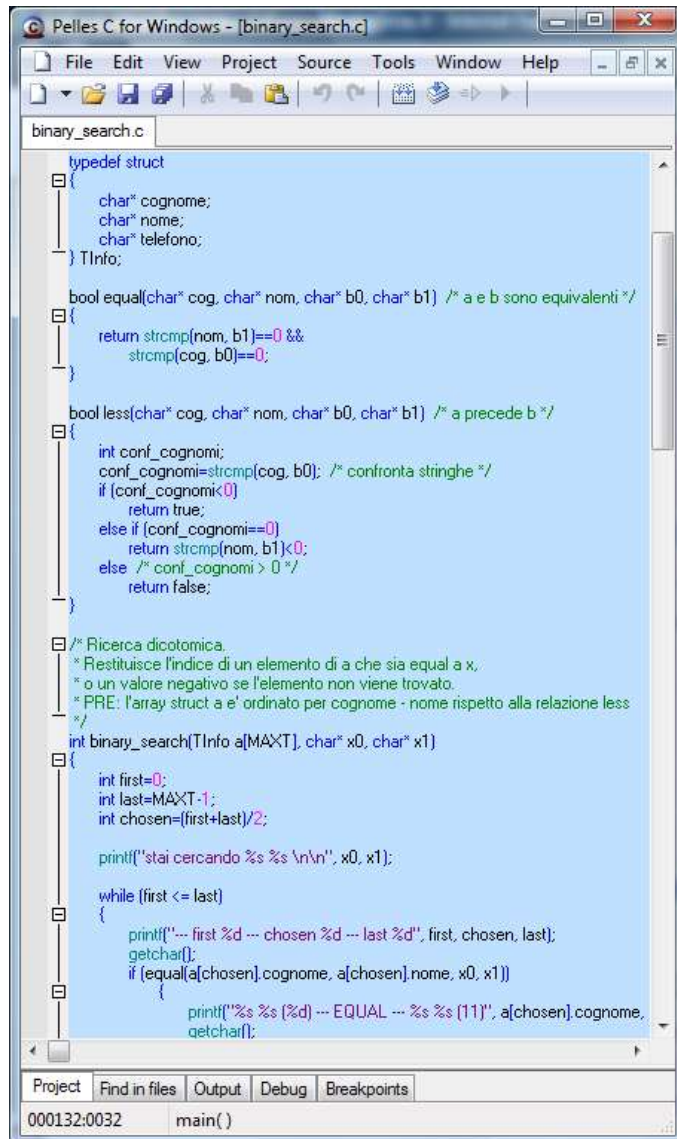
Una prima implementazione ITERATIVA: il procedimento si basa sulla scelta di un elemento centrale nell'intervallo degli elementi utili.

Se da un confronto tra l'elemento scelto e quello desiderato emerge che sono uguali (equal) la ricerca è finita.

Se invece l'elemento scelto viene prima (less) dell'elemento desiderato, l'algoritmo procede riducendo e ridefinendo l'intervallo agli elementi utili successivi.

Analogo procedimento se l'elemento scelto viene dopo (greater) di quello desiderato, con la differenza che il nuovo intervallo da considerare si baserà sugli elementi precedenti.

# Ricerca binaria o dicotomica



```
typedef struct
{
    char* cognome;
    char* nome;
    char* telefono;
} TInfo;

bool equal(char* cog, char* nom, char* b0, char* b1) /* a e b sono equivalenti */
{
    return strcmp(nom, b1)==0 &&
        strcmp(cog, b0)==0;
}

bool less(char* cog, char* nom, char* b0, char* b1) /* a precede b */
{
    int conf_cognomi;
    conf_cognomi=strcmp(cog, b0); /* confronta stringhe */
    if (conf_cognomi<0)
        return true;
    else if (conf_cognomi==0)
        return strcmp(nom, b1)<0;
    else /* conf_cognomi > 0 */
        return false;
}

/* Ricerca dicotomica.
 * Restituisce l'indice di un elemento di a che sia equal a x,
 * o un valore negativo se l'elemento non viene trovato.
 * PRE: l'array struct a e' ordinato per cognome - nome rispetto alla relazione less
 */
int binary_search(TInfo a[MAXT], char* x0, char* x1)
{
    int first=0;
    int last=MAXT-1;
    int chosen=(first+last)/2;

    printf("stai cercando %s %s\n\n", x0, x1);

    while (first <= last)
    {
        printf("--- first %d --- chosen %d --- last %d", first, chosen, last);
        getchar();
        if (equal(a[chosen].cognome, a[chosen].nome, x0, x1))
        {
            printf("%s %s (%d) --- EQUAL --- %s %s (11)", a[chosen].cognome,
                a[chosen].nome, chosen, x0, x1);
            return chosen;
        }
        if (less(a[chosen].cognome, a[chosen].nome, x0, x1))
            last = chosen - 1;
        else
            first = chosen + 1;
    }
    return -1;
}
```

BINARY\_SEARCH.c -> IL LISTATO IN C CHE IMPLEMENTA L'ALGORITMO DI RICERCA BINARIA DEL NUMERO DI TELEFONO DI UN NOMINATIVO (cognome, nome) AVUTO IN INPUT, SE PRESENTE IN UN ELENCO TELEFONICO (cognome, nome, numero).

Utilizzo la funzione strcmp (string compare) che restituisce :

- un valore negativo se il primo elemento di comparazione è più piccolo == viene prima
- zero se sono identici
- un valore positivo se il primo elemento è più grande == viene dopo

# Ricerca binaria RICORSIVA

```
binary_search_recursive.c
* Utilizzo la funzione strcmp (string compare) che restituisce :
* - un valore negativo se il primo elemento di comparazione è più piccolo == viene
* - zero se sono identici
* - un valore positivo se il primo elemento è più grande == viene dopo
*/

bool binary_search_recursive(int sinistra, int destra, char* a[], char* s)
{
    bool trovato;
    printf("sinistra %d [%s]", sinistra, a[sinistra]);
    printf("- destra %d [%s]\n\n", destra, a[destra]);
    if (sinistra > destra) // (1)
        //all'inizio sinistra==0 e quindi questo passo cattura anche il caso di vettore
    {
        trovato = false; // NOT_FOUND
        printf("l'animale %s non e' in elenco\n\n", s);
    }
    else
    {
        int m = (destra+sinistra)/2;
        if (strcmp(a[m], s) == 0)
        {
            trovato = true; // (2)
            printf("l'animale %s e' in elenco in posizione %d (indice %d)\n\n", s, m);
        }
        else if (strcmp(a[m], s) < 0)
            trovato = binary_search_recursive(m+1, destra, a, s); // (3)
        else
            trovato = binary_search_recursive(sinistra, m-1, a, s); // (4)
    }
    return trovato;
}
```

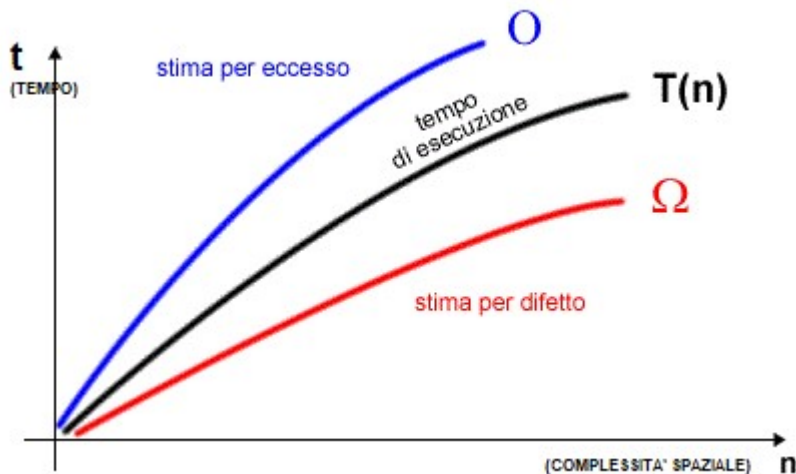
BINARY\_SEARCH\_RECURSIVE.  
c -> IL LISTATO IN C CHE  
IMPLEMENTA  
RICORSIVAMENTE  
L'ALGORITMO DI RICERCA  
BINARIA DI UNA STRINGA IN UN  
VETTORE (CON NOMI DI  
ANIMALI).

Caso base (1) elemento non trovato perché l'incremento del punto mediano ha portato l'elemento inferiore a scavalcare quello superiore. Caso base (2) l'elemento è stato trovato.

Altrimenti si invoca ricorsivamente il metodo: sulla metà superiore dell'array corrente (3) o sulla metà inferiore dell'array corrente (4) a seconda che il punto mediano sia inferiore (venga prima) o superiore (venga dopo) al valore cercato.

Si noti che ogni chiamata ricorsiva eseguita tramite l'invocazione del metodo viene eseguita sul vettore dimezzato e quindi tanto la versione ricorsiva quanto quella iterativa della **ricerca binaria** hanno complessità computazionale  $\Theta(\log 2n)$ , con  $n$  la dimensione della collezione di dati in cui cercare l'elemento, a differenza della **ricerca lineare** che – dovendo percorrere tutti gli elementi di un array – ha complessità  $\Theta(n)$ .

# Complessità computazionale (spazio e tempo)



**Notazione asintotica** per il tasso di crescita (equivalente al concetto matematico di ordine all'infinito della funzione):

**$\Theta$  “Theta”**  $T(n)$  fornisce informazioni più dettagliate sul tasso di crescita della funzione, tanto che per dare una rappresentazione semplice della complessità computazionale (che trascuri i dettagli di scarso interesse, come le costanti moltiplicative o i termini di ordine inferiore) si usano il caso migliore  $T_{\text{best}}$ , peggiore  $T_{\text{worst}}$ , medio  $T_{\text{average}}$

Altre notazioni forniscono un limite lasco, rispettivamente:

- per il tempo di esecuzione massimo  **$O$  “O grande”** (limite superiore asintotico – ovvero una sorta di caso peggiore, uno scenario più pessimistico in termini di consumo di risorse)
- per il tempo di esecuzione minimo  **$\Omega$  “Omega”** (limite inferiore asintotico – ovvero una sorta di caso migliore, uno scenario più ottimistico in termini di consumo di risorse)