

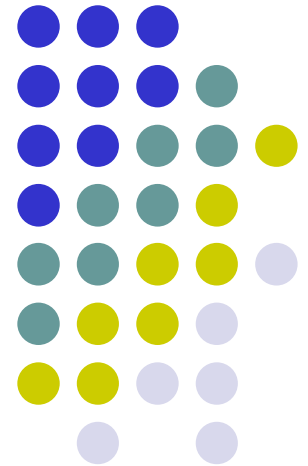
Laboratorio di Calcolatori 1

Corso di Laurea in Fisica

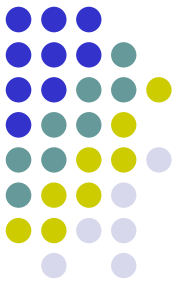
A.A. 2007/2008

Dott. Davide Di Ruscio

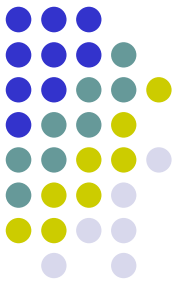
Dipartimento di Informatica
Università degli Studi di L'Aquila



Nota



Questi lucidi sono tratti dal materiale distribuito dalla McGraw-Hill e basati su del materiale fornito dal Prof. Flammini Michele



Sommario (II parte)

Il Linguaggio C

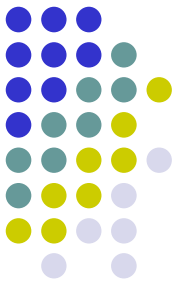
- Caratteristiche generali
- Un linguaggio C semplificato ed esempi di semplici programmi
- Struttura di un programma C
- Direttive del pre-processore
- Parte dichiarativa:
 - tipi
 - definizioni di tipi
 - definizioni di variabili
- Parte esecutiva
 - istruzione di assegnamento
 - istruzioni (funzioni) di input-output
 - istruzioni di selezione
 - istruzioni iterative
- Vettori mono e multidimensionali
- Funzioni e procedure
- File
- Allocazione dinamica di memoria
- Suddivisione dei programmi in piu' file e compilazione separata
- Algoritmi elementari
 - ricerca sequenziale e binaria
 - ordinamento di un vettore: per selezione, per inserimento, per fusione e a bolle
- Aspetti avanzati di programmazione
 - ricorsione
 - strutture dati dinamiche

RIFERIMENTI

Ceri, Mandrioli, Sbattella

[Informatica arte e mestiere](#)

McGraw-Hill



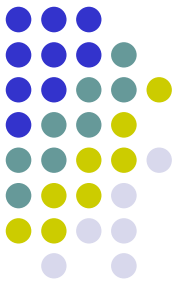
Esercizio

Scrivere un programma in C che, data in ingresso una sequenza di numeri interi terminata da uno 0, il cui primo numero è diverso da 0, stampa in uscita 1 se il primo numero occorre nel resto della sequenza, altrimenti stampa 0

Ad esempio, data in ingresso la sequenza 3 -5 4 3 -10 0, il programma stampa 1 mentre, data la sequenza 2 -5 4 3 -10 0, il programma stampa 0.

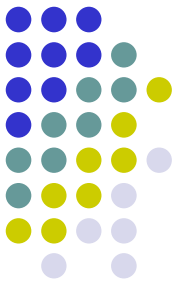
```
#include <stdio.h>
#define lung 20 //Hp. la lunghezza massima della sequenza in ingresso e' 20
main()
{
    int i, j, trovato;
    int a[lung];
    i = 0;
    scanf("%d", &a[i]);
    while (a[i] != 0 && i < lung-1)
    {
        i = i+1;
        scanf("%d", &a[i]);
    }
    j = 1;
    trovato = 0;
    while (j <= i && trovato == 0)
    {
        if (a[0] == a[j]) trovato = 1;
        else j = j+1;
    }
    printf("%d\n", trovato);
}
```

Ricerca del massimo in un array (1/2)



- Leggere un insieme di numeri interi e inserirlo in un array. L'insieme sarà terminato dal numero zero
- Successivamente, trovare:
 - Il numero massimo
 - La posizione del numero massimo

Ricerca del massimo in un array (2/2)

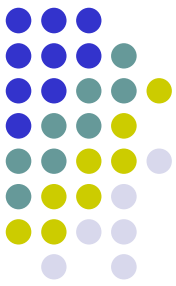


Pseudocodice

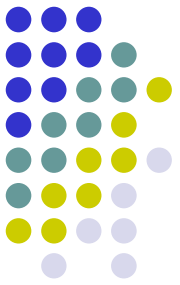
- leggi numero
- if (numero \neq 0 AND contatore numeri è inferiore al massimo permesso dall'array)
 - inserisci il numero nell'array alla posizione data dal contatore
 - incrementa il contatore
 - leggi numero successivo
- il massimo è il numero nella cella zero
- la posizione del massimo è la posizione zero
- for (l'indice i va dalla cella 1 a quella indicata da contatore-1)
 - if (numero cella i-esima è maggiore nel massimo)
 - il massimo è il numero nella cella i-esima
 - la posizione del massimo è la posizione i-esima
- scrivi massimo e posizione del massimo

```
#include <stdio.h>

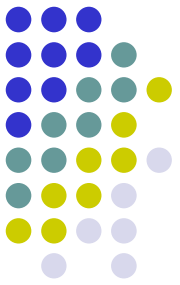
main()
{
    const unsigned int MAX = 1000;
    int arrayNumeri[MAX], dato, max;
    unsigned int i, num = 0, posizMax;
    arrayNumeri[0]=0; /*inizializzo la cella 0 perchè, se non immetto niente*/
        /* nell'array, sarà lei il max */
    scanf ("%d", &dato);
    while (dato != 0 && num < MAX)
    {
        arrayNumeri[num] = dato;
        num = num + 1;
        if (num < MAX) /* non leggo il dato successivo se l'array è finito */
            scanf ("%d", &dato);
    }
    max = arrayNumeri[0];
    posizMax = 0;
    for (i = 1; i < num; i++)
    {
        if (arrayNumeri[i] > max)
        {
            max = arrayNumeri[i];
            posizMax = i;
        }
    }
    printf ("Max:%d, posizione:%d", max, posizMax);
}
```



Ricerca di un elemento in un array

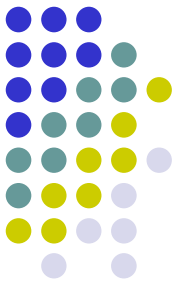


- Vediamo ora due metodi fondamentali per determinare se, dati un array di n interi ed un intero x forniti in input, l'elemento x è presente nell'array, ossia se esiste una componente dell'array avente lo stesso valore di x .
- **Ricerca sequenziale:** a partire dalla prima componente scandisce in sequenza una per una le componenti dell'array alla ricerca di x
- **Ricerca binaria:**
 - Può essere utilizzata solo se le componenti dell'array sono ordinate, ad esempio in modo crescente
 - E' molto più efficiente della ricerca sequenziale perché, sfruttando l'ordinamento delle componenti, dimezza ad ogni passo lo spazio di ricerca, ossia il numero di componenti da controllare



Ricerca sequenziale array

- Dati un array di n interi ed un intero x forniti in input, determina se l'elemento x è presente nell'array, ossia se esiste una componente dell'array avente lo stesso valore di x .
- Idea di risoluzione:
 - Leggiamo da input gli n elementi dell'array tramite un ciclo for.
 - Successivamente, a partire dalla prima componente, scandiamo in sequenza una per una le componenti dell'array alla ricerca di x



Pseudocodice:

- Inizializza *trovato* a 0
- Per i che varia da 0 a $n-1$ incrementando i di 1 ad ogni iterazione
 - Leggi da input il valore da memorizzare nella posizione i dell'array a .
- Leggi x
- Mentre i è minore di n e *trovato* è uguale a 0
 - Se x è uguale ad $a[i]$
 - Poni *trovato* uguale a 1
 - Altrimenti
 - Incrementa i di 1
- Se *trovato* è uguale a 1
 - Stampa "L'elemento è presente nell'array in posizione " i
- Altrimenti
 - Stampa "L'elemento non è presente nell'array"

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

```
main()
```

```
{
```

```
    const int n=50;
```

```
    int a[n], trovato, i, x;
```

```
    trovato = 0;
```

```
    for (i=0; i<n; i=i+1)
```

```
        scanf("%d",&a[i]);
```

```
    scanf("%d",&x);
```

```
    i=0;
```

```
    while (i < n && trovato == 0)
```

```
        if (x==a[i])
```

```
            trovato=1;
```

```
        else
```

```
            i = i + 1;
```

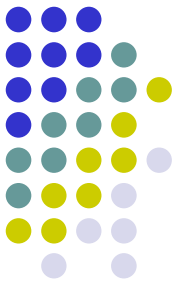
```
    if (trovato==1)
```

```
        printf("L'elemento è presente in posizione %d", i);
```

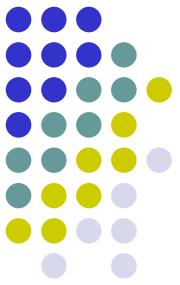
```
    else
```

```
        printf("L'elemento non è presente");
```

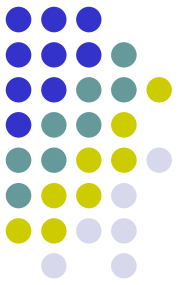
```
}
```



Ricerca binaria in un array



- Dati un array di n interi ordinato in modo crescente ed un intero x forniti in input, determina se l'elemento x è presente nell'array, ossia se esiste una componente dell'array avente lo stesso valore di x
- Idea di risoluzione:
 - Poiché l'array è ordinato, si confronta x con l'elemento centrale
 - Se sono uguali abbiamo trovato l'elemento, altrimenti proseguiamo nel sottovettore a sinistra o a destra dell'elemento centrale a seconda se x è minore o maggiore di tale elemento, dimezzando così lo spazio di ricerca a metà degli elementi.
 - Ripetendo il procedimento giungiamo ad individuare x , se presente, in un numero al più logaritmico in n (base 2) di iterazioni.



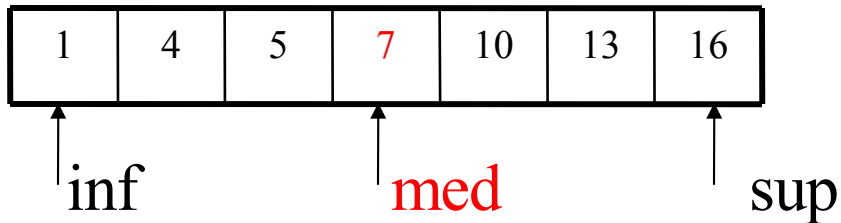
Pseudocodice:

- Inizializza *inf* a 0, *sup* a $n-1$ e *med* alla parte intera inferiore di $(inf+sup)/2$
- Per *i* che varia da 0 a $n-1$ incrementando *i* di 1 ad ogni iterazione
 - Leggi da input il valore da memorizzare nella posizione *i* dell'array a.
- Leggi *x*
- Mentre *inf* è minore o uguale a *sup* e *x* è diverso da $a[med]$
 - Se *x* è maggiore di $a[med]$
 - Poni *inf* uguale a $med+1$
 - Altrimenti
 - Poni *sup* uguale a $med-1$
 - Poni *med* uguale alla parte intera inferiore di $(inf+sup)/2$
- Se *inf* è minore o uguale a *sup*
 - Stampa "L'elemento è presente nell'array in posizione " *med*
- Altrimenti
 - Stampa "L'elemento non è presente nell'array"

Esempio

- Vogliamo cercare l'elemento 5

- **Passo 1**

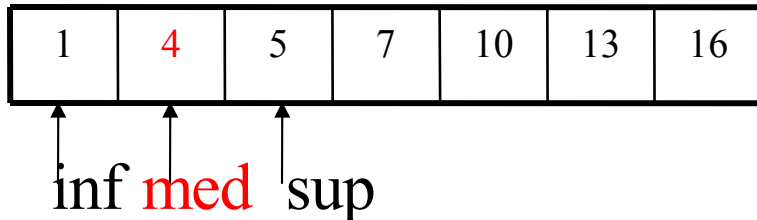


inf = 0
sup = 6
med = 3 ((inf+sup)/2)

$A[\text{med}] == 5$? No

Dato che $A[\text{med}] > 5$, consideriamo il sotto-array da inf a med-1;

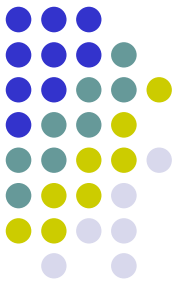
- **Passo 2**

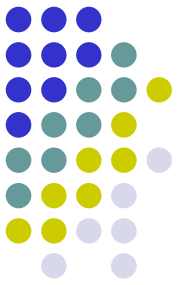


inf = 0
sup = 2
med = 1

$A[\text{med}] == 5$? No

Dato che $A[\text{med}] < 5$, consideriamo il sotto-array da med+1 a sup;





- **Passo 3**

1	4	5	7	10	13	16
---	---	---	---	----	----	----

↑
inf, **med**, sup

inf = 2
sup = 2
med = 2

$A[\text{med}] == 5$? **SI**

Dato che abbiamo trovato l'intero che cerchiamo, la ricerca ha termine

- Poiché ogni volta l'ampiezza della zona da ricercare viene dimezzata, l'algoritmo termina al massimo dopo un numero di confronti necessari a portare $j-i+1$ ad 1, cioè

$$\lceil \log_2(n) \rceil$$


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

```
main()
```

```
{
```

```
    const int n=10;
```

```
    int a[n],inf,med,sup,i,x;
```

```
    for (i=0; i<n; i=i+1)
```

```
        scanf("%d",&a[i]);
```

```
    scanf("%d",&x);
```

```
    inf=0;
```

```
    sup=n-1;
```

```
    med=(inf+sup)/2;
```

```
    while (inf<=sup && x!=a[med])
```

```
    {
```

```
        if (x>a[med])
```

```
            inf = med+1;
```

```
        else
```

```
            sup = med-1;
```

```
        med=(inf+sup)/2;
```

```
    }
```

```
    if (inf<=sup)
```

```
        printf(" L'elemento e' presente in posizione %d", med);
```

```
    else
```

```
        printf(" L'elemento non e' presente");
```

```
}
```

