Alcuni esercizi su algoritmi e programmazione

Fondamenti di Informatica A

Ingegneria Gestionale

Università degli Studi di Brescia

Docente: Prof. Alfonso Gerevini

Esercizio

- Scrivere l'algoritmo e il diagramma di flusso per il seguente problema: l'esecutore deve leggere in ingresso una sequenza di numeri naturali (i.e. interi positivi strettamente maggiori di zero) e calcolarne (per poi stamparli) il **massimo**, il **minimo** e la **media**
- La sequenza si interrompe non appena viene introdotto un numero negativo o uguale a zero
- Per esempio, data la sequenza 5, 1, 2, 3, 4, -5, il risultato dovrebbe essere:

  "Il massimo è 5, il minimo è 1, la media è 3"

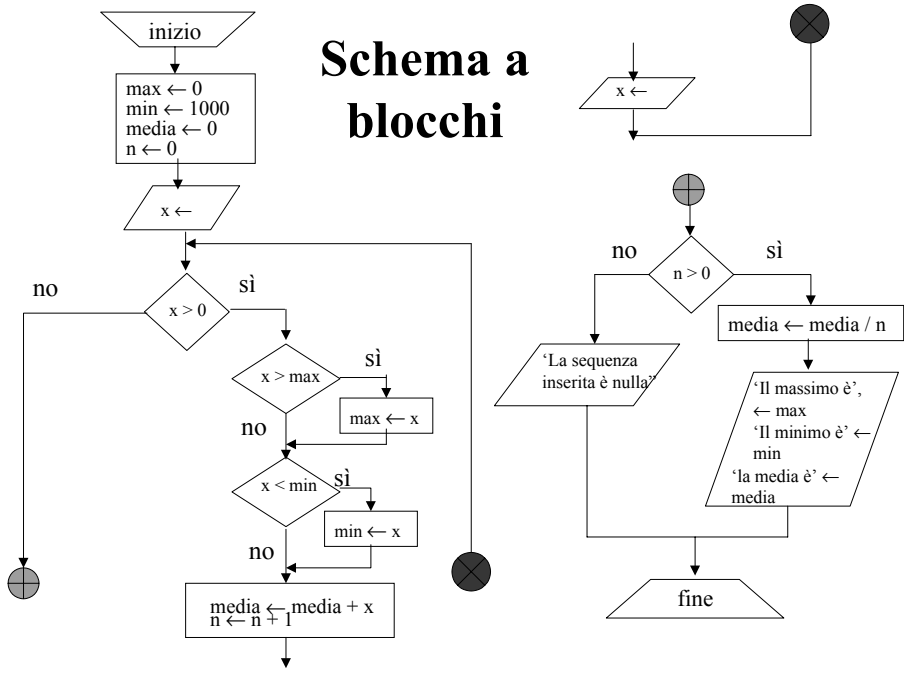
Calcolo massimo, minimo e media

Scomposizione in sottoproblemi:

1. *max* ← 0, *min* ← 1000, *media* ← 0, *n* ← 0
2. Leggi un valore dall'esterno e inseriscilo nella variabile *x*
3. Se *x* > 0 allora vai al passo 4 altrimenti vai al passo 10
4. Se *x* > *max* allora *max* ← *x*
5. Se *x* < *min* allora *min* ← *x*
6. *media* ← *media* + *x*
7. *n* ← *n* + 1
8. Leggi un valore dall'esterno e inseriscilo nella variabile *x*
9. Vai al passo 3
10. Se n > 0 allora vai al passo 11 altrimenti vai al passo 13
11. media ← media / n
12. Stampa "Massimo, minimo, media = " seguita dai valori in *max, min, media* e vai ai al passo 14
13. Stampa "La sequenza inserita è nulla"
14. Fine

Ipotesi: si supponga che il massimo numero rappresentabile sia 1000

Schema a blocchi



Esecuzione passo passo dell'algoritmo: data la sequenza 5, 1, 2, 3, 4, -5

1 Assegna 0 a *max*, 1000 a *min*, 0 a *media* e 0 a *n*
2 Lettura di un numero e memorizzazione nella variabile *x* (*x*=5)
3 Controllo se *x* > 0 ➔ è vero
4 Controllo se *x* > *max* ➔ è vero
5 *max* = 5
6 Controllo se *x* < *min* ➔ è vero
7 *min* = 5
8 *media* = 0 + 5 = 5
9 *n* = 0 + 1 = 1
10 Lettura di un numero e memorizzazione nella variabile *x* (*x*=1)
11 Controllo se *x* > 0 ➔ è vero
12 Controllo se *x* > *max* ➔ non è vero
13 Controllo se x < *min* ➔ è vero
14 *min* = 1

Esecuzione passo passo dell'algoritmo: data la sequenza 5, 1, 2, 3, 4, -5

15 *media* = 5 + 1 = 6
16 *n* = 1 + 1 = 2
17 Lettura di un numero e memorizzazione nella variabile *x* (*x*=2)
18 Controllo se *x* > 0 ➔ è vero
19 Controllo se *x* > *max* ➔ non è vero
20 Controllo se x < *min* ➔ non è vero
21 *media* = 6 + 2 = 8
22 *n* = 2 + 1 = 3
23 Lettura di un numero e memorizzazione nella variabile *x* (*x*=3)
24 Controllo se *x* > 0 ➔ è vero
25 Controllo se *x* > *max* ➔ non è vero
26 Controllo se x < *min* ➔ non è vero
27 *media* = 8 + 3 = 11
28 *n* = 3 + 1 = 4

Esecuzione passo passo dell'algoritmo: data la sequenza 5, 1, 2, 3, 4, -5

29 Lettura di un numero e memorizzazione nella variabile *x* (*x* = 4)
30 Controllo se *x* > 0 ➔ è vero
31 Controllo se *x* > *max* ➔ non è vero
32 Controllo se *x* < *min* ➔ non è vero
33 *media* = 11 + 4 = 15
34 *n* = 4 + 1 = 5
35 Lettura di un numero e memorizzazione nella variabile *x* (*x* = -5)
36 Controllo se *x* > 0 ➔ non è vero
37 Controllo se *n* > 0 ➔ è vero
38 *media* = 15 / 5 = 3
39 Visualizza "il Massimo è" 5
40 Visualizza "Il minimo è" 1
41 Visualizza "La media è" 3

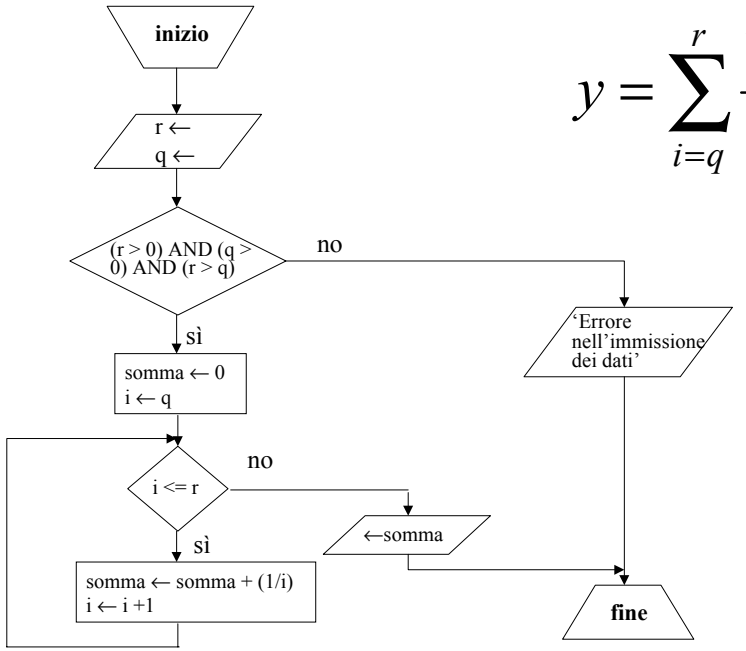
Esecuzione passo passo dell'algoritmo: data la sequenza –2 (cioè sequenza nulla)

1 Assegna 0 a *max*, 1000 a *min*, 0 a *media* e 0 a *n*
2 Lettura di un numero e memorizzazione nella variabile *x* (*x*=-2)
3 Controllo se *x* > 0 ➔ non è vero
4 Controllo se n > 0 ➔ non è vero
5 Visualizza 'La sequenza inserita è nulla'

Esercizio

- Sia $y = \sum_{i=q}^{r} \frac{1}{i}$

- Scrivere il diagramma di flusso per il calcolo di *y* assumendo di acquisire *r* e *q* dall'esterno (dati di input)
- Nota: controllare che *r* e *q* siano interi positivi tali che *r* > *q*



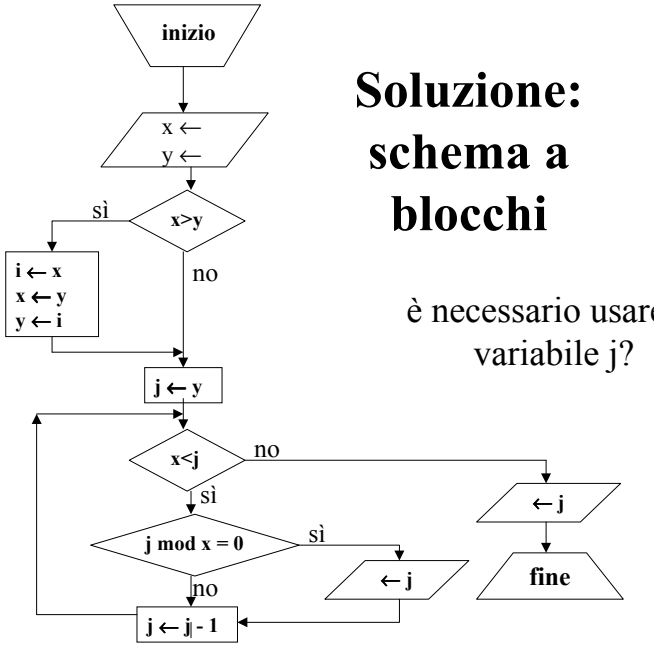
$$y = \sum_{i=q}^{r} \frac{1}{i}$$

Esecuzione passo passo dell'algoritmo

1 Lettura di un numero e memorizzazione nella variabile r (r=6)
2 Lettura di un numero e memorizzazione nella variabile q (q=4)
3 Controllo se (r>0) AND (q > 0) AND (r>q) ➔ è vero
4 Somma = 0
5 I = 4
6 Controllo se 4 <= 6 ➔ è vero
7 Somma = 0 + 1/4 = 0,25
8 I = 4 + 1 = 5
9 Controllo se 5 <= 6 ➔ è vero
10 Somma = 0,25 + 1/5 = 0,45
11 I = 5 + 1 = 6
12 Controllo se 6 <= 6 ➔ è vero
13 Somma = 0,45 + 1/6 = 0,6166666….
14 I = 6 + 1 = 7
15 Controllo se 7 <= 6 ➔ non è vero
16 Visualizza 0,6166666….

Esercizio

- Scrivere un diagramma di flusso *strutturato* che, dati in ingresso due numeri positivi x e y visualizza in ordine *de*crescente la sequenza di numeri interi compresi tra x e y che sono *divisibili per per il minore tra x e y.* Ad esempio, se x = 7 e y = 35, la sequenza è 35 28 21 14 7
- Scrivere il corrispondente programma in C

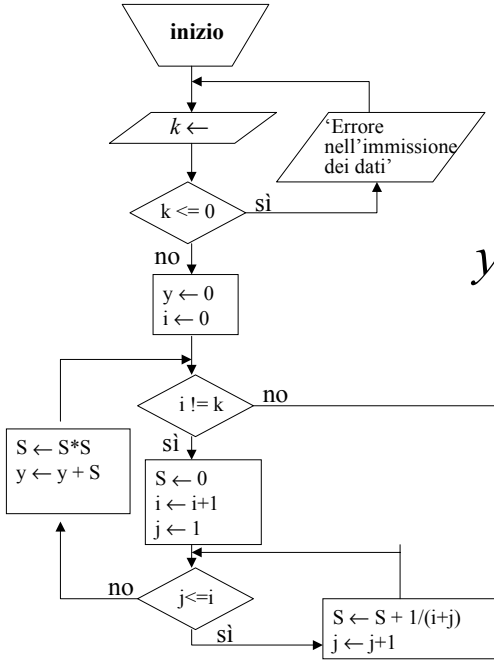


Soluzione: schema a blocchi

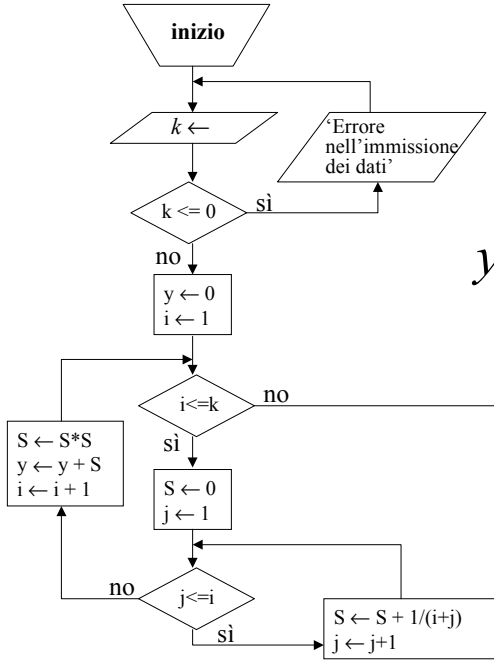
è necessario usare la variabile j?

Esercizio

- Sia $y = \sum_{i=1}^{k} (\sum_{j=1}^{i} \frac{1}{i+j})^2$
- Scrivere il diagramma di flusso per il calcolo di y assumendo di acquisire k dall'esterno (dato di input) controllando che k sia intero positivo
- Scrivere il corrispondente programma in C e in Basic



$$y = \sum_{i=1}^{k} (\sum_{j=1}^{i} \frac{1}{i+j})^2$$



Una soluzione alternativa?

$$y = \sum_{i=1}^{k} (\sum_{j=1}^{i} \frac{1}{i+j})^2$$

Traduzione in C

```
main() /* in C */
{
 int k, i, j;
 float y, s;
 do
    { scanf("%d", &k);
      if (k <= 0) printf("Errore nell'immissione di k");
    }
 while (k<=0)
 y = 0; i = 1;
 while (i <= k)
 {
  s = 0; j = 1;
  while (j <= i)
  { s = s + 1/(i+j);
    j = j + 1; }
  s = s * s;
  y = y + s;
  i = i + 1;
 }
 printf("%d", y)
}
```

Ciclo di acquisizione di k

Inizializzazioni per ciclo esterno

Inizializzazioni per ciclo interno

Sommatoria interna

Sommatoria esterna

Traduzione in Basic

```
' in BASIC
 dim i as integer, j as integer, k as integer
 dim y as single,  s as single
 do
    input k
    if k <= 0 then print "Errore nell'immissione di k"
    end if
 loop while k<=0
 y = 0: i = 1
 while i <= k
   s = 0: j = 1
  while j <= i
    s = s + 1/(i+j)
    j = j + 1
  wend
  s = s * s
  y = y + s
  i = i + 1
 wend
 print  y
```

Esercizio:
fare l'esecuzione passo passo del programma

Esercizio: ricerca elemento in un vettore

- Si supponga di avere *n* numeri memorizzati in un vettore di *n* elementi
- Scrivere un algoritmo (in pseudo-codice o con uno schema a blocchi) che, preso in ingresso un numero, ***ricerca tale numero nel vettore*** e se esiste risponde "ho trovato il numero", altrimenti risponde "il numero non è nel vettore"
- Scrivere il corrispondente programma in C e in Basic (nota: per la stampa di stringhe usare printf, es. printf("ho trovato il numero") e print es. print "ho trovato il numero"

Ricerca di un elemento in un vettore

Dati
  n=100 intero
  a[ ] vettore di interi
  i, t, num interi positivi
Risoluzione
  leggi num
  i ← 1
  t = 0
  finchè (i <= n) ripeti
     se a[i] = num allora t = 1
       i ← i + 1
  fine ciclo
  se t = 1 allora scrivi "trovato"
       altrimenti scrivi "non trovato"

```
main()    /* C  */
{
  int a[100];
  int i, t, num;
  scanf("%d",&num);
  i = 0;
  t = 0;
  while (i < 100)
  {  if (a[i] == num) t = 1;
     i = i + 1;
  }
  if (t == 1) printf ("Trovato");
     else printf("Non trovato");
}
```

E'meglio di un ciclo FOR? Perché?

Ricerca di un elemento in un vettore

Dati
n=100 intero
a[ ] vettore di interi
i, t, num interi positivi

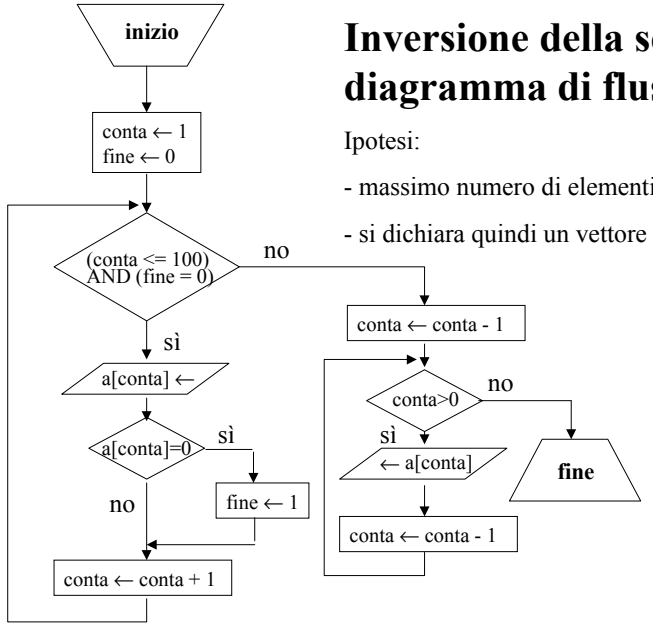
Risoluzione
leggi num
i ← 1
t = 0
finchè (i <= n) ripeti
se a[i] = num allora t = 1
i ← i + 1
fine ciclo
se t = 1 allora scrivi "trovato"
altrimenti scrivi "non trovato"

Ciclo inefficiente!
Idee?

```
' in Basic
dim a(100) as integer
dim i as integer, t as integer
dim num as integer
input num
i = 1
t = 0
while i <= 100
  if  a(i) = num then t = 1
  end if
  i = i + 1
wend
if t = 1 then print "Trovato"
  else print "Non trovato"
end if
```


Esercizio

- Si consideri il problema di ***leggere una sequenza arbitraria di numeri in ingresso e riscriverla in ordine inverso***
- Risolvere il problema usando i vettori
- Si supponga che la fine della sequenza sia indicata dal numero 0
- Scrivere il diagramma di flusso
- Scrivere i corrispondenti programmi in C e in Basic



Inversione della sequenza: diagramma di flusso

Ipotesi:

- massimo numero di elementi acquisibili = 100

- si dichiara quindi un vettore a[100]

Programma 'inverti-sequenza'

```
main()     /* C  */
{
  int a[100];
  int conta, fine;
  conta = 0; fine = 0;
  while ((conta < 100) &&
       (fine==0))
  {   scanf("%d",&a[conta]);
      if (a[conta] == 0) fine = 1;
      conta = conta + 1; }
  conta = conta - 1;
  while (conta >= 0)
  { printf("%d",a[conta]);
    conta = conta – 1; }
}
```

```
' Basic
dim a(100) as integer
dim conta as integer, fine as
      integer
conta = 1: fine = 0
while conta <= 100 AND fine=0
     input a(conta)
     if a(conta) = 0 then fine = 1
     end if
     conta = conta + 1
wend
conta = conta - 1
while conta > 0
  print a(conta)
  conta = conta – 1
wend
```

Esercizio: riscrivere come ciclo FOR

Esercizio

- Si supponga di avere *n* numeri memorizzati in un vettore di *n* elementi
- Scrivere un algoritmo per l'ordinamento in ordine crescente di tali numeri
- Esempio:
  - Vettore: a[1]=4, a[2]=5, a[3]=1, a[4]=2, a[5]=8

    applicando l'algoritmo avrò il vettore a trasformato nel seguente modo
  - Vettore: a[1]=1, a[2]=2, a[3]=4, a[4]=5, a[5]=8
- L'algoritmo deve essere generico, ovvero deve poter funzionare per un generico numero *n* di elementi
- (Per la soluzione si veda il file "*Algoritmi di ordinamento*")

Esercizio

- Riscrivere l'algoritmo di ***ordinamento per inserimento diretto*** nel seguente caso:
  - I numeri da ordinare vengono acquisiti dall'esterno uno ad uno finché non viene fornito il numero 0
  - Supporre che il vettore in cui si vanno a memorizzare i numeri sia di 100 posizioni e sia inizialmente vuoto
  - Fare in modo che durante la memorizzazione nel vettore i numeri acquisiti vengano inseriti nella giusta posizione, ovvero che alla fine dell'acquisizione i numeri risultino già ordinati

Esercizio

- Data la seguente sequenza di numeri

  3 11 2 5 9

E se al posto del valore 9 ci fosse 4 ?

- Indicare **la sequenza di scambi** di effettuata dall'algoritmo di ordinamento
  - **per inserimento** 11→2 3→11 2→3 11↔5 11↔9
  - **per selezione** 3↔2 11↔3 11↔5 11↔9
  - **a bolle** 11↔2 3↔2 11↔5 11↔9

Esercizio

- Modificare l'algoritmo ed i programmi in C e Basic per la ***ricerca di un elemento in un vettore*** in modo da renderlo più efficiente:

  si deve uscire dal ciclo quando il numero degli elementi esaminati è esaurito oppure quando l'elemento è stato trovato

Esercizio

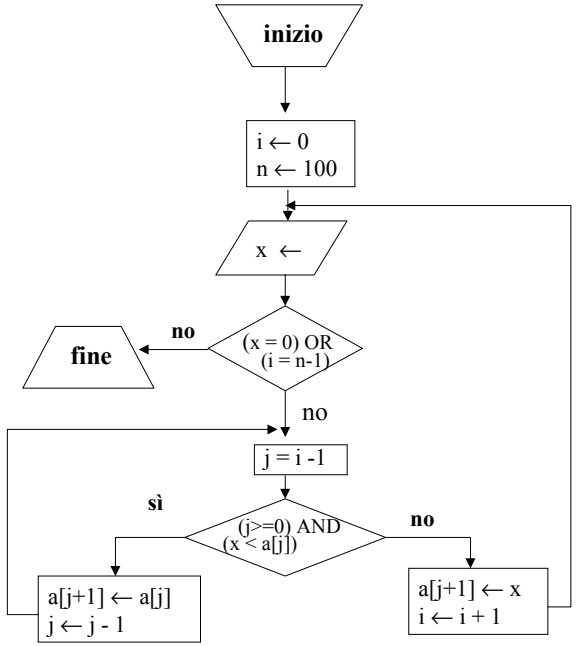
- Scrivere un programma che dato in ingresso un numero *n* fornisce in uscita le ***radici quadrate intere*** dei numeri compresi fra 1 e *n*
- In particolare scrivere un ***sottoprogramma*** che dato in ingresso un certo numero *i* produca la radice quadrata intera di *i*
- Chiamare il sottoprogramma all'interno del programma
- **Calcolo della radice quadrata intera di *i*:**

  Fare un ciclo su *j* a partire da *j*= 1 verificando ogni volta se

  *j*j* <=*i*, quando *j*j* non è più minore o uguale di *i* allora è possibile restituire la radice intera di *i* che vale *j-1*.

  Esempio: i = 7

  Provo con 0 … 0 x 0 = 0 < 7 prosegui
  Provo con 1 … 1 x 1 = 1 < 7 prosegui
  Provo con 2 … 2 x 2 = 4 < 7 prosegui
  Provo con 3 … 3 x 3 = 9 < 7 no, allora il risultato è 3 – 1 = 2 !!!



Inserimento diretto con vettore inizialmente vuoto

Ipotesi: supponiamo di considerare un vettore di 100 interi (il numero di elementi acquisiti può essere al massimo 100)

Come funziona?

- Supponiamo all'inizio di avere il vettore a seguente

a	0	0	0	0	0	0	…	0	0	0	0
	0	1	2	3	4	5		96	97	98	99

1 i = 0, n = 100
2 Leggo x → ***supponiamo di leggere il numero 44***
3 Controllo se ((x = 0) OR (i = 99)) → non è vero
4 j = i – 1 = -1
5 Controllo se ((j >= 0) AND (x < a[-1])) → non è vero
6 a[j+1] = a[0] ← x = 44

a	44	0	0	0	0	…

Esecuzione passo passo

7 i = i + 1 = 1
8 Leggo x → ***supponiamo di leggere il numero 55***
9 Controllo se ((x = 0) OR (i = 99)) → non è vero
10 j = i – 1 = 0
11 Controllo se ((j >= 0) AND (x < a[0])) → non è vero (x contiene 55 e a[0]=44)
12 a[j+1] = a[1] ← x = 55
13 i = i + 1 = 2
14 Leggo x → ***supponiamo di leggere il numero 12***
15 Controllo se ((x = 0) OR (i = 99)) → non è vero
16 j = i – 1 = 1
17 Controllo se ((j >= 0) AND (x < a[1])) → è vero (x contiene 12 e a[1]=55)
18 a[j+1] = a[j] = 55

a	44	55	0	0	0	…

a	44	55	55	0	0	…

Esecuzione passo passo

19 j = j - 1 = 0
20 Controllo se ((j >= 0) AND (x < a[0])) ➔ è vero (x contiene 12 e a[0]=44)
21 a[j+1] = a[j] = 44
22 j = j - 1 = -1
23 Controllo se ((j >= 0) AND (x < a[-1])) ➔ non è vero
24 a[j+1] = x = 12
25 i = i + 1 = 3
26 Leggo x ➔ ***supponiamo di leggere il numero 42***
27 Controllo se ((x = 0) OR (i = 99)) ➔ non è vero
28 j = i – 1 = 2
29 Controllo se ((j >= 0) AND (x < a[2])) ➔ è vero (x contiene 42 e a[2]=55)
30 a[j+1] = a[j] = 55

a

44	44	55	0	0	…

a

12	44	55	0	0	…

a

12	44	55	55	0	…

Esecuzione passo passo

31 j = j - 1 = 1
32 Controllo se ((j >= 0) AND (x < a[1])) ➔ è vero (x contiene 42 e a[1]=44)
33 a[j+1] = a[j] = 44
34 j = j - 1 = 0
35 Controllo se ((j >= 0) AND (x < a[0])) ➔ non è vero (x contiene 42 e a[0]=12)
36 a[j+1] = x = 42
37 i = i + 1 = 4
38 Leggo x ➔ ***supponiamo di leggere il 0***
39 Controllo se ((x = 0) OR (i = 99)) ➔ è vero
40 Fine ➔ ***il vettore è ordinato***

a

12	44	44	55	0	…

a

12	42	44	55	0	…

Ricerca di un elemento in un vettore

```
main()      /* C  */
{
  int a[100];
  int i, num;
  bool t;
  scanf("%d",&num);
  i = 0;
  t = 0;
  while ((i < 100) && (!t))
  {  if (a[i] == num) t = 1;
     i = i + 1;
  }
  if (t) printf ("Trovato");
     else printf("Non trovato");
}
```

```
' in Basic
  dim a(100) as integer
  dim i as integer, t as integer
  dim num as integer
  input num
  i = 1
  t = 0
  while i <= 100 AND not t
    if  a(i) = num then t = -1
    end if
    i = i + 1
  wend
  if t then print "Trovato"
    else print "Non trovato"
  end if
```

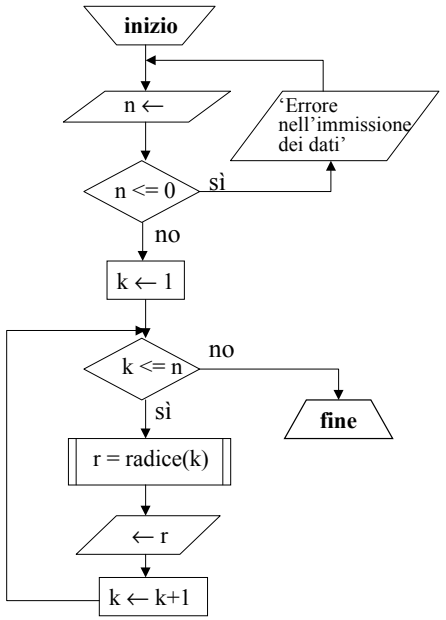
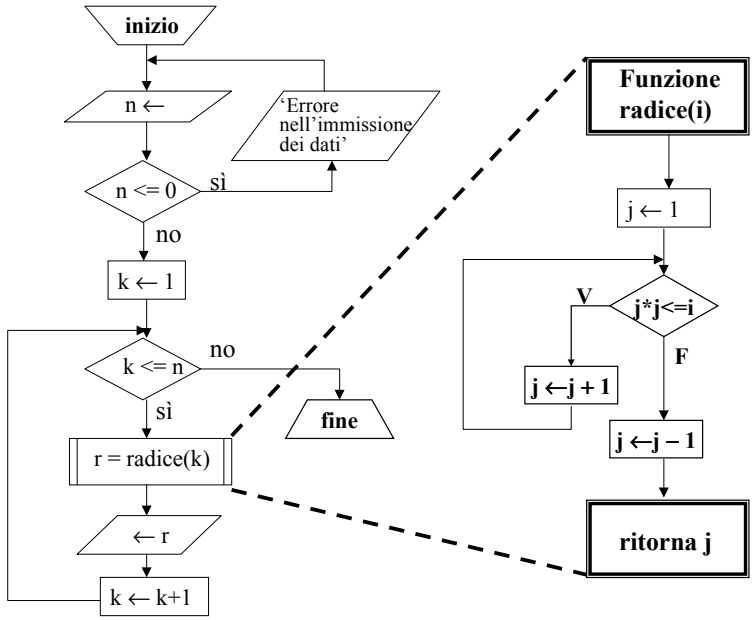



Diagramma di flusso
per il calcolo delle radici intere di k (k=1..n)



Programma per il calcolo delle radici

```
/* Radici in C */
int n, k, r;
main()
{
 do
 {
  scanf("%d", &n);
  if (n <= 0) printf("Errore
      nell'immissione di n");
 } while (n<=0)

 for (k=1;k<=n;k++)
 {
  r = radice(k);
  printf("%d", r);
 }
}
```

```
' in BASIC
 dim n as integer, k as integer, r as integer
 do
    input n
    if n <= 0 then print "Errore
                  nell'immissione di n"
    end if
 loop while n<=0

 for k=1 to n
          r = radice(k)
          print  r
next k
```


La funzione 'Radice'

Funzione radice(i)
Dati
  j intero
Risoluzione
  j ← 1
  finchè (j*j <= i) ripeti
      j ← j + 1
  fine ciclo
  restituisci j-1
fine funzione

```
int radice(int i)
{
  int j;

  j = 1;
  while (j*j <= i)
       j = j + 1;

  return (j-1);
}
```

```
function radice(i as integer)

dim j as integer

j = 1
while j*j <= i
      j = j + 1
wend
radice = j - 1
end function
```


Esecuzione passo passo del programma

Supponiamo che l'utente abbia inserito il valore n = 5

1 k = 1
2 Controllo se k <= 5 è vero
3 Chiamata della funzione **radice(1)**
4 j = 1
5 Controllo se 1*1 <= 1 è vero
6 j = j + 1 = 2
7 Controllo se 2*2 <= 1 non è vero
8 Ritorna il valore **1** al programma chiamante
9 Stampa 1
10 k = k + 1 = 2

Esecuzione del sottoprogramma con parametro i =1

Esecuzione passo passo del programma

11 Controllo se k <= 5 è vero
12 Chiamata della funzione **radice(2)**
13 j = 1
14 Controllo se 1*1 <= **2** è vero
15 j = j + 1 = 2
16 Controllo se 2*2 <= **2** non è vero
17 Ritorna il valore **1** al programma chiamante

Esecuzione del sottoprogramma con parametro i =2

18 Stampa 1
19 k = k + 1 = 3
20 Controllo se k <= 5 è vero

Esecuzione passo passo della funzione 'radice' (cont…)

21 Chiamata della funzione **radice(3)**
22 j = 1
23 Controllo se 1*1 <= **3** è vero
24 j = j + 1 = 2
25 Controllo se 2*2 <= **3** non è vero
26 Ritorna il valore **1** al programma chiamante

Esecuzione del sottoprogramma con parametro i =3

27 Stampa 1
28 k = k + 1 = 4
29 Controllo se k <= 5 è vero

Esecuzione passo passo della funzione 'radice' (cont…)

30 Chiamata della funzione **radice(4)**
31 j = 1
32 Controllo se 1*1 <= **4** è vero
33 j = j + 1 = 2
34 Controllo se 2*2 <= **4** è vero
35 j = j + 1 = 3
36 Controllo se 3*3 <= **4** non è vero
37 Ritorna il valore **2** al programma chiamante

Esecuzione del sottoprogramma con parametro i = 4

38 Stampa 2
39 k = k + 1 = 5
40 Controllo se k <= 5 è vero

Esecuzione passo passo della funzione 'radice' (cont…)

41 Chiamata della funzione **radice(5)**
42 j = 1
43 Controllo se 1*1 <= **5** è vero
44 j = j + 1 = 2
45 Controllo se 2*2 <= **5** è vero
46 j = j + 1 = 3
47 Controllo se 3*3 <= **5** non è vero
48 Ritorna il valore **2** al programma chiamante

Esecuzione del sottoprogramma con parametro i =5

49 Stampa 2
50 k = k + 1 = 6
51 Controllo se k <= 5 non è vero
52 Fine