Esempio: Leggere due interi positivi e calcolarne il massimo comun divisore.

$MCD(12, 8) = 4$
$MCD(12, 6) = 6$
$MCD(12, 7) = 1$

- Sfruttando direttamente la definizione di MCD
  - osservazione: $1 \leq MCD(m,n) \leq min(m,n)$
    $\Longrightarrow$ si provano i numeri compresi tra 1 e $min(m,n)$
  - conviene iniziare da $min(m,n)$ e scendere verso 1

Algoritmo: stampa MCD di due interi positivi letti da tastiera

```
leggi m ed n
inizializza mcd al minimo tra m ed n
while mcd > 1 e non si è trovato un divisore comune
{
  if mcd divide sia m che n
    si è trovato un divisore comune
  else decrementa mcd di 1
}
stampa mcd
```

Osservazioni

- il ciclo termina sempre perché ad ogni iterazione
  - o si è trovato un divisore
  - o si decrementa mcd di 1 (al più si arriva a 1)
- per verificare se si è trovato il MCD si utilizza una variabile booleana (nella guardia del ciclo)
- Implementazione in C ...

```
int m, n;              /* i due numeri letti */
int mcd;              /* il massimo comun divisore */
int trovato = 0;      /* var. booleana:  inizialmente false */
if (m <= n)           /*inizializza mcd al minimo tra m e n*/
  mcd = m;
else
  mcd = n;
while (mcd > 1 && !trovato)
  if ((m % mcd == 0) && (n % mcd == 0))
    /* mcd divide entrambi */
    trovato = 1;
  else
    mcd = mcd -1;
printf("MCD di %d e %d:  %d", m, n, mcd);
```


Quante volte viene eseguito il ciclo?

- caso migliore: 1 volta (quando m divide n o viceversa)
  es. MCD(500, 1000)
- caso peggiore: *min*(m,n) volte (quando MCD(m,n)=1)
  es. MCD(500, 1001)
- l'algoritmo si comporta *male* se m e n sono grandi e MCD(m,n) è piccolo

Metodo di Euclide per il calcolo del MCD

- Già visto nell'introduzione (pseudo-linguaggio). Permette di ridursi più velocemente a numeri più piccoli, sfruttando le seguenti proprietà:
  MCD(x, x) = x
  MCD(x, y) = MCD(x-y, y) se x>y
  MCD(x, y) = MCD(x, y-x) se y>x
- I divisori comuni di m ed n, (m>n), dividono anche m-n.
  Es.: MCD(12, 8) = MCD(12-8, 8) = MCD(4, 8-4) = 4
- Come si ottiene un algoritmo? Si applica ripetutamente il procedimento fino a che m=n.

Esempio:

m	n	maggiore - minore
210	63	147
147	63	84
84	63	21
21	63	42
21	42	21
21	21	

Algoritmo: di Euclide per il calcolo del MCD

```
int m,n;
scanf("%d%d", &m, &n);
while (m != n)
  if (m > n)
    m = m - n;
  else
    n = n - m;

printf("MCD: %d\n", m);
```

- Cosa succede se m=n=0?

  $\Longrightarrow$ il risultato è 0
- E se m=0 e n$\neq$ 0 (o viceversa)?

  $\Longrightarrow$ si entra in un ciclo infinito

- Per assicurarci che l'algoritmo venga eseguito su valori corretti, possiamo inserire una verifica sui dati in ingresso, attraverso un ciclo di lettura

  Proposte?

```
do {
  printf("Immettere due interi positivi:  ");
  scanf("%d%d", &m, &n);
  if (m <= 0 || n <= 0)
    printf("Errore:  i numeri devono essere > 0!\n");
} while (m <= 0 || n<= 0);
```


Metodo di Euclide con i resti per il calcolo del MCD

- Cosa succede se $m \gg n$?

Esempio:

MCD(1000, 2)		MCD(1001, 500)	
1000	2	1001	500
998	2	501	500
996	2	1	500
...		...	
2	2	1	1

- Come possiamo comprimere questa lunga sequenza di sottrazioni?
- Metodo di Euclide: sia

$$m = n \cdot k + r \quad (\text{con } 0 \leq r < m)$$

$$\begin{array}{lcll} MCD(m, n) & = & n & \text{se } r=0 \\ MCD(m, n) & = & MCD(r, n) & \text{se } r \neq 0 \end{array}$$

Algoritmo di Euclide per il calcolo del MCD

```
leggi m ed n
while m ed n sono entrambi ≠ 0
{ sostituisci il maggiore tra m ed n con
  il resto della divisione del maggiore per il minore
}
stampa il numero tra i due che è diverso da 0
```

Esercizio
Tradurre l'algoritmo in C

Digressione sulle costanti: la direttiva #define

- La dichiarazione di costante, ad esempio
  ```
  const int Nmax = 10;
  ```
  causa l'allocazione di memoria (si tratta di una dichiarazione di variabile *read only*)
- C'è un altro modo per ottenere un identificatore costante, che utilizza la direttiva **#define**.
  ```
  #define Nmax 10
  ```
- **#define** è una direttiva di compilazione
- dice al compilatore di sostituire ogni occorrenza di `Nmax` con `10` prima di compilare il programma
- a differenza di **const** non alloca memoria

Assegnamento e altri operatori

- In C, l'operazione di assegnamento `x = exp` è un'espressione
  - il valore dell'espressione è il valore di `exp` (che è a sua volta un'espressione)
  - la valutazione dell'espressione `x = exp` ha un side-effect: quello di assegnare alla variabile `x` il valore di `exp`
- Dunque in realtà, "=" è un operatore (associativo a destra).
  Esempio: Qual è l'effetto di `x = y = 4` ?
  - È equivalente a: `x = (y = 4)`
  - `y = 4` ... espressione di valore `4` con modifica (side-effect) di `y`
  - `x = (y = 4)` ... espressione di valore `4` con ulteriore modifica su `x`
- L'eccessivo uso di assegnamenti come espressioni rende il codice difficile da comprendere e quindi anche da correggere/modificare.

Operatori di incremento e decremento

- Assegnamenti del tipo: `i = i + 1` / `i = i - 1` sono molto comuni.
  - operatore di incremento: `++`
  - operatore di decremento: `--`
- In realtà `++` corrisponde a due operatori:
- postincremento: `i++`
  - il valore dell'espressione è il valore di `i`
  - side-effect: incrementa `i` di `1`
- L'effetto di

```
int i,j;
i=6;
j=i++;
```

è j=6, i=7.

- preincremento: `++i`
  - il valore dell'espressione è il valore di `i+1`
  - side-effect: incrementa `i` di `1`
- L'effetto di

```
int i,j;
i=6;
j=++i;
```

è j=7, i=7.

(analogamente per `i--` e `--i`)

- Nota sull'uso degli operatori di incremento e decremento

Esempio:

	Istruzione	x	y	z
1	`int x, y, z;`	?	?	?
2	`x = 4;`	4	?	?
3	`y = 2;`	4	2	?
4a	`z = (x + 1) + y;`	4	2	7
4b	`z = (x++) + y;`	5	2	6
4c	`z = (++x) + y;`	5	2	7

- N.B.: Non usare mai in questo modo!
  In un'istruzione di assegnamento non ci devono essere altri side-effect (oltre a quello dell'operatore di assegnamento) !!!
- Riscrivere, ad esempio, come segue:

4b: `z = (x++) + y;` $\Longrightarrow$

```
z = x + y;
x++;
```

4c: `z = (++x) + y;` $\Longrightarrow$

```
x++;
z = x + y;
```

Ordine di valutazione degli operandi

- In generale il C non stabilisce quale è l'ordine di valutazione degli operandi nelle espressioni. Dipende quindi dal compilatore.

  Esempio:
  ```
  int x, y, z;
  x = 2;
  y = 4;
  z = x++ + (x * y);
  ```
- Quale è il valore di `z`?
  - se viene valutato prima `x++`: $2 + (3 * 4) = 14$
  - se viene valutato prima `x*y`: $(2 * 4) + 2 = 10$

Forme abbreviate dell'assegnamento

```
a = a + b;   ⟹   a += b;
a = a - b;   ⟹   a -= b;
a = a * b;   ⟹   a *= b;
a = a / b;   ⟹   a /= b;
a = a % b;   ⟹   a %= b;
```


Espressioni condizionali

Al posto di
```
int x, y, z;
if (x > y)
z = x;
else
z = y;
```
possiamo scrivere, parafrasando quanto scritto, come:
```
int x, y, z;
z = (x > y)?  x :  y;
```
Abbiamo usato quelle che vengono definite espressioni condizionali, la cui sintassi è:
```
 (espr-1)?  espr-2 :  espr-3
```
A seconda dell'esito della valutazione della prima espressione, si valuta la seconda oppure la terza.

Tipi di dato strutturati: Array

- I tipi di dato visti finora sono tutti semplici: `int`, `char`, `float`, ...
- ma i dati manipolati nelle applicazioni reali sono spesso complessi (o strutturati)
- Gli array sono uno dei tipi di dato strutturati
  - sono composti da elementi omogenei (tutti dello stesso tipo)
  - ogni elemento è identificato all'interno dell'array da un numero d'ordine detto indice dell'elemento
  - il numero di elementi dell'array è detto lunghezza (o dimensione) dell'array
- Consentono di rappresentare tabelle, matrici, matrici n-dimensionali, ...

Array monodimensionali (o vettori)

- Supponiamo di dover rappresentare e manipolare la classifica di un campionato cui partecipano 16 squadre.
- È del tutto naturale pensare ad una tabella

Classifica

Squadra A	Squadra B	...	Squadra C
1^o posto	2^o posto		16^o posto

che si evolve con il procedere del campionato

Classifica

Squadra B	Squadra A	...	Squadra C
1^o posto	2^o posto		16^o posto

Sintassi: dichiarazione di variabile di tipo vettore

```
tipo-elementi nome-array [lunghezza];
```

Esempio: `int vet[6];`
dichiara un vettore di 6 elementi, ciascuno di tipo intero.

- All'atto di questa dichiarazione vengono riservate (allocate) 6 locazioni di memoria consecutive, ciascuna contenente un intero. La lunghezza del vettore è 6.
- La lunghezza di un vettore deve essere costante (nota a tempo di compilazione). Non si può ad esempio chiedere all'utente di immettere il valore della lunghezza.
- Ogni elemento del vettore è una variabile identificata dal nome del vettore e da un indice

  Sintassi: elemento di array `nome-array[espressione];`

  Attenzione: `espressione` deve essere di tipo intero ed il suo valore deve essere compreso tra 0 a lunghezza-1.

- **Esempio:**

indice	elemento	variabile
0	?	`vet[0]`
1	?	`vet[1]`
2	?	`vet[2]`
3	?	`vet[3]`
4	?	`vet[4]`
5	?	`vet[5]`

- `vet[i]` è l'elemento del vettore `vet` di indice `i`.
  Ogni elemento del vettore è una variabile.

```
int vet[6], a;
vet[0] = 15;
a = vet[0];
vet[1] = vet[0] + a;
printf("%d", vet[0] + vet[1]);
```

- `vet[0]`, `vet[1]`, ecc. sono variabili intere come tutte le altre e dunque possono stare a sinistra dell'assegnamento (es. `vet[0] = 15`), come all'interno di espressioni (es. `vet[0] + a`).
- Come detto, l'indice del vettore è un'espressione.

```
index = 2;
vet[index+1] = 23;
```

Manipolazione di vettori

- avviene solitamente attraverso cicli **for**
- l'indice del ciclo varia in genere da 0 a lunghezza-1
- spesso conviene definire la lunghezza come una costante attraverso la direttiva `#define`

Esempio: Lettura e stampa di un vettore.

```
#include <stdio.h>
#define LUNG 5

main ()
{
int v[LUNG]; /* vettore di LUNG elementi, indicizzati da 0 a LUNG-1 */
int i;

for (i = 0; i < LUNG; i++) {
  printf("Inserisci l'elemento di indice %d:  ", i);
  scanf("%d", &v[i]);
}
printf("Indice Elemento\n");
for (i = 0; i < LUNG; i++) {
  printf("%6d %8d\n", i, v[i]); }
}
```


Inizializzazione di vettori

- Gli elementi del vettore possono essere inizializzati con valori costanti (valutabili a tempo di compilazione) contestualmente alla dichiarazione del vettore.

  Esempio: `int n[4] = {11, 22, 33, 44};`

- l'inizializzazione deve essere contestuale alla dichiarazione

  Esempio:
  ```
  int n[4];
  n = {11, 22, 33, 44};
  ```
  $\Longrightarrow$ errore!

- se i valori iniziali sono meno degli elementi, i rimanenti vengono posti a 0

  `int n[10] = {3};` azzera i rimanenti 9 elementi del vettore
  `float af[5] = {0.0};` pone a 0.0 i 5 elementi
  `int x[5] = {};` errore!

- se ci sono più inizializzatori di elementi, si ha un errore a tempo di compilazione

  Esempio: `int v[2] = {1, 2, 3};` errore!

- se si mette una sequenza di valori iniziali, si può omettere la lunghezza (viene presa la lunghezza della sequenza)

  Esempio: `int n[] = {1, 2, 3};` equivale a `int n[3] = {1, 2, 3};`

- In C l'unica operazione possibile sugli array è l'accesso ai singoli elementi.
- Ad esempio, non si possono effettuare direttamente delle assegnazioni tra vettori.

Esempio:

```
int a[3] = {11, 22, 33};
int b[3];
b = a;                          errore!
```


Esempi

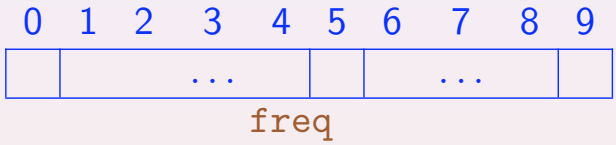
- Calcolo della somma degli elementi di un vettore.

```
int a[10], i, somma = 0;
...
for (i = 0; i < 10; i++)
  somma += a[i];
printf("%d", somma);
```

- Leggere N interi e stampare i valori maggiori di un valore intero y letto in input.

```
#include <stdio.h>
#define N 4
main()   {
int ris[N];
int y, i;
printf("Inserire i %d valori:\n", N);
for (i = 0; i < N; i++) {
  printf("Inserire valore n.  %d:  ", i+1);
  scanf("%d", &ris[i]);     }
printf("Inserire il valore y:\n");
scanf("%d", &y);
printf("Stampa i valori maggiori di %d:\n", y);
for (i = 0; i < N; i++)
  if (ris[i] > y)
    printf("L'elemento %d:  %d è maggiore di %d\n",
            i+1, ris[i],y);
}
```


- Leggere una sequenza di caratteri terminata dal carattere \n di fine linea e stampare le frequenze delle cifre da '0' a '9'.
- utilizziamo un vettore freq di 10 elementi nel quale memorizziamo le frequenze dei caratteri da '0' a '9'



freq[0] conta il numero di occorrenze di '0'

...

freq[9] conta il numero di occorrenze di '9'

- utilizziamo un ciclo per l'acquisizione dei caratteri in cui aggiorniamo una delle posizioni dell'array tutte le volte che il carattere letto è una cifra

```
int i; char ch;
int freq[10] = {0};
do {
  ch = getchar();
  switch (ch) {
    case '0':  freq[0]++; break;
    case '1':  freq[1]++; break;
    case '2':  freq[2]++; break;
    case '3':  freq[3]++; break;
    case '4':  freq[4]++; break;
    case '5':  freq[5]++; break;
    case '6':  freq[6]++; break;
    case '7':  freq[7]++; break;
    case '8':  freq[8]++; break;
    case '9':  freq[9]++; break;
    }
} while (ch != '\n');
printf("Le frequenze sono:\n");
for (i = 0; i < 10; i++)

  printf("Freq.  di %d:  %d\n", i, freq[i]);
```


- Nel ciclo **do-while**, il comando `switch` può essere rimpiazzato da un **if** come segue

```
if (ch >= '0' && ch <= '9')
  freq[ch - '0']++;
```

  Infatti:
- i codici dei caratteri da `'0'` a `'9'` sono consecutivi
- dato un carattere `ch`, l'espressione intera `ch - '0'` è la distanza del codice di `ch` dal codice del carattere `'0'`. In particolare:
  - `'0' - '0' = 0`
  - `'1' - '0' = 1`
  - ...
  - `'9' - '0' = 9`

- Leggere da tastiera e memorizzare una sequenza di interi di lunghezza prefissata DIM. Stampare quindi la sequenza in ordine inverso.

```
#include <stdio.h>
#define DIM 20
main() {
int Mem[DIM];
int i = 0;
/* inserimento dei valori */
printf("Inserire i %d interi della sequenza:\n", DIM);
for (i = 0; i < DIM; i++) {
  printf("Inserire un intero n.  %d:  ", i);
  scanf("%d", &Mem[i]); }
/* stampa in ordine inverso */
for (i = DIM -1; i > = 0; i--)
printf("%d\n", Mem[i]);
```


- Leggere da tastiera i risultati (double) di 20 esperimenti. Stampare il numero d'ordine ed il valore degli esperimenti per i quali il risultato è minore del 50% della media.

```
#include <stdio.h>
#define DIM 20
main() {
double ris[DIM], media;
int i;
/* inserimento dei valori */
printf("Inserire i %d risultati dell'esperimento:\n", DIM);
for (i = 0; i < DIM; i++) {
  printf("Inserire risultato n.  %d:  ", i);
  scanf("%lg", &ris[i]); }
/* calcolo della media */
media = 0.0;
for (i = 0; i < DIM; i++)
    media = media + ris[i];
media = media/DIM;
printf("Valore medio:  %g\n", media);
/* stampa dei valori minori di media*0.5 */
printf("Stampa dei valori minori di media*0.5:\n");
for (i = 0; i < DIM; i++)
    if (ris[i] < media * 0.5)
      printf("Risultato n.  %d:  %g\n", i, ris[i]);        }
```

Possibili ottimizzazioni

- Calcolare la media mano a mano che si leggono i valori (si risparmia un for).
- Dichiarare una variabile soglia dove memorizzare il valore media * 0.5 da usare nei confronti, senza ogni volta doverlo calcolare di nuovo.
- ATTENZIONE: utilizzare la variabile media per fare altrettanto ci permetterebbe di risparmiare sul numero di variabili, ma renderebbe anche il codice poco leggibile e modificabile. Altrove ci potrebbe infatti servire ancora il valore originario della media.