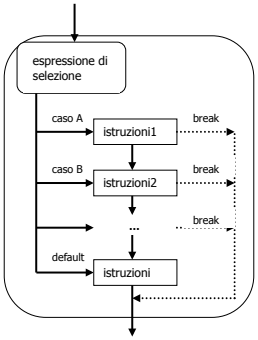
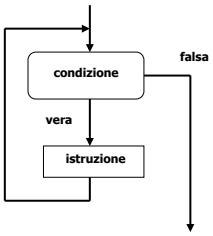
Il linguaggio C
La programmazione strutturata





Operatore condizionale

Operatore condizionale:

<condizione> ? <parteVera> : <parteFalsa>

restituisce il valore di <parteVera> o di <parte Falsa>, in base al seguente criterio:

- la <parteVera> viene valutata solo se la <condizione> è verificata (valore diverso da 0)
- la <parteFalsa> viene valutata solo se la <condizione> non è verificata (valore uguale a zero)

Esempio:

```
ris=a>b? a:b;
```

equivale a:

```
if(a>b)
    ris=a;
  else
    ris=b;
```

Operatore condizionale

```
int a,b,c;
// …input variabili
if( ( a>b+c )||( b>a+c )||( c>a+b ) )
     printf( "non è un triangolo" );
else
     printf( "triangolo %s",
           ( a==b )||( b==c )||( a==c )?
                ( ( a==b )&&( b==c )?
                     "equilatero" :
                     "isoscele"
                ):
                "scaleno" );
```


Istruzioni per il trasferimento del controllo: `break` e `continue`

Istruzione `break`:

L'istruzione **`break`** provoca l'uscita immediata dal ciclo (o da un'istruzione **`switch`**) in cui è racchiusa.

Esempio: fattoriale

```
for (F=1, i=1; ; i++)/* manca il controllo */
   if (i>N) break;
   else F=F*i;
```

Istruzione `continue`:

L'istruzione **`continue`** provoca l'inizio della successiva iterazione del ciclo in cui è racchiusa (non si applica a **`switch`**).

Esempio: somma dei valori pari compresi tra 1 e N

```
for (sum=0, i =1; i <= N; i++)
   if (i%2) continue;
   else  sum+=i;
```

Istruzioni per il trasferimento del controllo: `break` e `continue`

```
int n, break_loop=0;

  while( !break_loop ) {
       printf( "inserisci un numero" );
       scanf( "%d", &n );
       if( n%2 ) {
             printf( "%d dispari\n", n );
       }
       else break_loop=1;
  }
```


Istruzioni per il trasferimento del controllo: `break` e `continue`

```
int n;

  while( 1 ) {
       printf( "inserisci un numero" );
       scanf( "%d", &n );
       if( n%2 ) {
             printf( "%d dispari\n", n );
       }
       else break;
  }
```

Istruzioni per il trasferimento del controllo: `break` e `continue`

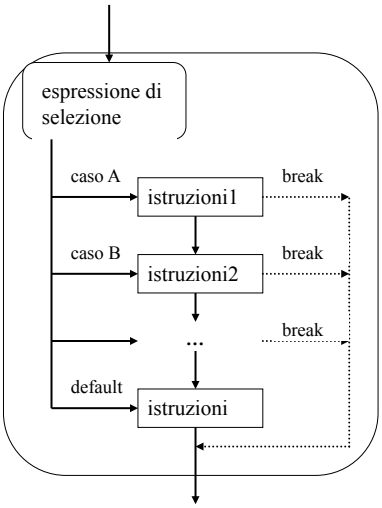
```
int n, break_loop=0, proceed_to_next_cycle;

  while( !break_loop ) {
       proceed_to_next_cycle=0;
       printf( "inserisci un numero" );
       scanf( "%d", &n );
       if( n%2 ) {
             printf( "%d dispari\n", n );
             if( n>10 )
                   proceed_to_next_cycle=1;
       }
       else break_loop=1;
       if( !break_loop&&!proceed_to_next_cycle ) {
             printf( "dicevo: %d dispari\n", n );
       }
  }
```


Istruzioni per il trasferimento del controllo: `break` e `continue`

```
int n, break_loop=0;

  while( !break_loop ) {
       printf( "inserisci un numero" );
       scanf( "%d", &n );
       if( n%2 ) {
             printf( "%d dispari\n", n );
             if( n>10 )
                   continue;
       }
       else break_loop=1;
       if( !break_loop ) {
             printf( "dicevo: %d dispari\n", n );
       }
  }
```

Istruzione di scelta multipla: `switch`

- Consente di scegliere fra molte istruzioni (alternative o meno) in base al valore di una espressione di selezione.
- L'espressione di selezione (*selettore*) deve denotare un valore enumerabile (intero, carattere,…).



Istruzione di scelta multipla: `switch`

Sintassi:

```
switch (<selettore>)
{
  case <costante1>: <blocco1>; [break;]
  case <costante2>: <blocco2>; [break;]
  ...
  case <costanteN>: <bloccoN>; [break;]
  [default: <bloccoDiDefault>;]
}
```

Dove:

- <**selettore**> è un'espressione di tipo enumerabile (intero o char)
- <**costante1**>, <**costante2**> ecc. sono costanti dello stesso tipo del selettore (**etichette**)
- <**blocco1**>, <**blocco2**> , ecc. sono sequenze di istruzioni .

Istruzione di scelta multipla: `switch`

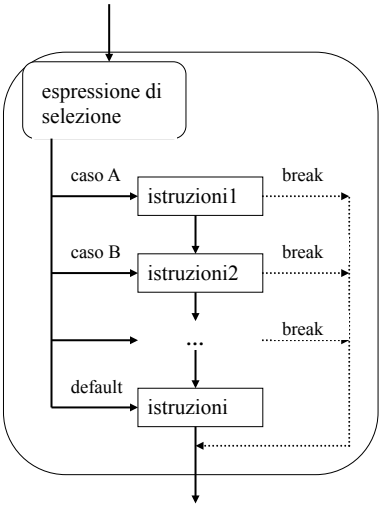
```
switch (<selettore>)
{
  case <costante1>: <blocco1>; [break;]
  case <costante2>: <blocco2>; [break;]
  ...
  case <costanteN>: <bloccoN>; [break;]
  [default: <bloccoDiDefault>;]
}
```

Significato:

- Il valore dell'esepressione <selettore> viene confrontato con ogni etichetta (<costante1>, <costante2>, ecc.) nell'ordine testuale.
- Se l'espressione <selettore> restituisce un valore uguale ad una delle costanti indicate (per esempio <costantei>), si esegue il <bloccoi> e tutti i blocchi dei rami successivi.
- I blocchi **non sono** mutuamente esclusivi: possibilità di eseguire in sequenza più blocchi di istruzioni.

`switch`

I vari rami **non sono mutuamente esclusivi:** imboccato un ramo, si eseguono anche tutti i rami successivi a meno che non ci sia il comando break a forzare esplicitamente l'uscita.



Switch: esempio

Calcolo della durata di un mese: **12 rami mutuamente esclusivi**

```
int mese;
...
switch (mese)
    {
    case 1 : giorni = 31; break;
    case 2:  if (bisestile) giorni = 29;
             else giorni = 28;
             break;
    case 3:  giorni = 31;  break;
    case 4:  giorni = 30;  break;
    case 5:  giorni =31;   break;
    case 6:  giorni = 30;  break;
    case 7:  giorni =31;   break;
    case 8:  giorni =31;   break;
    case 9:  giorni =30;   break;
    case 10:  giorni =31;   break;
    case 11:  giorni =31;   break;
    case 12:  giorni = 31;
    }
```


Esempio switch

Oppure, soluzione piu` sintetica mediante il ramo **`default`**:

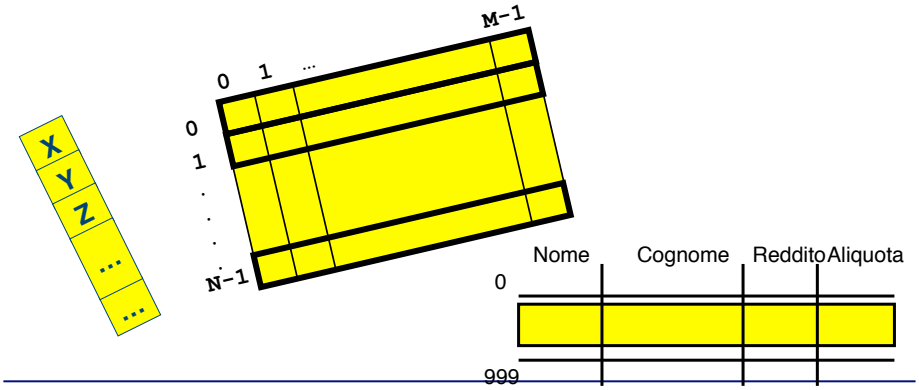
```
switch (mese)
{
case 2:
  if (bisestile) giorni = 29;
  else giorni = 28;
  break;
case 4:  giorni = 30;  break;
case 6:  giorni = 30;  break;
case 9:  giorni = 30;  break;
case 11:  giorni = 30;  break;
default:  giorni = 31;
}
```

Esempio switch

Oppure:

```
switch (mese)
{
case 2:
  if (bisestile) giorni = 29;
  else giorni = 28;
  break;
case 4:
case 6:
case 9:
case 11:  giorni = 30;  break;
default:  giorni = 31;
}
```


Il linguaggio C
Tipi strutturati



Tipi di dato strutturati

I dati strutturati (o strutture di dati) sono ottenuti mediante composizione di altri dati (di tipo semplice, oppure strutturato).

Tipi strutturati in C:

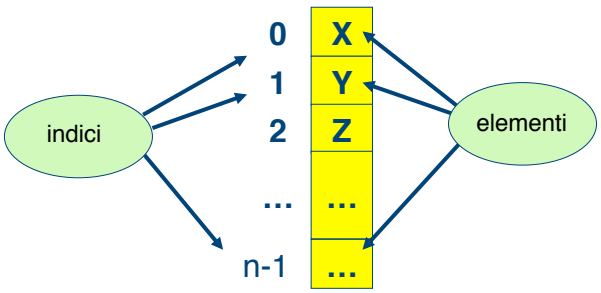
- **vettori** (o ***array***)
- **record** (***struct***)
- [record varianti (*union*)]

Vettori (o *array*)

Un **vettore** (o ***array***) è un insieme ordinato di elementi tutti dello stesso tipo.

Caratteristiche del vettore:

- omogeneità
- ordinamento, ottenuto mediante dei valori interi (***indici***) che consentono di accedere ad ogni elemento della struttura.



Definizione di vettori in C

Nel linguaggio C per definire vettori, si usa il ***costruttore di tipo*** **[ ]**.

Sintassi:

<id-tipo> <id-variabile> `[` **<dimensione>** `];`

dove:

- **<id-tipo>** è l'identificatore di tipo degli elementi componenti,
- **<dimensione>** è una costante intera che rappresenta il numero degli elementi componenti,
- **<id-variabile>** è l'identificatore della variabile strutturata (il vettore) così definita

Esempio:

```
int    V[10];  /* vettore di dieci elementi interi */
```

NB: La dimensione (numero di elementi del vettore) deve essere una **costante intera**, nota al momento della dichiarazione.

```
int    N;
char   V[N];  /* def. sbagliata!!! */
```


Vettori in C

```
int V[25]; /*def. di un vettore di 25 elementi interi*/
```

Indici:

- ad ogni elemento è associato univocamente un ***indice***.
- è possibile riferire un singolo elemento specificando l'indice `i` corrispondente, utilizzando la notazione `V[i]`.
- L'indice deve essere di tipo enumerabile (ad esempio: `int, char`).
- se **N** è la **dimensione** del vettore (numero degli elementi), il dominio degli indici è l'intervallo

[0,N-1]

Esempio:

```
float  A[3], i; /* A è un vettore di 3 reali*/
A[0]=22.05; /*ass. di un valore al primo elemento */
A[2]=17.9; /*ass. di un valore al terzo elemento */
A[1]=0; /*ass. di un valore al secondo elemento */
```

	A
0	22.05
1	0.0
2	17.09

Vettori in C

Operatori:

In C **non esistono operatori specifici** per i vettori.

Per operare sui vettori è necessario operare singolarmente sugli elementi componenti (coerentemente con il tipo ad essi associato).

Pertanto:

➔ **non è possibile l'assegnamento diretto** tra vettori:

```
int V[10], W[10];
…
V=W;    /* è scorretto! */
```

➔ **non è possibile leggere (o scrivere)** un intero vettore (fanno eccezione, come vedremo, le stringhe); occorre leggere/scrivere le sue componenti;

Esempio: lettura di un vettore:

```
int V[100];
int  i;

for(i=0; i<100; i++)
   {            printf("valore %d-simo? ", i);
        scanf("%d", &V[i]);
   }
```


Gestione degli elementi di un vettore

Le singole componenti di un vettore possono essere elaborate con gli operatori del tipo ad esse associato.

Esempio: agli elementi di un vettore di interi è possibile applicare tutti gli operatori definiti per gli interi:

```
int A[10], n=7 i=2;
A[i] =n%i;              /* A[2]=1 */
A[9]=A[i++]+7;          /* A[9]=1+7=8, i=3 */
scanf("%d",&A[i]);      /*ass. ad A[3] il valore letto
*/
```

typedef … … [ ]

In C è possibile utilizzare **typedef … … [ ]** per ***dichiarare*** tipi di dato non primitivi rappresentati da vettori.

Sintassi della dichiarazione:

```
typedef <tipo-componente> <tipo-vettore> [<dim>];
```

dove:

- **<tipo-componente>** è l'identificatore di **tipo di ogni singola componente**
- **<tipo-vettore>** è l'identificatore che si attribuisce al **nuovo tipo**
- **<dim>** è il **numero di elementi** che costituiscono il vettore (deve essere una costante).

Esempio:

```
typedef int Vettore[30];
Vettore  V1,V2;    /* V1 e V2 sono variabili di tipo
            Vettore; ognuno rappresenta un vettore
                 di 30 elementi interi. */
```

➔ **V1 e V2 possono essere utilizzati come vettori di interi.**

Vettori in sintesi

Riassumendo:

- **Definizione di variabili di tipo vettore:**

```
<tipo-componente> <nome-variabile>[<dim>];
```

- **Dichiarazione di tipi** non primitivi rappresentati da vettori:

```
typedef  <tipo-componente>  <nome-tipo>  [<dim>];
```

Caratteristiche:

- **<dim>** è una costante enumerabile.
- **<tipo-componente>** è un qualsiasi tipo, semplice o strutturato.
- il vettore è una sequenza di dimensione fissata **<dim>** di componenti dello stesso tipo **<tipo-componente>.**
- la singola componente i-esima di un vettore V è individuata dall'indice i-esimo, secondo la notazione **V[i].**
- L'intervallo di variazione degli indici è **[0 , … , dim-1]**
- sui singoli elementi è possibile operare secondo le modalità previste dal tipo **<tipo-componente>.**

Inizializzazione di un vettore

Come attribuire un valore iniziale ad ogni elemento di un vettore?

2 possiblità. 1) Mediante un ciclo:

Per attribuire un valore iniziale agli elementi di un vettore, si può eseguire una sequenza di assegnamenti alle N componenti del vettore.

Esempio:

```
#define N 30
typedef int vettore [N];
vettore v;
int i;
...
for( i=0; i<N; i++ )
  v[i]=0;
```

`#define`: La **`#define`** è una direttiva del *preprocessore* C che serve ad associare un identificatore (nell'esempio N) ad una costante (nell'esempio, 30). Rende il programma più facilmente modificabile.

Inizializzazione di un vettore

Come attribuire un valore iniziale ad ogni elemento di un vettore?

2) In alternativa al ciclo, è possibile l'inizializzazione in fase di definizione:

Esempio:

```
int v[10] = {1,2,3,4,5,6,7,8,9,10};

/*  v[0] = 1; v[1]=2;...v[9]=10; */
```

Addirittura è possibile fare:

```
int v[]={1,2,3,4,5,6,7,8,9,10};
```

➔ **La dimensione è determinata sulla base dell'inizializzazione.**

Esempio

Problema: (somma di due vettori)

Si realizzi un programma che, dati da standard input gli elementi di due vettori A e B, entrambi di 10 interi, calcoli e stampi gli elementi del vettore C (ancora di 10 interi), ottenuto come somma di A e B.

- Dichiariamo il nuovo tipo di dato **`vettint`**:

```
typedef int vettint[10];
```

- Definiamo i due vettori dati A e B, e il vettore C risultante dalla somma :

```
vettint A,B,C;
```



```
#include <stdio.h>
typedef int vettint[10];

main()
{  vettint A, B, C;
   int i;
   for(i=0; i<10; i++) /* lettura del vettore A */
   {     printf("valore di A[%d] ? ", i);
     scanf("%d", &A[i]);
   }
   for(i=0; i<10; i++) /* lettura del vettore B */
   {     printf("valore di B[%d] ? ", i);
     scanf("%d", &B[i]);
   }

   for(i=0; i<10; i++) /* calcolo del risultato */
     C[i]=A[i]+B[i];

   for(i=0; i<10; i++) /* stampa del risultato C */
     printf("C[%d]=%d\n",i, C[i]);
}
```

risultato

Esempio

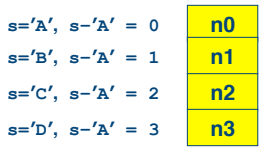
(calcolo del numero medio di iscritti per classe)

Si consideri una scuola media costituita da 4 sezioni (da 'A' a 'D'), ognuna costituita da tre classi (“livello” 1, 2, 3).

Dati gli iscritti ad ogni classe di ogni sezione, si vuole calcolare il numero medio di studenti per classe relativo ad ogni sezione.

Definiamo un nuovo tipo che rappresenta il **`livello`**:

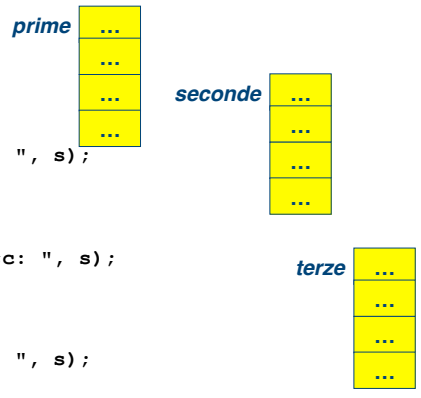
```
/* un elemento per ogni sezione */

typedef int livello[4];
livello prime, seconde, terze;
```



NB: L’indice di un vettore deve essere di tipo ***enumerabile*** ➔ può essere un **`char`**:

```
typedef char sezione;
sezione s;  // s = 'A', 'B', …
/*indice per accedere ai vettori delle classi */
```


```
#include <stdio.h>
typedef int livello[4]; /*un elemento per ogni sezione */
typedef char sezione;
main()
{  livello prime, seconde, terze;
   sezione s;
   float  media=0;
   for(s='A'; s<='D'; s++)
   {     printf("Iscritti alla prima %c: ", s);
     scanf("%d", &prime[s-'A']);
   }
   for(s='A'; s<='D'; s++)
   {     printf("Iscritti alla seconda %c: ", s);
     scanf("%d", &seconde[s-'A']);
   }
   for(s='A'; s<='D'; s++)
   {     printf("Iscritti alla terza %c: ", s);
     scanf("%d", &terze[s-'A']);
   }
   for(s='A'; s<='D'; s++, media=0)
   {     media = ( prime[s-'A']+seconde[s-'A']+terze[s-'A'] )/3 ;
     printf("\nLa media degli iscritti per classe
                 nella sezione %c è: %f\n", s, media );
   }
}
```

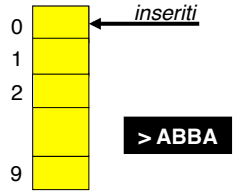


Esempio
(stampa caratteri senza ripetizioni)

Si leggano da input alcuni caratteri alfabetici maiuscoli (hp: al massimo, 10) e si riscrivano in uscita evitando di ripetere caratteri già stampati.

Soluzione:

***while** <ci sono caratteri da leggere>*
{ <leggi carattere>;
if <non già memorizzato>
<memorizzalo in una struttura dati>;
};
***while** <ci sono elementi della struttura dati>*
<stampa elemento>;



- Occorre una variabile dove memorizzare (senza ripetizioni) gli elementi letti in ingresso: il ***vettore A***

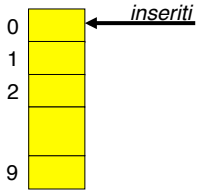
```
char A[10];
```

- il vettore A verrà riempito con al più 10 valori: definiamo la variabile ***inseriti***, che rappresenterà la *"dimensione **logica**"* di A (cioè, il numero di elementi significativi):

```
int inseriti=0;
```

- *la dimensione **fisica** del vettore è 10, mentre la dimensione **logica** è data dal numero effettivo di caratteri diversi tra loro*

Soluzione




```
#include <stdio.h>
main()
{  char A[10], c;
   int i, j, inseriti=0, trovato;
   printf( "\n Dammi 10 caratteri: " );
   for ( i=0; ( i<    10    ); i++ )
   {     scanf("%c", &c);
     /* verifica unicità: */
        trovato=0;
        for( j=0 ; (j<inseriti)&&!trovato ; j++ )
               if (c==A[j])
                     trovato=1;
     /* se c non è ancora presente in A, lo inserisco: */
     if ( !trovato )
     {    A[inseriti]=c;
          inseriti++;
     }
   }
   printf( "Inseriti %d caratteri \n", inseriti );
   for ( i=0 ; i<inseriti ; i++ )
     printf( "%c\n", A[i] );
}
```

Esempio

ordinamento di un vettore

Dati n valori interi forniti in ordine casuale, stampare in uscita l'elenco dei valori dati in ordine crescente.

➔ È necessario mantenere in memoria tutti i valori dati per poter effettuare i confronti necessari: utilizziamo i vettori.

Soluzione:

Esistono vari procedimenti risolutivi per questo problema (algoritmi di ordinamento, o ***sorting***):

- **naïve-sort**
- bubble sort
- quick sort
- merge sort
- ...

Risolveremo il problema utilizzando il Metodo dei Massimi successivi (*naïve-sort*).

Ordinamento di un vettore mediante *naïve sort*

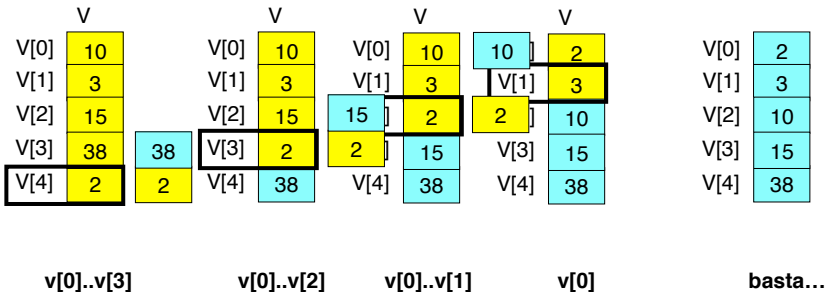
Descrizione dell' algoritmo:

Dato un vettore: `int V[dim];`

1. eleggi un elemento come massimo temporaneo (`V[max]`)
2. confronta il valore di `V[max]` con tutti gli altri elementi del vettore (`V[i]`):
   se `V[i]>V[max] , max=i`
3. quando hai finito i confronti, scambia `V[max]` con `V[dim-1];` il massimo ottenuto dalla scansione va in ultima posizione.
4. riduci il vettore di un elemento (`dim=dim-1`) e, se `dim>1`, torna a 1.

naïve sort

con vettore di dimensione `dim = 5`



v[0]..v[3] **v[0]..v[2]** **v[0]..v[1]** **v[0]** **basta…**


```
#include <stdio.h>
#define dim 10
main()
{  int V[dim], i,j, max, tmp, quanti;
   /* lettura dei dati */
   for ( i=0; i<dim; i++ )
   {     printf("valore n. %d: ",i);
               scanf("%d", &V[i]);
   }
   /*ordinamento */
   for( i=dim-1; i>1; i-- )
   {     max=i;
     for( j=0; j<i; j++ )
        if ( V[j]>V[max] )
          max=j;
     if ( max!=i ) /*scambio */
     {    tmp=V[i];
          V[i]=V[max];
          V[max]=tmp;
     }
   }
/* stampa del vettore ordinato per esercizio */
```