Esempi comparati `while`, `do...while` □

Algoritmo.
- `inizializza Somma a zero`
- `finchè la risposta è "continuazione"`
  - `visualizza messaggio`
  - `acquisisci un numero I`
  - `aggiungi I a Somma`
  - `domanda se continuare`
- `visualizza il risultato`

- si domanda se continuare solo all'interno del corpo del ciclo
  $\Longrightarrow$ l'algoritmo si presta ad essere implementato con `do...while`

Esempi comparati while, do...while

Programma.

```
#include <stdio.h>
main() {
  int I, Somma=0;
  char Ch;
  do {
    printf("Intero in input \n");
    scanf("%d", &I);
    Somma+=I;
    printf("Continua (S/N) ? \n");
    scanf("%1s", &Ch);
  } while (Ch !='N');
  printf("\n Somma dei numeri positivi: %d" ,Somma);
}
```

Istruzione `for`

- Sintassi dell'istruzione `for`:

$$\langle \text{istr-for} \rangle \longrightarrow \texttt{for (} \langle \text{espr} \rangle \texttt{;} \langle \text{espr} \rangle \texttt{;} \langle \text{espr} \rangle \texttt{)} \langle \text{istr} \rangle$$

- Il significato dell'istruzione `for`:

```
for (espressione1 ; espressione2 ; espressione3)
  istruzione
```

è riconducibile alla seguente istruzione `while`:

```
espressione1 ;
while (espressione2) {
  istruzione
  espressione3 ;
}
```

Istruzione `for`

- Per ora ci limitiamo ad un utilizzo particolare dell'istruzione `for`, volta al controllo del numero di ripetizioni
- In tale forma limitata
  - una variabile di controllo assume in successione tutti i valori compresi tra un minimo e un massimo
  - *espressione1* assegna di un valore iniziale alla variabile di conteggio
  - *espressione2* esprime il valore finale della variabile di conteggio
  - *espressione3* incrementa o decrementa la variabile di conteggio

Esempio. Visualizza i cubi degli interi da 1 a 10

```
for (J = 1; J<= 10; J++)
  printf("%d\n",J*J*J);
```

Istruzione `for`

- Se il valore della variabile di controllo non è modificato esplicitamente con istruzioni di assegnamento nel corpo del ciclo allora
  - la variabile di controllo assume tutti i valori compresi fra l'espressione iniziale e quella finale
  - al termine delle ripetizioni il valore della variabile di controllo contiene il valore successivo o precedente al valore finale

Ripetizioni nidificate

- Una ripetizione nidificata è un'istruzione contenuta in una istruzione di ripetizione che è a sua volta una istruzione di ripetizione

  Esempio. Per ogni numero fornito da input, calcolare una potenza letta da input (una sola volta, in anticipo)

  Algoritmo.
  ```
  - acquisisci l'esponente E
  - visualizza messaggio
  - acquisisci un numero B
  - fino a quando B è diverso da zero
    - calcola la potenza B elevato a E
    - acquisisci un numero B
    - visualizza il risultato
  ```

- Il passo in cui si calcola la potenza è una ulteriore ripetizione, in cui il numero di ripetizioni è noto a priori

Ripetizioni nidificate

- Consideriamo allora il sottoproblema

  Problema. Elevare B alla potenza E

  Algoritmo.
  ```
  - inizializza P a uno
  - per tutti i valori di I da 1 a E
    - moltiplica P per B
  ```

- Nidificando il secondo algoritmo nel primo

  Algoritmo.
  ```
  - acquisisci l'esponente E
  - visualizza messaggio
  - acquisisci un numero B
  - fino a quando B è diverso da zero
    - inizializza P a uno
    - per tutti i valori di I da 1 a E
      - moltiplica P per B
    - acquisisci un numero B
    - visualizza il risultato
  ```

Ripetizioni nidificate

- Il programma implementato in C contiene una ripetizione realizzata con `for` all'interno di una ripetizione realizzata con `while`

  Programma.

```
#include <stdio.h>
main () {
  int B,P,E,I;
  printf("Inserire l'esponente non negativo > ");
  scanf("%d", &E) ;
  printf("Inserire la base (zero per terminare) ");
  scanf("%d", &B) ;
  while (B!= 0) {
    P=1;
```

  (continua)

Ripetizioni nidificate

- *(continua dalla pagina precedente)*

```
    for(I=1;I<=E;I++)
      P=P*B;
    printf("%d elevato a %d = %d\n",B,E,P);
    printf("Inserire la base (zero per terminare) ");
    scanf("%d", &B) ;
  }
}
```

- L'inserimento del valore zero per `B` determina la fine dell'esecuzione senza che nemmeno una ripetizione sia eseguita

Istruzioni di controllo

- Una istruzione di controllo permette di scegliere fra una o più istruzioni da eseguire, secondo il valore di una espressione
- Le istruzionifra cui si sceglie possono essere semplici o composte
- Caso particolare: è specificata una sola istruzione
  - la scelta è tra eseguire e non eseguire quella istruzione
- In C esistono due istruzioni di scelta
  - `if`
    - scelta binaria
  - `switch`
    - scelta multipla

Istruzione `if`

- Permette di scegliere tra l'esecuzione di due istruzioni, in base al valore di una espressione
- Sintassi:

  $\langle \text{istr-if} \rangle \longrightarrow \texttt{if(}\ \langle \text{espr} \rangle\ \texttt{)}\ \langle \text{istr} \rangle \mid \texttt{if(}\ \langle \text{espr} \rangle\ \texttt{)}\ \langle \text{istr} \rangle\ \texttt{else}\ \langle \text{istr} \rangle$

  - **prima variante** valuta l'espressione; se non è zero, esegui l'istruzione

    seconda variante valuta l'espressione; se non è zero, esegui la prima istruzione, altrimenti esegui la seconda istruzione
  - In caso di nidificazione di piú istruzioni `if`, ogni eventuale `else` si riferisce sempre alla parola `if` piú vicina

interpretazione errata $\quad \texttt{if}\ \langle \text{espr} \rangle\ \texttt{if}\ \langle \text{espr}_1 \rangle\ \langle \text{istr}_1 \rangle\ \texttt{else}\ \langle \text{istr}_2 \rangle$

interpretazione corretta $\quad \texttt{if}\ \langle \text{espr} \rangle\ \texttt{if}\ \langle \text{espr}_1 \rangle\ \langle \text{istr}_1 \rangle\ \texttt{else}\ \langle \text{istr}_2 \rangle$

Istruzione `if`

- Nidificazione di istruzioni `if`

```
if (E1)
  S1
else
  if (E2)
    S2
  else
    S3
```

E1	E2	Istruzione eseguita
vero	non valutata	S1
falso	vero	S2
falso	falso	S3

Istruzione `if`

- Nidificazione di istruzioni `if`

```
if (E1)
  if (E2)
    S1
  else
    S2
```

E1	E2	Istruzione eseguita
vero	vero	S1
vero	falso	S2
falso	non valutata	-

Istruzione `if`

- Nidificazione di istruzioni `if`

```
if (E1) {
  if (E2)
    S1
}
else
  S3
```

E1	E2	Istruzione eseguita
vero	vero	S1
falso	falso	-
falso	non valutata	S3

Istruzione `if`

Esempio. Dati tre interi positivi in ordine non decrescente, trovare il tipo di triangolo (equilatero, isoscele, scaleno) da essi definito

Confronto fra i lati (A,B,C)	Analisi
$A + B < C$	non è un triangolo
$A = B = C$	triangolo equilatero
$A = B$ o $B = C$	triangolo isoscele
$A \neq B \neq C$	triangolo scaleno

Istruzione `if`

- È sufficiente eseguire una serie di test tra coppie di lati per isolare il caso che si è verificato

Algoritmo.

```
- acquisisci la terna di misure A,B,C
- se A+B < C allora non è un triangolo
- altrimenti
  - se A=C allora è equilatero (sono in ordine)
  - altrimenti
    - se A=B o B=C allora è isoscele
    - altrimenti è scaleno
```

Istruzione `if`

Programma.

```
#include <stdio.h>
main () {
  int A, B, C;
  /*Legge e stampa i dati */
  do {
    printf("Lunghezza lati, in ordine non decrescente? ");
    scanf("%d%d%d", &A, &B, &C);
  }
  while( A>B || B>C);
  printf("%d %d %d\n", A, B, C);
```

(continua)

Istruzione `if`

(continua dalla pagina precedente)

```
  /* Esegue l'analisi e stampa i risultati */
  if (A + B < C)
    printf("non e' un triangolo");
  else {
    printf("e' un triangolo ");
    /* determina il tipo di triangolo */
    if (A == C)
      printf("equilatero");
    else
      if ((A == B) || (B == C))
        printf("isoscele");
      else
        printf("scaleno");
  }
}
```

Istruzione `break`

- La prossima istruzione da eseguire diventa la prima istruzione seguente il blocco che contiene `break`
- Spesso utilizzato nel blocco di una ripetizione per determinare un'uscita immediata al verificarsi di una condizione
  - Es. verificare una condizione con al piú 10 tentativi

```
...
for (I=0; I<10; I++) {
  scanf ("%d", &K) ;
  if (I==K){
    break;
  }
}
...
```

Istruzione `break`

- L'utilizzo di `break` ha l'effetto di nascondere una condizione di controllo all'interno del corpo del ciclo
- La lettura dei programmi risulta piú difficile
- Non se consiglia l'uso quando esistono alternative piú semplici

Istruzione `break`

Esempio (deprecato utilizzo di break). Calcolare la somma dei numeri in input, terminando al primo numero negativo, che non deve essere aggiunto alla somma.

```
int n,s;

while (1)
  scanf("%d", &n);
  if (n < 0)
    break;
  s += n;
```

```
scanf("%d", &n);
while ( n >= 0 )
  s += n;
  scanf("%d", &n);
```

Istruzione `continue`

- Interrompe l'iterazione corrente e avvia l'iterazione successiva
- Può essere utilizzata solo all'interno di `while`, `do...while`, `for`, con il seguente comportamento:

  `while` viene valutata l'espressione; se non ha valore non zero, viene eseguita l'istruzione

  `do...while` stesso comportamento

  `for` viene eseguita l'espressione di incremento e si valuta l'espressione; se non ha valore non zero, viene eseguita l'istruzione

Istruzione `continue`

Esempio. Acquisire un numero da input e determinare se è divisibile per `I`, con `I` variabile da 1 a 10

- *acquisisci dividendo K*
- *per I=1,...,10*
  - *se K/I non dà resto zero*
    - *avvia nuova iterazione*
  - *visualizza dividendo, divisore*

```
#include <stdio.h>
main() {
  int I,K;
  scanf("%d",&K);
for (I=1; I<=10; I++) {
  if (K%I != 0)
    continue;
  printf("%d è divisibile per %d\n",K,I)
  }
}
```

Istruzione `continue`

- Riscritto senza `continue`

```
#include <stdio.h>
main () {
  int I,K;
  scanf (" %d" , &K) ;
  for (I=1; I<=10; I++) {
    if (K%I==0)
    printf( "%d e' divisibile per %d\n", K, I);
  }
}
```

Istruzione `switch`

- L'istruzione `switch` permette la scelta tra piú di due alternative, in base al confronto tra il valore di una espressione di tipo `int` o `char` e un insieme di valori costanti
- Sintassi:

$$\langle \text{istr-switch} \rangle \longrightarrow \texttt{switch(}\ \langle \text{espr-intera} \rangle\ \texttt{)}\ \langle \text{corpo-switch} \rangle$$

$$\langle \text{corpo-switch} \rangle \longrightarrow \langle \text{lista-case} \rangle\ \langle \text{sequenza-istr} \rangle$$

$$\langle \text{sequenza-istr} \rangle \longrightarrow \langle \text{istr} \rangle \mid \langle \text{istr} \rangle\ \langle \text{sequenza-istr} \rangle$$

$$\langle \text{lista-case} \rangle \longrightarrow \langle \text{case} \rangle \mid \langle \text{case} \rangle\ \langle \text{lista-case} \rangle$$

$$\langle \text{case} \rangle \longrightarrow \langle \text{espr-intera-costante} \rangle\ \texttt{:} \mid \texttt{default:}$$

Istruzione `switch`

- La sintassi di **`switch`** è soggetta ai seguenti ulteriori vincoli:
- ⟨espr-intera⟩, ⟨espr-intera-costante⟩ sono espressioni di tipo **`int`** o **`char`**
- Non sono ammessi piú ⟨case⟩ con lo stesso valore di ⟨espr-intera-costante⟩
- **`default:`** può comparire una volta sola

Istruzione `switch`

- Funzionamento dell'istruzione `switch`:
  - L'espressione $\langle$espr-intera$\rangle$ viene valutata e confrontata con le espressioni costanti
  - Se il valore di una (e quindi una sola) delle espressioni costanti è uguale al valore dell'espressione intera, allora
    - si esegue la prima istruzione in posizione successiva ai due punti che seguono l'espressione costante; sia I tale istruzione
  - altrimenti
    - se è presente `default`, si esegue la prima istruzione in posizione successiva ai due punti seguenti `default`
  - tutte le istruzioni seguenti I fino al termine dell'istruzione `switch` vengono eseguite in sequenza

Istruzione `switch`

Osservazione. L'istruzione `break` è utile per evitare l'esecuzione delle istruzioni presenti in ⟨case⟩ diversi da quello contenente la ⟨espr-intera-costante⟩ che ha verificato l'uguaglianza con ⟨espr-intera⟩. Pertanto un forma spesso usata dell'istruzione `switch` è la seguente

```
switch (e) {
case c1 :
  ...
  break;
case c2 :
  ...
  break;
  ⋮
default :
  ...
}
```

Istruzione `switch`

Esempio.

```
/*  http://www.its.strath.ac.uk/courses/c */
main() {
  int n;
  do {
    printf("Intero non negativo: ");
    scanf("%d", &n);
  }
  while (n<0);
  switch(n) {
  case 0 :
    printf("Nessuno\n");
    break;
  case 1 :
    printf("Uno\n");
```

(continua)

Istruzione `switch`

(continua)

```
    break;
  case 2 :
    printf("Due\n");
    break;
  case 3 :
  case 4 :
  case 5 :
    printf("Alcuni\n");
    break;
  default :
    printf("Molti\n");
    break;
  }
}
```

Istruzione `switch`

Esempio. Costruire un simulatore di calcolatore tascabile: un programma che legge in input un'espressione aritmetica, cioè una sequenza di valori numerici, separati da operatori `+ - * /`, terminata da `=`, e calcola il valore dell'espressione

- *acquis. primo operando memor. nel risultato*
- *acquis. primo operatore, scartando caratteri non validi*

```
#include <stdio.h>
main()
{
  float Risultato, Valore;
  int Op;
  printf("Espressione: ");
  scanf("%f",&Risultato);
  do
    Op = getchar();
  while (Op!='+' && Op!='-' &&
         Op!='*' && Op!='/' &&
         Op!='=');
```

Istruzione switch

- *finché l'operat. non è =*
  - *acquisisci operando*
  - *esegui operazione tra operando e risult.*

```
while (Op !='=') {
  scanf("%f",&Valore);
  switch(Op) {
  case '+':
    Risultato = Risultato+Valore;
    break;
  case '-':
    Risultato = Risultato-Valore;
    break;
  case '*':
    Risultato = Risultato*Valore;
    break;
  case '/':
    Risultato = Risultato/Valore;
    break;
  }
```

Istruzione switch

- *acquisisci operatore, scartando caratteri non validi*

- *visualizza risultato*

```
                    do
                      Op=getchar();
                    while (Op!='+' &&
                           Op!='-' &&
                           Op!='*' &&
                           Op!='/' &&
                           Op!='=');
                  }
  printf("Risultato: %f",Risultato);
}
```