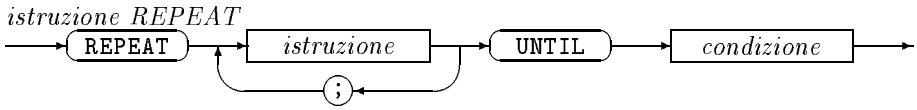
Lezione 7
19–20 ottobre 1999

Argomenti trattati
– L'iterazione: i costrutti `REPEAT`, `WHILE` e `FOR`.

7.1 Il ciclo REPEAT

In Pascal, l'iterazione con controllo in coda del tipo `ESEGUI ... FINO A QUANDO ...` può essere espressa direttamente utilizzando l'istruzione `REPEAT`, la cui carta sintattica è:



L'esecuzione avviene nel seguente modo: prima di tutto vengono eseguite le istruzioni racchiuse tra le parole riservate `REPEAT` e `UNTIL`. Viene quindi valutata la condizione che segue la parola `UNTIL`. Se essa risulta falsa, si ripete eseguendo nuovamente il blocco di istruzioni e valutando di nuovo la condizione. Quando questa risulta vera, si passa ad eseguire l'istruzione successiva. Come già osservato per la struttura `ESEGUI ... FINO A QUANDO ...`, si ricordi che:

- il blocco di istruzioni racchiuse tra le parole `REPEAT` e `UNTIL` viene eseguito sempre *almeno* una volta;
- il ciclo termina quando la condizione risulta vera.

Poiché le parole riservate `REPEAT` e `UNTIL` indicano l'inizio e la fine delle istruzioni da ripetere, non è necessario utilizzare `BEGIN` e `END` per raggruppare le istruzioni che costituiscono il corpo del ciclo.

Esempio: calcolo del massimo comun divisore tra due numeri

Nella Lezione 4, abbiamo presentato una versione dell'algoritmo di Euclide per il calcolo del massimo comun divisore tra due numeri, basato su un costrutto iterativo con controllo in coda:

```
variabili x, y, resto: numeri interi

leggi x, y
ESEGUI
```

```
        resto ← x MOD y
        x ← y
        y ← resto
    FINO A QUANDO resto = 0
    scrivi x
```

Il seguente programma Pascal è basato sul precedente algoritmo:

```
PROGRAM massimocomundivisore (input, output);

{calcola il massimo comun divisore tra due interi positivi}

   VAR
      x, y, resto: integer;

BEGIN
   write('Inserire due interi positivi: ');
   readln(x, y);
   REPEAT
      resto := x MOD y;
      x := y;
      y := resto
   UNTIL resto = 0;
   writeln('Il massimo comun divisore e'' ', x : 1)
END.
```

Si osservi che il programma precedente non effettua alcun controllo sui dati inseriti dall'utente. Nel caso i numeri inseriti non siano entrambi positivi, sarebbe opportuno fornire all'utente un messaggio d'errore e chiedere di ripetere l'inserimento dei dati. A tal fine, possiamo collocare l'istruzione di lettura all'interno di un ciclo, dal quale si esce solo quando entrambi i numeri sono positivi:

```
PROGRAM massimocomundivisore (input, output);

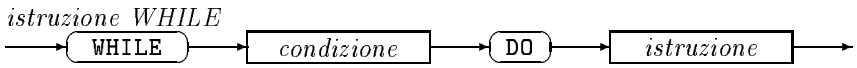
{calcola il massimo comun divisore tra due interi positivi}

   VAR
      x, y, resto: integer;

BEGIN
   REPEAT
      write('Inserire due interi positivi: ');
      readln(x, y);
      IF (x <= 0) OR (y <= 0) THEN
         writeln('Errore: dati non corretti')
   UNTIL (x > 0) AND (y > 0);
   REPEAT
      resto := x MOD y;
      x := y;
      y := resto
   UNTIL resto = 0;
   writeln('Il massimo comun divisore e'' ', x : 1)
END.
```

7.2 Il ciclo WHILE

Il Pascal dispone anche di un'istruzione corrispondente all'iterazione con controllo in testa, del tipo `FINO A QUANDO ... RIPETI`. Questa istruzione, detta ciclo `WHILE`, ha la seguente sintassi:



L'esecuzione avviene nel seguente modo. Prima di tutto viene valutata la condizione. Se essa risulta vera, si esegue l'istruzione contenuta nel ciclo e si valuta nuovamente la condizione. Quando questa risulta falsa, si passa ad eseguire l'istruzione successiva al costrutto iterativo. Si noti che:

- l'istruzione può essere eseguita anche zero volte;
- il ciclo termina quando la condizione risulta falsa.

Come sempre, è possibile raggruppare piú istruzioni tra le parole riservate `BEGIN` ed `END`, formando un'unica istruzione composta.

Esempio: di nuovo il massimo comun divisore

Nell'ultima versione presentata del programma per il massimo comun divisore, è stato scritto il seguente ciclo che viene ripetuto fino a quando l'utente inserisce i dati correttamente:

```
REPEAT
   write('Inserire due interi positivi: ');
   readln(x, y);
   IF (x <= 0) OR (y <= 0) THEN
      writeln('Errore: dati non corretti')
UNTIL (x > 0) AND (y > 0)
```

Si osservi che le condizioni dell'`IF` e del ciclo sono una la negazione dell'altra. In pratica, ad ogni iterazione si effettua un doppio controllo su `x` e `y`. Si potrebbe evitare ciò utilizzando un ciclo con il controllo in testa, ripetendo la prima istruzione `write` e l'istruzione di lettura prima di tale ciclo:

```
write('Inserire due interi positivi: ');
readln(x, y);
WHILE (x <= 0) OR (y <= 0) DO
   BEGIN
      writeln('Errore: dati non corretti');
      write('Inserire due interi positivi: ');
      readln(x, y);
   END
```

A questo punto, può risultare piú comodo sostituire la `writeln` e la `write` all'interno del ciclo, con un'unica istruzione in cui si comunica l'errore e si chiede di ripetere l'inserimento. Ecco il testo completo del programma dopo tale modifica:

```
PROGRAM massimocomundivisore (input, output);

{calcola il massimo comun divisore tra due interi positivi}

   VAR
      x, y, resto: integer;

BEGIN
```

```
   write('Inserire due interi positivi: ');
   readln(x, y);
   WHILE (x <= 0) OR (y <= 0) DO
      BEGIN
         write('Errore nei dati - Inserire due interi positivi: ');
         readln(x, y);
      END; {while}
   REPEAT
      resto := x MOD y;
      x := y;
      y := resto
   UNTIL resto = 0;
   writeln('Il massimo comun divisore e'' ', x : 1)
END.
```

Presentiamo ora un'ulteriore versione del programma, in grado di operare anche con numeri negativi. Ricordiamo che il massimo comun divisore tra due numeri è dato dal piú grande intero che divide entrambi i numeri. Ad esempio, il piú grande intero che divide -25 e 15 è 5, che è anche il massimo comun divisore tra 25 e 15, tra 25 e -15 e tra -25 e -15. In pratica, calcolare il massimo comun divisore tra due numeri equivale a calcolare il massimo comun divisore tra i loro valori assoluti. Possiamo dunque estendere il nostro programma per operare anche con i numeri negativi, utilizzando il seguente schema:

leggi i due numeri
sostituisci i due numeri con i loro valori assoluti
calcola il massimo comun divisore tra i numeri ottenuti
scrivi il risultato

Osserviamo inoltre che, poiché ogni numero divide zero, il massimo comun divisore tra zero e un numero n diverso da zero coincide con il valore assoluto di n. Esaminiamo cosa succede eseguendo il ciclo per il calcolo del massimo comun divisore con `x` che vale zero o `y` che vale zero. Per comodità riportiamo il testo del ciclo:

```
   REPEAT
      resto := x MOD y;
      x := y;
      y := resto
   UNTIL resto = 0
```

Ricordiamo inoltre che al termine dell'esecuzione del ciclo, il valore del massimo comun divisore è contenuto nella variabile `x`. Se `x` contiene inizialmente zero, e `y` un valore n diverso da zero, il calcolo di `x MOD y` fornisce risultato zero, provocando l'uscita dal ciclo, con il valore n in `x`, che è appunto il massimo comun divisore. Al contrario, se `y` contiene inizialmente zero e `x` un valore n diverso da zero, il calcolo di `x MOD y` produce un errore in esecuzione. Per evitare questo problema sarebbe sufficiente non eseguire alcuna volta il corpo del ciclo, visto che `x` contiene già il massimo comun divisore. A tale scopo osserviamo che alla fine di ogni iterazione le variabili `resto` e `y` contengono lo stesso valore (quindi la condizione `resto = 0` potrebbe essere rimpiazzata dalla condizione `y = 0`) e che il ciclo `REPEAT` può essere sostituito dal seguente ciclo `WHILE` che evita, nel caso `y` valga già zero, la prima divisione:

```
   WHILE y <> 0 DO
      BEGIN
         resto := x MOD y;
         x := y;
```

```
        y := resto
    END
```

Osserviamo, infine, che il massimo comun divisore tra zero e zero non esiste.

Alla luce di queste osservazioni, possiamo basare la nuova versione del programma sul seguente schema:

```
variabili x, y, resto: numeri interi

leggi i due numeri
SE entrambi i numeri sono zero
    ALLORA
        scrivi "il massimo comun divisore non esiste"
    ALTRIMENTI
        sostituisci i due numeri con i loro valori assoluti
        calcola il massimo comun divisore tra i numeri ottenuti
        scrivi il risultato
FINESE
```

Per il calcolo dei valori assoluti possiamo utilizzare la funzione predefinita `abs`. La selezione precedente può essere riscritta utilizzando un'istruzione `IF`. Nel ramo `ELSE`, per il calcolo del massimo comun divisore, utilizziamo il ciclo `WHILE` scritto sopra. In questo modo, otteniamo il seguente programma:

```
PROGRAM massimocomundivisore (input, output);

{calcola il massimo comun divisore tra due interi}

   VAR
      x, y, resto: integer;

BEGIN {massimocomundivisore}
   write('Inserire due interi: ');
   readln(x, y);
   IF (x = 0) AND (y = 0) THEN
      writeln('Il massimo comun divisore non esiste')
   ELSE
      BEGIN
         {sostituisci i numeri con i loro valori assoluti}
         x := abs(x);
         y := abs(y);
         {calcola il massimo comun divisore}
         WHILE y <> 0 DO
            BEGIN
               resto := x MOD y;
               x := y;
               y := resto
            END; {while}
         {scrivi il risultato}
         writeln('Il massimo comun divisore e'' ', x : 1)
      END {else}
END. {massimocomundivisore}
```

Esempio: calcolo del prodotto

Presentiamo ora un programma per il calcolo del prodotto di due interi non negativi, utilizzando esclusivamente l'operazioni di somma. Il programma si basa sulla semplice osservazione che il prodotto di due numeri x e y è ottenibile sommando y volte x a zero. Pertanto, possiamo utilizzare una variabile di nome `prodotto`, inizializzata a zero, alla quale aggiungiamo, con un ciclo ripetuto y volte, il valore di x:

```
azzera prodotto
WHILE non hai ripetuto y volte DO
    aggiungi al prodotto il valore di x
```

L'operazione `azzera prodotto` può essere facilmente realizzata con un assegnamento. Per contare il numero di ripetizioni, introduciamo un contatore `cont`, inizializzato a 1 e incrementato ad ogni iterazione:

```
prodotto := 0;
cont := 1;
WHILE cont <= y DO
   BEGIN
      prodotto := prodotto + x;
      cont := cont + 1
   END {while}
```

Si noti che la scelta del ciclo `WHILE` permette di calcolare correttamente anche il prodotto per zero.

Riportiamo il programma completo, ottenuto aggiungendo le dichiarazioni di variabile, una fase iniziale di lettura e una fase finale di scrittura del risultato.

```
PROGRAM prod (input, output);

{calcola il prodotto di due numeri non negativi, utilizzando esclusivamente
 l'operazione di somma}

   VAR
      x, y: integer;
      prodotto, cont: integer;

BEGIN {prod}
   {fase di lettura}
   write('Inserire due numeri non negativi ');
   readln(x, y);
   WHILE (x < 0) OR (y < 0) DO
      BEGIN
         writeln('Errore: i due numeri non possono essere negativi');
         write('Ripetere l''inserimento ');
         readln(x, y)
      END; {while}

   {fase di calcolo}
   prodotto := 0;
   cont := 1;
   WHILE cont <= y DO
      BEGIN
         prodotto := prodotto + x;
```

```
         cont := cont + 1
      END; {while}

   {fase di scrittura}
   writeln('Il prodotto e'' uguale a ', prodotto : 1)
END. {prod}
```

Estendiamo ora il programma precedente al fine di calcolare il prodotto anche di numeri negativi. In realtà, se il valore di `x` è negativo, si potrebbe utilizzare il ciclo già scritto, senza bisogno di alcuna modifica. Se invece il valore di `y` è negativo, occorrerebbe utilizzare un ciclo "all'indietro", come:

```
   prodotto := 0;
   cont := 1;
   WHILE cont >= y DO
      BEGIN
         prodotto := prodotto - x;
         cont := cont - 1
      END {while}
```

Per evitare di introdurre un altro ciclo, utilizziamo una strategia differente. Osserviamo che il valore assoluto del prodotto di due numeri è uguale al prodotto dei loro valori assoluti, mentre il segno è positivo se i due numeri hanno lo stesso segno, negativo se hanno segni diversi.

L'idea è dunque di scrivere un algoritmo che, dopo la lettura dei dati, compia le seguenti operazioni:

> calcola e memorizza il segno del risultato
> sostituisci i valori di `x` e `y` con i rispettivi valori assoluti
> calcola il prodotto dei valori assoluti
> assegna al prodotto il segno memorizzato

Per memorizzare il segno del risultato introduciamo una variabile di tipo `boolean` e di nome `negativo`. Questa variabile dovrà contenere `true` se il risultato ha segno negativo o, in altre parole, se i segni di `x` e `y` sono discordi. I due segni sono discordi se `x < 0` e `y > 0` oppure se `x > 0` e `y < 0`, o, piú brevemente, se i due test `x > 0` e `y > 0` forniscono un risultato differente. Pertanto, l'operazione calcola e memorizza il segno del risultato può essere realizzata utilizzando il seguente assegnamento:

```
   negativo := (x > 0) <> (y > 0)
```

L'operazione successiva, cioè la sostituzione dei valori di `x` e di `y` con i rispettivi valori assoluti, può essere realizzata ricorrendo alla funzione `abs`.

Per il calcolo del prodotto ricorriamo al ciclo già presentato, che lascia nella variabile `prodotto` il risultato. Infine, se `negativo` contiene `true`, cambiamo segno a tale risultato:

```
   IF negativo THEN prodotto := -prodotto
```

Il testo del programma completo è il seguente:

```
PROGRAM prod (input, output);

{calcola il prodotto di due numeri utilizzando esclusivamente l'operazione
 di somma}

   VAR
```

```
      x, y: integer;
      prodotto, cont: integer;
      negativo: boolean;

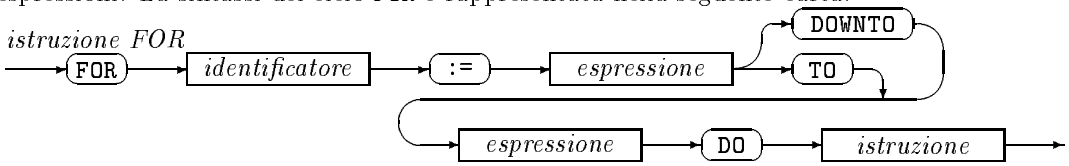
BEGIN {prod}
   {fase di lettura}
   write('Inserire due numeri interi ');
   readln(x, y);

   {fase di calcolo}
   {memorizza il segno}
   negativo := (x > 0) <> (y > 0);
   {sostituisce i numeri con i valori assoluti}
   x := abs(x);
   y := abs(y);
   {calcola il prodotto dei valori assoluti}
   prodotto := 0;
   cont := 1;
   WHILE cont <= y DO
      BEGIN
         prodotto := prodotto + x;
         cont := cont + 1
      END; {while}
   {pone l'eventuale segno meno davanti al risultato}
   IF negativo THEN
      prodotto := -prodotto;

   {fase di scrittura}
   writeln('Il prodotto e'' uguale a ', prodotto : 1)
END. {prod}
```

7.3 Il ciclo FOR

Oltre ai cicli **REPEAT** e **WHILE**, il Pascal dispone di un altro costrutto iterativo, il ciclo **FOR**, che permette di ripetere l'esecuzione di un'istruzione un certo numero di volte. Tale numero (che può essere anche zero) viene stabilito *immediatamente prima* dell'esecuzione del ciclo, valutando due espressioni. La sintassi del ciclo **FOR** è rappresentata nella seguente carta:



L'identificatore specificato dopo la parola riservata **FOR** è l'identificatore della *variabile di controllo* del ciclo, cioè della variabile utilizzata per contare il numero di iterazioni prestabilito. Tale variabile *deve essere di un tipo scalare* qualsiasi (compresi i tipi enumerativi). I valori delle due espressioni devono essere di un tipo compatibile con quello della variabile.

Descriviamo ora la modalità di esecuzione del ciclo **FOR**, nel caso in cui tra le due espressioni sia utilizzata la parola riservata **TO**:

1. vengono valutate entrambe le espressioni; il risultato della prima espressione viene assegnato alla variabile di controllo, il secondo viene memorizzato (ad esempio in un registro o in una variabile temporanea);

2. se il valore della variabile di controllo è maggiore del risultato della seconda espressione, precedentemente memorizzato, il ciclo termina e l'esecuzione presegue con le istruzioni che seguono il costrutto `FOR`, altrimenti l'esecuzione prosegue con l'istruzione scritta dopo la parola `DO`;

3. dopo l'esecuzione dell'istruzione, viene aggiornato il contenuto della variabile di controllo, sostituendolo con il relativo successore, e si ripete dal passo precedente.

È importante ricordare che:

- la variabile di controllo deve essere di un tipo scalare e deve essere dichiarata nello *stesso* modulo (programma o sottoprogramma) che contiene il ciclo `FOR`;
- il valore della variabile di controllo non deve essere alterato (mediante ad esempio assegnamenti o letture da input) da istruzioni all'interno del ciclo;
- in uscita il valore della variabile di controllo è indefinito (ciò è dovuto al fatto che per ragioni di efficienza, spesso, durante l'esecuzione del ciclo, la variabile di controllo viene spostata in uno dei registri della CPU);
- la seconda espressione viene valutata solo una volta, immediatamente prima di eseguire il ciclo; dunque, il numero di ripetizioni del ciclo (che può anche essere zero) viene fissato all'inizio; pertanto, se la seconda espressione contiene delle variabili, il cui valore viene modificato all'interno del ciclo, il numero di ripetizioni del ciclo resta in ogni caso quello calcolato all'inizio.

Si noti che, se il valore della prima espressione è maggiore del valore della seconda espressione, l'istruzione contenuta nel ciclo non verrà eseguita, e si passerà subito all'esecuzione della parte successiva di programma.

La parola riservata `TO`, utilizzata tra le due espressioni, indica che il ciclo è *in avanti*. Per utilizzare un ciclo *all'indietro*, in cui cioè ad ogni iterazione il contenuto della variabile di controllo viene sostituito con il predecessore, e in cui si esce dal ciclo quando il valore della variabile di controllo è minore del valore della seconda espressione, calcolato all'inizio del ciclo, è sufficiente sostituire la parola `TO` con la parola `DOWNTO`.

Il ciclo `FOR` è utilizzabile quando il numero di iterazioni sia noto prima dell'esecuzione del ciclo. Ad esempio, il programma per il calcolo del prodotto di due numeri, può essere riscritto come segue, facendo uso di un ciclo `FOR`:

```
PROGRAM prod (input, output);

{calcola il prodotto di due numeri utilizzando esclusivamente l'operazione }
 di somma}

   VAR
      x, y: integer;
      prodotto, cont: integer;
      negativo: boolean;

BEGIN {prod}
   {fase di lettura}
   write('Inserire due numeri interi ');
   readln(x, y);

   {fase di calcolo}
```

```
   {memorizza il segno}
   negativo := (x > 0) <> (y > 0);
   {sostituisce i numeri con i valori assoluti}
   x := abs(x);
   y := abs(y);
   {calcola il prodotto dei valori assoluti}
   prodotto := 0;
   FOR cont := 1 TO y DO
      prodotto := prodotto + x;
   {pone l'eventuale segno meno davanti al risultato}
   IF negativo THEN
      prodotto := -prodotto;

   {fase di scrittura}
   writeln('Il prodotto e'' uguale a ', prodotto : 1)
END. {prod}
```

Esempio: stampa dei codici ASCII

Il seguente programma stampa un elenco delle lettere maiuscole e minuscole, con i relativi codici ASCII. L'elenco delle maiuscole viene stampato in ordine, dalla `'A'` alla `'Z'`, quello delle minuscole in ordine inverso:

```
PROGRAM alfabeto (input, output);

   VAR
      lettera: char;

BEGIN {alfabeto}
   writeln('lettera  cod. ASCII');
   writeln;
   FOR lettera := 'A' TO 'Z' DO
      writeln(lettera : 4, ord(lettera) : 10);
   writeln;

   writeln('lettera  cod. ASCII');
   writeln;
   FOR lettera := 'z' DOWNTO 'a' DO
      writeln(lettera : 4, ord(lettera) : 10)

END. {alfabeto}
```

Ecco lo stesso programma, riscritto sostituendo al posto dei cicli `FOR` due cicli `WHILE` equivalenti:

```
PROGRAM alfabeto (input, output);

   VAR
      lettera: char;

BEGIN {alfabeto}
   writeln('lettera  cod. ASCII');
   writeln;
   lettera := 'A';
```

```
   WHILE lettera <= 'Z' DO
      BEGIN
         writeln(lettera : 4, ord(lettera) : 10);
         lettera := succ(lettera)
      END; {while}
   writeln;

   writeln('lettera  cod. ASCII');
   writeln;
   lettera := 'z';
   WHILE lettera >= 'a' DO
      BEGIN
         writeln(lettera : 4, ord(lettera) : 10);
         lettera := pred(lettera)
      END {while}

END. {alfabeto}
```

Esercizi

1. Si costruiscano due programmi equivalenti al seguente programma, nei quali il ciclo FOR sia sostituito rispettivamente da un ciclo WHILE e da un ciclo REPEAT.

   ```
   PROGRAM p (input,output);
   VAR n,i: integer;
   BEGIN
      readln(n);
      FOR i := 1 TO n DO
         writeln(i);
      writeln('Fine programma')
   END.
   ```

2. Scrivere un programma che dato un elenco di numeri ne determini il minimo. Nella prima versione del programma il numero dei numeri da leggere viene inserito come primo dato da input; nella seconda versione del programma si utilizza il numero `0` per indicare la fine della sequenza.

3. Scrivere un programma che dato un elenco di numeri ne determini:

   - la somma totale,
   - la somma di quelli di posto dispari,
   - la somma di quelli di posto pari,
   - il massimo tra le due somme precedenti.

4. Per ognuno degli esercizi delle lezioni 3 e 4 in cui si chiedeva la costruzione di un algoritmo, scrivere un programma Pascal basato sull'algoritmo richiesto.

5. Dopo avere scritto le dichiarazioni di variabile opportune, riscrivere ognuno dei tre frammenti di codice seguenti sostituendo i cicli `REPEAT` e `FOR` con ciclo `WHILE`.

   - ```
     FOR i:= succ(i) TO 'Z' DO
         write(i)
     ```
     ```

- ```
  FOR i:= i+1 DOWNTO j DO
      writeln(i)
  ```
- ```
  REPEAT
      x:= x+1;
      writeln(x)
  UNTIL x >= 10
  ```

6. Riscrivere il programma per la stampa delle lettere dell'alfabeto, sostituendo il primo ciclo `FOR` con un ciclo `WHILE` e il secondo ciclo `FOR` con un ciclo `REPEAT`.

7. Nei seguenti frammenti di programma, vengono utilizzati dei cicli `FOR`. Alcuni di questi frammenti utilizzano il ciclo in maniera impropria. Si individuino tali frammenti spiegando perché non sono corretti. Si riscrivano invece i frammenti corretti sostituendo al posto del ciclo `FOR` un ciclo `WHILE` equivalente (si supponga che le variabili `i,x` siano di tipo `integer` e che la variabile `a` sia di tipo `char`). Inoltre, per ogni frammento corretto, si stabilisca (in funzione dei valori delle variabili prima dell'esecuzione del frammento) quante volte verrà eseguita l'istruzione contenuta nel frammento.

   - ```
     FOR i := 1 TO 10 DO
     BEGIN
        writeln(i);
        i := i + 2
     END
     ```
   - ```
     FOR i := 1 TO 10 DO
        x := x + i;
     writeln(x,i)
     ```
   - ```
     FOR a := 'z' DOWNTO 'a' DO
        write(a);
     writeln('a')
     ```
   - ```
     FOR a := 'z' DOWNTO 'a' DO
        write(a);
     writeln(a)
     ```
   - ```
     FOR i := 1 TO ord(a) DO
        x := x + i
     ```
   - ```
     FOR i := 1 TO ord(a) DO
        a := chr(i)
     ```
   - ```
     FOR i := 1 TO ord(a) DO
        i := ord(a)
     ```
   - ```
     FOR i := 1 TO x DO
        x := x + i;
     writeln(x)
     ```

8. Si considerino i seguenti programmi. Per ciascuno di essi indicare se ci saranno errori in fase di compilazione, se ci saranno errori in fase d'esecuzione (in tal caso indicare dei valori di ingresso che causino errore), se il programma può entrare in un ciclo infinito (in tal caso indicare dei valori di ingresso per cui ciò avviene). Giustificare le risposte. Non si tenga conto degli errori d'esecuzione che si potrebbero avere durante l'esecuzione della procedura `readln`, ad esempio a causa dell'inserimento di lettere in operazioni di lettura di interi.

   - ```
     PROGRAM p1 (input, output);
     ```
     ```

```
   VAR
      i, x: integer;

BEGIN
   readln(x);
   FOR i := 10 TO x DO
      writeln(i)
END.
```

- `PROGRAM p2 (input, output);`

```
   VAR
      i, x: integer;

BEGIN
   readln(x);
   i := 10;
   WHILE i <= x DO
      BEGIN
         writeln(i);
         i := succ(i)
      END
END.
```

- `PROGRAM p3 (input, output);`

```
   VAR
      i, x: integer;

BEGIN
   readln(x);
   i := 10;
   WHILE i <> x + 1 DO
      BEGIN
         writeln(i);
         i := succ(i)
      END
END.
```

- `PROGRAM p4 (input, output);`

```
   VAR
      i, x: integer;

BEGIN
   readln(x);
   i := 10;
   REPEAT
      writeln(i);
      i := succ(i)
   UNTIL i > x
END.
```

- `PROGRAM p5 (input, output);`

```
   VAR
      i, x: real;

BEGIN
   readln(x);
   FOR i := 10 TO x DO
      writeln(i)
END.
```

- ```
  PROGRAM p6 (input, output);

     VAR
        i: integer;
        x: real;

  BEGIN
     readln(x);
     FOR i := 10 TO x DO
        writeln(i)
  END.
  ```
- ```
  PROGRAM p7 (input, output);

     VAR
        i, x: integer;

  BEGIN
     readln(x);
     FOR i := 10 TO x DO;
        writeln(i)
  END.
  ```
- ```
  PROGRAM p8 (input, output);

     VAR
        x, y, r: integer;
        c: char;
        errore: boolean;

  BEGIN
     readln(x, y);
     readln(c);
     r := 0;
     CASE c OF
        '+':
           r := x + y;
        '-':
           r := x - y;
        '*':
           r := x * y;
        '/':
           IF y = 0 THEN
              errore := true
           ELSE
  ```
  ```

```
            r := x DIV y
      END;

      IF errore THEN
         writeln('Operazione impossibile')
      ELSE
         writeln(r)
   END.
```

- ```
  PROGRAM p9 (input, output);

     VAR
        x, y: integer;

   BEGIN
     readln(x, y);
     IF y = 0 THEN
        writeln('Operazione impossibile');
     ELSE
        writeln(x DIV y)
   END.
  ```

- ```
  PROGRAM p10 (input, output);

     VAR
        x, y: integer;

   BEGIN
     readln(x, y);
     IF y = 0 THEN
        writeln('Operazione impossibile')
     ELSE
        writeln(x / y)
   END.
  ```

- ```
  PROGRAM p11 (input, output);

     VAR
        x, y, r: integer;

   BEGIN
     readln(x, y);
     IF y = 0 THEN
        writeln('Operazione impossibile')
     ELSE
        BEGIN
           r := x / y;
           writeln(r)
        END
   END.
  ```

- ```
  PROGRAM p12 (input, output);

     VAR
        x, y, r: integer;
  ```

```
BEGIN
   readln(x, y);
   IF y <= 0 THEN
      writeln('Operazione impossibile')
   ELSE
      BEGIN
         r := x MOD y;
         writeln(r)
      END
END.
```

- ```
  PROGRAM p13 (input, output);

     VAR
        c: integer;

  BEGIN
     REPEAT
        readln(c)
     UNTIL (c >= 0) AND (c <= 3);
     CASE c OF
        0:
           writeln('zero');
        1:
           writeln('uno');
        2:
           writeln('due');
        3:
           writeln('tre')
     END
  END.
  ```

- ```
  PROGRAM p14 (input, output);

     VAR
        c: real;

  BEGIN
     REPEAT
        readln(c)
     UNTIL (c >= 0) AND (c <= 3);
     CASE c OF
        0:
           writeln('zero');
        1:
           writeln('uno');
        2:
           writeln('due');
        3:
           writeln('tre')
     END
  END.
  ```

- ```
  PROGRAM p15 (input, output);

     VAR
        x, c: char;

  BEGIN
     REPEAT
        readln(c)
     UNTIL (c >= 'a') AND (c <= 'z');
     FOR x := c TO pred(c) DO
        writeln(x)
  END.
  ```
- ```
  PROGRAM p16 (input, output);

     VAR
        x, c: char;

  BEGIN
     REPEAT
        readln(c)
     UNTIL (c >= 'a') AND (c <= 'z');
     FOR x := c DOWNTO pred(c) DO
        writeln(x)
  END.
  ```
- ```
  PROGRAM p17 (input, output);

     VAR
        x, y: integer;

  BEGIN
     readln(x);
     y := 1;
     REPEAT
        x := x - 2;
        y := y + 1;
        writeln(x, y)
     UNTIL x = y
  END.
  ```

```
PROGRAM p18 (input, output);

   VAR
      x, y: integer;

BEGIN
   REPEAT
      readln(x)
   UNTIL (x >= 1);
   WHILE x <> 1 DO
      BEGIN
         writeln(x);
         x := x DIV 2
```
```

```
        END
    END.
```

9. Si consideri il seguente programma

```
PROGRAM p (input,output);
VAR int: integer;
    pari, dispari, grande: boolean;

BEGIN
   REPEAT
      write('inserire un numero intero ');
      readln(int);
      dispari := int MOD 2 = 1;
      pari := NOT dispari;
      grande := int > 100;
      writeln(int)
   UNTIL dispari AND grande OR pari;
   writeln('Fine programma')
END.
```

   A che punto della sequenza di input `98`, `99`, `100`, `101`,... termina il ciclo? Si deduca il comportamento del programma, e si scriva un programma equivalente che risulti piú leggibile.

10. Individuate il comportamento dei seguenti programmi:

```
PROGRAM conta1(input,output);
VAR numero: integer;
BEGIN
   numero := 0;
   WHILE numero < 4 DO
      numero := numero+1;
      writeln(numero)
END.

PROGRAM conta2(input,output);
VAR numero: integer;
BEGIN
   numero := 0;
   WHILE numero < 4 DO
      writeln(numero);
      numero := numero+1
END.

PROGRAM conta3(input,output);
VAR numero: integer;
BEGIN
   numero := 0;
   WHILE numero < 4 DO;
      numero := numero+1;
      writeln(numero)
END.
```

Si scriva un programma che, utilizzando un ciclo `WHILE`, stampi i numeri da `1` a `4`.

11. Si riscriva il seguente programma:

    - sostituendo il ciclo `WHILE` con un ciclo `FOR`;
    - sostituendo il ciclo `WHILE` con un ciclo `REPEAT`.

```
PROGRAM p(input,output);
VAR a,b: integer;
BEGIN
   readln(a,b);
   WHILE (a<=b) AND (a<100) DO
   BEGIN
      writeln(a);
      a:=a+1
   END
END.
```

12. Per ognuno dei seguenti frammenti di programma dire se è possibile sostituire i cicli `WHILE` con cicli `FOR` motivando la risposta. In caso affermativo, riscrivere il codice dopo la sostituzione.

    - ```
      i := k;
      WHILE i <n DO
      BEGIN
         writeln(i);
         i := i + 1
      END
      ```
    - ```
      i := k;
      WHILE i < n DO
      BEGIN
         writeln(i);
         i := i + 3
      END
      ```
    - ```
      i := k;
      WHILE i < n DO
      BEGIN
         writeln(i);
         i := i + 1
         readln(n)
      END
      ```

13. Per ognuno dei seguenti frammenti di codice individuare delle dichiarazioni di variabile ed eventualmente di tipo in modo che le istruzioni che vi appaiono risultino corrette dal punto di vista della compatibilità dei tipi. Se ciò non fosse possibile spiegare il motivo.

    - ```
      a := 10;
      z := 100;
      FOR i := 'a' TO 'z' DO
         writeln(succ(i))
      ```
    - ```
      a := 10;
      z := 100;
      FOR i := a TO z DO
         writeln(succ(i))
      ```
      ```

- ```
  a := 10;
  z := 100;
  FOR i := a TO 'z' DO
     writeln(succ(i))
  ```

14. Per ognuno dei seguenti frammenti di codice, scrivere un frammento equivalente utilizzando esclusivamente sequenza, selezione e cicli `WHILE` (le variabili `x, y` sono di tipo `integer`, `a, b` di tipo `char`):

    - ```
      REPEAT
         x := x - 1;
         y := y + 2
      UNTIL (x <= 0) AND (y > x)
      ```
    - ```
      REPEAT
         writeln(x, y);
         x := y DIV 2;
         y := x DIV 2
      UNTIL (x <> 4) OR (y < 3)
      ```
    - ```
      REPEAT
         writeln(a);
         a := pred(a)
      UNTIL a < b
      ```
    - ```
      REPEAT
         a := succ(a);
         b := pred(b);
         writeln(a, b)
      UNTIL (b < 'a') OR (a > 'z')
      ```

15. Per ognuno dei seguenti frammenti di codice, scrivere un frammento equivalente utilizzando esclusivamente sequenza, selezione e cicli `REPEAT` (le variabili `x, y` sono di tipo `integer`, `a, b` di tipo `char`):

    - ```
      WHILE x <> y DO
      BEGIN
         x := x + 1;
         y := y - 2
      END
      ```
    - ```
      WHILE a = b DO
         readln(a)
      ```
    - ```
      WHILE (x > y * y) OR (y < 150) DO
      BEGIN
         x := x DIV 2;
         y := y * y
      END
      ```