Compitino 1
28–29 ottobre 1999

Correzione del compitino del Turno 1

Esercizio 1

Utilizzando *esclusivamente* variabili di *tipo semplice*, scrivere un programma che legga da input una sequenza di numeri interi positivi e negativi e scriva in output:

a. il valore del piú grande multiplo di `3` contenuto nella sequenza;
b. la media dei numeri pari presenti nella sequenza.

Si supponga che l'inserimento di `0` indichi la fine della sequenza (`0` non deve essere considerato come elemento della sequenza). Nel caso le quantità richieste non possano essere calcolate, il programma dovrà fornire un messaggio d'errore.

Il programma deve elaborare una sequenza di numeri interi ricevuti da input, in cui `0` indica la fine della sequenza. Viste le richieste del problema, non occorre memorizzare tutti i numeri letti (cosa tra l'altro impossibile utilizzando solo variabili di tipo semplice), ma bisogna elaborare i singoli valori, man mano vengono ricevuti. È dunque possibile adottare uno dei seguenti schemi iterativi:

```
SCHEMA 1

inizializzazione variabili
leggi un numero
WHILE numero <> 0 DO
BEGIN
   operazioni su numero
   leggi un numero
END {while}
elaborazioni finali e scrittura del risultato

SCHEMA 2

inizializzazione variabili
leggi un numero
```

```
REPEAT
   IF numero <> 0 THEN
       BEGIN
         operazioni su numero
         leggi un numero
       END {if}
UNTIL numero = 0
elaborazioni finali e scrittura del risultato
```

```
SCHEMA 3

inizializzazione variabili
leggi un numero
IF numero <> 0 THEN
   REPEAT
      operazioni su numero
      leggi un numero
   UNTIL numero = 0
elaborazioni finali e scrittura del risultato
```

Si noti, nel secondo e nel terzo schema, l'uso del costrutto `IF` per evitare che le operazioni vengano eseguite nel caso in cui venga inserito subito `0`.

Scegliamo di adottare il primo schema che appare, tra i tre, il piú semplice.

Per il calcolo della media dei numeri pari letti occorre ricordare la somma dei numeri pari e il numero di numeri pari presenti nella sequenza. Il calcolo della media verrà effettuato fuori dal ciclo, nella fase di elaborazione finale. Pertanto introduciamo due variabili `sommapari` e `npari` di tipo `integer`, in cui appunto accumulare la somma e il numero di numeri pari, che andranno inizializzate a zero, e una variabile `mediapari` di tipo `real` in cui porre il valore della media. Dovendo calcolare solo la media dei numeri pari letti e tenendo conto del fatto che per calcolare la media occorre avere letto almeno un numero, si ottiene un codice con la seguente struttura:

```
sommapari := 0;
npari := 0;
readln(numero);
WHILE numero <> 0 DO
BEGIN
   IF numero MOD 2 = 0
      THEN
         BEGIN
            sommapari := sommapari + numero;
            npari := npari + 1
         END;
   readln(numero)
END; {while}
IF npari <> 0
   THEN
      BEGIN
         mediapari := sommapari / npari;
         scrittura del risultato
      END
   ELSE messaggio d'errore
```

Studiamo ora, indipendentemente dal calcolo della media, il problema del calcolo del piú grande multiplo di `3` contenuto nella sequenza.

Occorre utilizzare una variabile per ricordare il massimo tra i multipli di `3` sinora letti. Chiameremo tale variabile, di tipo `integer`, `max`. Ogni volta che leggiamo un nuovo numero, se esso è multiplo di `3` ne confrontiamo il valore con il contenuto di `max`, aggiornando eventualmente il contenuto di `max`. È necessario assegnare un valore iniziale a `max`. Una prima soluzione consiste nell'inizializzare `max` con `-maxint - 1`, che corrisponde al piú piccolo intero rappresentabile. In questo caso il codice avrà il seguente schema:

```
max := -maxint - 1;
readln(numero);
WHILE numero <> 0 DO
BEGIN
   IF (numero MOD 3 = 0) AND (numero > max)
      THEN max := numero;
   readln(numero)
END; {while}
IF max = -maxint - 1
   THEN messaggio d'errore
   ELSE scrittura di max
```

Una soluzione alternativa, piú elegante, consiste nell'utilizzare una variabile `primo` di tipo `boolean` che valga `true` fino a quando non è stato ancora trovato il primo multiplo di `3`. In questo caso, `max` viene inizializzata con il valore del primo multiplo di `3` che viene trovato. In altre parole si utilizza il seguente schema:

```
primo := true;
readln(numero);
WHILE numero <> 0 DO
BEGIN
   IF (numero MOD 3 = 0)
       THEN
          IF primo THEN
                       BEGIN
                          max := numero;
                          primo := false
                       END
                    ELSE
                       IF numero > max
                          THEN max := numero;
   readln(numero)
END; {while}
IF primo
   THEN messaggio d'errore
   ELSE scrittura di max
```

A questo punto, per ottenere un programma che soddisfi alle specifiche date, possiamo fondere i due cicli, per il calcolo della media e del minimo, in un unico ciclo. Ecco il listato completo, in cui abbiamo scelto, per il calcolo del minimo, la seconda soluzione proposta.

```
PROGRAM compito1 (input, output);

   VAR
      sommapari, npari, numero, max: integer;
      mediapari: real;
      primo: boolean;
```

```
BEGIN
   sommapari := 0;
   npari := 0;
   primo := true;

   writeln('Inserire un numero intero (0 per terminare)');
   readln(numero);
   WHILE numero <> 0 DO
      BEGIN
         {operazioni utili per il calcolo successivo della media}
         IF numero MOD 2 = 0 THEN
            BEGIN
               sommapari := sommapari + numero;
               npari := npari + 1
            END;

         {operazioni utili per il calcolo del massimo}
         IF numero MOD 3 = 0 THEN
            IF primo THEN
               BEGIN
                  max := numero;
                  primo := false
               END
            ELSE IF numero > max THEN
               max := numero;

         {lettura del prossimo numero da esaminare}
         writeln('Inserire un numero intero (0 per terminare)');
         readln(numero)
      END; {while}

   {calcolo e scrittura della media}
   IF npari <> 0 THEN
      BEGIN
         mediapari := sommapari / npari;
         writeln('La media dei numeri pari inseriti e'' ', mediapari : 6 : 2);
      END
   ELSE
      writeln('Non sono stati inseriti numeri pari');

   {scrittura del massimo}
   IF primo THEN
      writeln('Non sono stati inseriti multipli di 3')
   ELSE
      writeln('Il massimo tra i multipli di 3 inseriti e'' ', max : 1)
END.
```

Esercizio 2

Siano `x` e `y` due variabili di tipo `integer`. Si esprimano in linguaggio Pascal, senza ricorrere all'operatore `NOT`, le seguenti condizioni:

- `y < x ≤ y + 10`
- la negazione della condizione precedente.

Nella prima condizione si richiede che il valore di `x` sia contemporaneamente maggiore di quello di `y` e minore oppure uguale a quello di `y + 10`. Pertanto, la condizione può essere facilmente espressa come segue, facendo uso degli operatori di confronto e dell'operatore `AND`:

```
(x > y) AND (x <= y + 10)
```

La negazione della condizione precedente è `NOT((x > y) AND (x <= y + 10))` che, mediante le leggi di De Morgan, può essere riscritta come `NOT(x > y) OR NOT(x <= y + 10)`, che, eliminando i `NOT`, è equivalente a:

```
(x <= y) OR (x > y + 10)
```

Esercizio 3

Nei seguenti frammenti di programma vengono utilizzati dei cicli `FOR`. Si individuino i frammenti in cui il ciclo è utilizzato in maniera impropria, indicando il motivo. Si riscrivano invece i frammenti corretti sostituendo al posto del ciclo `FOR` un ciclo `WHILE` equivalente (le variabili `somma`, `i`, `x`, `n` sono di tipo `integer`):

- ```
  somma := 0;
  FOR i := 1 TO n DO
     BEGIN
        readln(x);
        IF x = 0 THEN i := n + 1
                 ELSE somma := somma + x
     END;
  writeln(somma)
  ```

- ```
  somma := 0;
  FOR i := 1 TO n DO
     BEGIN
        readln(x);
        IF x = 0 THEN n := n + i
                 ELSE somma := somma + x
     END;
  writeln(somma)
  ```

All'interno del primo frammento vi è un assegnamento alla variabile di controllo del ciclo. Pertanto il ciclo è utilizzato in maniera impropria.

Nel secondo frammento, invece, il ciclo è utilizzato in maniera corretta. Nell'esecuzione dei cicli `FOR`, l'espressione che indica la fine del ciclo, in questo caso `n`, viene valutata solo una volta, all'inizio del ciclo. Poiché il valore di `n` potrebbe essere modificato durante il ciclo, per tradurre il ciclo `FOR` in un ciclo `WHILE`, introduciamo una variabile `temp` di tipo `integer`, in cui memorizziamo il valore finale che dovrà assumere la variabile di controllo prima di uscire dal ciclo. Ecco il frammento di codice riscritto con un ciclo `WHILE`:

```
somma := 0;
temp := n;
i := 1;
WHILE i <= temp DO
```

```
   BEGIN
      readln(x);
      IF x = 0 THEN n := n + i
               ELSE somma := somma + x;
      i := i + 1
   END;
writeln(somma)
```

Esercizio 4

Si consideri il seguente frammento di programma, in cui `i` e `x` sono due variabili di tipo `integer`, mentre `c` è una variabile di tipo `char`:

```
FOR i := x DOWNTO -x DO
   FOR c := 'A' TO 'Z' DO
      writeln(i,c)
```

a Nell'ipotesi che il valore contenuto inizialmente in `x` non sia negativo, indicare, in funzione del contenuto di `x`, quante volte verrà eseguita l'istruzione `writeln(i,c)`.
b Riscrivere il frammento di codice sostituendo i due cicli `FOR` con cicli `WHILE`. Nel ciclo esterno

`i` varia da `x` a `-x`, pertanto vi sono `2x+1` iterazioni. Per ognuna di esse, mediante il ciclo interno, l'istruzione `writeln(i,c)` viene ripetuta con `c` che varia sulla sequenza delle lettere maiuscole. Pertanto l'istruzione `writeln(i,c)` verrà eseguita per un numero di volte pari a `2x+1` per il numero di lettere, cioè `26(2x+1)`.

Il ciclo esterno può essere riscritto utilizzando lo schema:

```
i := x
WHILE i >= -x DO
   BEGIN
      ...ciclo interno...
      i := i - 1
   END
```

Il ciclo interno, a sua volta, può essere riscritto come:

```
c := 'A';
WHILE c <= 'Z' DO
   BEGIN
      writeln(i, c);
      c := succ(c)
   END
```

Pertanto, il frammento di codice può essere riscritto come:

```
i := x
WHILE i >= -x DO
   BEGIN
   c := 'A';
   WHILE c <= 'Z' DO
      BEGIN
         writeln(i, c);
         c := succ(c)
      END;
      i := i - 1
   END
```

Esercizio 5

Si considerino i seguenti programmi. Per ciascuno di essi indicare se ci saranno errori in fase di compilazione (in tal caso indicare il motivo), 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 d'ingresso per cui ciò avviene). Non si tenga conto degli errori d'esecuzione che si potrebbero avere durante la lettura dei dati.

- ```
  PROGRAM p1 (input, output);
     VAR x: real;
  BEGIN
     readln(x);
     WHILE x >= 0 DO
        BEGIN
           writeln(x);
           x := x DIV 2
        END
  END.
  ```

- ```
  PROGRAM p2 (input, output);
     VAR x, y: integer;
  BEGIN
     REPEAT
        readln(x, y)
     UNTIL x <= y;
     WHILE x <> y DO
        BEGIN
           writeln(x,y);
           x := x + 1;
           y := y - 1
        END
  END.
  ```

- ```
  PROGRAM p3 (input, output);
     VAR a, z: char;
         b: boolean;
  BEGIN
     REPEAT
        readln(a);
        b := (a < 'a');
        b := b OR (a > 'z')
     UNTIL NOT b;
     FOR z := 'q' DOWNTO a DO
        writeln(z)
  END.
  ```

Il primo programma provocherà un errore di compilazione in quanto si tenta di utilizzare l'operatore `DIV` con un valore di tipo `real`.

Il secondo programma viene compilato correttamente, ma può entrare in un ciclo infinito per alcuni valori, ad esempio inserendo `4` e `5`.

Anche il terzo programma viene compilato correttamente. Si osservi che si esce dal primo ciclo quando `b` contiene una lettera minuscola. Se la lettera è successiva, in ordine alfabetico, alla `q`, il ciclo `FOR` non verrà mai eseguito. In ogni caso, il ciclo `FOR` termina sempre.
  ```

Esercizio 6

Si considerino i seguenti frammenti di codice. Per ognuno di essi scrivere delle dichiarazioni di variabile, in modo che le istruzioni risultino corrette dal punto di vista della compatibilità dei tipi. Se ciò non fosse possibile spiegare il motivo.

- ```
  x := a = b;
  a := b MOD (ord(x) + 100)
  ```
- ```
  z := chr(x DIV (ord(z) MOD x))
  ```
- ```
  z := succ(z + pred(y)) / round(y)
  ```

Nella prima istruzione del primo frammento, si assegna il risultato di un confronto (riconoscibile dall'uso dell'operatore `=`) alla variabile `x`, che pertanto deve essere di tipo `boolean`. I valori confrontati sono quelli di `a` e di `b`, che dunque dovranno essere di due tipi confrontabili tra loro. Nella seconda istruzione si applica la funzione `ord` a `x`. Questo è possibile in quanto `boolean` è un tipo scalare. Al risultato di `ord`, sempre di tipo `integer`, viene sommato 100. Il nuovo risultato, ancora di tipo `integer` viene utilizzato come secondo operando di `MOD`. Il primo operando è invece la variabile `b` che dovrà dunque essere di tipo `integer`. Infine il risultato di `MOD` viene assegnato alla variabile `a`, che può essere o `integer` oppure `real` (in questo caso sia prima dell'assegnamento che prima del confronto sarà necessaria una conversione implicita). Una possibile soluzione è data dalle seguenti dichiarazioni:

```
VAR
   a, b: integer;
   x: boolean;
```

Nel secondo frammento si assegna a `z` il risultato della funzione `chr`. La variabile `z` deve dunque essere di tipo `char`, ciò è anche compatibile con l'espressione `ord(z)` all'interno del lato destro dell'assegnamento. Per poter applicare correttamente `DIV` e `MOD`, la variabile `x` dovrà essere di tipo `integer`:

```
VAR
   x: integer;
   z: char;
```

Infine, nell'ultimo frammento, si assegna a `z` il risultato di una divisione reale. Pertanto `z` deve essere di tipo `real`. A destra dell'assegnamento viene utilizzata la funzione `succ`, che richiede un parametro di tipo `integer`. In queso caso il parametro di `succ` è `z + pred(y)`, che è di tipo `real` in quanto `z` è `real`. Dunque, in questo caso non è possibile definire correttamente i tipi.

Esercizio 7

Riscrivere il seguente frammento di programma utilizzando esclusivamente sequenza, selezione e cicli `REPEAT`.

```
WHILE (a > b) OR (b <= 36) DO
BEGIN
   a := a - 2;
   b := b + 1
END
```

Ricordiamo che l'uscita da un ciclo `REPEAT` avviene quando la condizione diventa *vera*, mentre nel ciclo `WHILE`, quando la condizione diventa *falsa*. Pertanto, in prima approssimazione, possiamo riscrivere il ciclo, negando la condizione, come:
```

```
REPEAT
   a := a - 2;
   b := b + 1
UNTIL NOT((a > b) OR (b <= 36))
```

La condizione `NOT((a > b) OR (b <= 36))` può essere riscritta come `(a <= b) AND (b > 36)`. Osserviamo tuttavia che le istruzioni del ciclo `REPEAT` vengono eseguite sempre almeno una volta, mentre quelle del ciclo `WHILE` possono essere eseguite anche zero volte (si pensi ad esempio al caso in cui i valori iniziali di `a` e di `b` siano rispettivamente `30` e `40`). Affinché le istruzioni del ciclo `REPEAT` non vengano mai eseguite, qualora la condizione del ciclo `WHILE` risulti falsa sin dall'inizio, effettuiamo inizialmente un controllo, mediante un'istruzione `IF`:

```
IF (a > b) OR (b <= 36)
   THEN
      REPEAT
         a := a - 2;
         b := b + 1
      UNTIL (a <= b) AND (b > 36)
```

Correzione del compitino del Turno serale

Esercizio 1

Utilizzando *esclusivamente* variabili di *tipo semplice*, scrivere un programma che legga da input una sequenza di numeri interi positivi e negativi e scriva in output:

a. quanti multipli di `5` contiene la sequenza;

b. la differenza tra il piú grande e il piú piccolo numero pari presente nella sequenza.

Si supponga che l'inserimento di `0` indichi la fine della sequenza (`0` non deve essere considerato come elemento della sequenza). Nel caso le quantità richieste non possano essere calcolate, il programma dovrà fornire un messaggio d'errore.

Anche in questo caso è opportuno adottare lo schema 1 (v. es. 1 compitino turno 1).

Per il calcolo del numeo di multipli di `5` contenuti nella sequenza è sufficiente utilizzare un contatore `mul5`, inizializzato a zero e incrementato ogni volta che si incontri un multiplo di `5`: Ottieniamo un codice con la seguente struttura:

```
mul5 := 0;
readln(numero);
WHILE numero <> 0 DO
BEGIN
   IF numero MOD 5 = 0
      THEN
         mul5 := mul5 + 1;
   readln(numero)
END; {while}
scrivi il valore di mul5
```

Studiamo ora, indipendentemente dal calcolo del numero di multipli di 5, il problema del calcolo della differenza tra il piú grande e il piú piccolo dei valori pari presenti nella sequenza.

Occorre utilizzare due variabili per ricordare il massimo e il minimo tra i numeri pari sinora letti. Chiameremo tali variabili, di tipo `integer`, `max` e `min`. Ogni volta che leggiamo un nuovo numero, se esso è pari ne confrontiamo il valore con il contenuto di `max` e `min`, aggiornando eventualmente il contenuto di `max` e `min`. È necessario assegnare un valore iniziale a `max` e `min`. Una

prima soluzione consiste nell'inizializzare `max` con `-maxint - 1` e `min` con `maxint`. Una soluzione alternativa consiste nell'utilizzare una variabile `primo` di tipo `boolean` che valga `true` fino a quando non è stato ancora trovato un numero pari. In questo caso, `max` e `min` vengono inizializzate con il valore del primo numero pari che viene trovato. In altre parole si utilizza il seguente schema:

```
primo := true;
readln(numero);
WHILE numero <> 0 DO
BEGIN
   IF numero MOD 2 = 0
       THEN
          IF primo THEN
                       BEGIN
                          max := numero;
                          min := numero;
                          primo := false
                       END
                    ELSE
                       BEGIN
                          IF numero > max
                             THEN max := numero;
                          IF numero < min
                             THEN min := numero
                       END;
   readln(numero)
END; {while}
IF primo
   THEN messaggio d'errore
   ELSE scrittura della differenza tra max e min
```

A questo punto, per ottenere un programma che soddisfi alle specifiche date, possiamo fondere i due cicli, per il calcolo della media e del minimo, in un unico ciclo. Ecco il listato completo, in cui abbiamo scelto, per il calcolo del minimo, la seconda soluzione proposta.

```
PROGRAM compito1 (input, output);

   VAR
      mul5, numero, max, min: integer;
      primo: boolean;

BEGIN
   mul5 := 0;
   primo := true;

   writeln('Inserire un numero intero (0 per terminare)');
   readln(numero);
   WHILE numero <> 0 DO
      BEGIN
         {operazioni per il calcolo del numero di multipli di 5}
         IF numero MOD 5 = 0 THEN
            mul5 := mul5 + 1;
         {operazioni per il calcolo del massimo e del minimo pari}
         IF numero MOD 2 = 0 THEN
```

```
            IF primo
                 THEN
                    BEGIN
                       max := numero;
                       min := numero;
                       primo := false
                    END
                 ELSE
                    BEGIN
                       IF numero > max
                          THEN max := numero;
                       IF numero < min
                          THEN min := numero
                    END;

         {lettura del prossimo numero da esaminare}
         writeln('Inserire un numero intero (0 per terminare)');
         readln(numero)
      END; {while}

   {scrittura del numero di multipli di 5}
   writeln('Sono stati inseriti ', mul5 : 1, ' multipli di 5');

   {scrittura della differenza tra max e min}
   IF primo THEN
      writeln('Non sono stati inseriti numeri pari')
   ELSE
      writeln('La differenza tra massimo e minimo pari e'' ', max - min : 1)
END.
```

Esercizio 2

Siano `x` e `y` due variabili di tipo `integer`. Si esprimano in linguaggio Pascal, senza ricorrere all'operatore `NOT`, le seguenti condizioni:

- i valori di `x` ed `y` non sono nulli e hanno lo stesso segno;
- la negazione della condizione precedente.

La prima condizione richiede che `x` ed `y` siano entrambi positivi, oppure entrambi negativi. Pertanto può essere espressa come:

```
((x > 0) AND (y > 0)) OR ((x < 0) AND (y < 0))
```

La seconda condizione è `NOT(((x > 0) AND (y > 0)) OR ((x < 0) AND (y < 0)))`, che, utilizzando le leggi di De Morgan, equivale a
`NOT((x > 0) AND (y > 0)) AND NOT((x < 0) AND (y < 0)))`, cioè a
`((x <= 0) OR (y <= 0)) AND ((x >= 0) OR (y >= 0)))`

In alternativa, si poteva esprimere la prima condizione come

```
x * y > 0
```

la cui negazione è semplicemente

```
x * y <= 0
```

Esercizio 3

Nei seguenti frammenti di programma vengono utilizzati dei cicli `FOR`. Si individuino i frammenti in cui il ciclo è utilizzato in maniera impropria, indicando il motivo. Si riscrivano invece i frammenti corretti sostituendo al posto del ciclo `FOR` un ciclo `WHILE` equivalente (le variabili `y`, `i`, `x`, `n` sono di tipo `integer`):

- ```
  readln(n);
  FOR i := 1 TO n DO
     BEGIN
        readln(x);
        n := n + x * i;
        writeln(i,x)
      END;
  writeln(i)
  ```

- ```
  readln(n);
  FOR i := 1 TO n DO
     BEGIN
        readln(x);
        i := n + x * i;
        writeln(i,x)
      END;
  writeln(n)
  ```

Alla fine del primo frammento, si tenta di scrivere il contenuto della variabile `i` di controllo del ciclo. Questo uso è improprio.

All'interno del secondo frammento vi è un assegnamento alla variabile di controllo del ciclo. Pertanto il ciclo è utilizzato in maniera impropria.

Esercizio 4

Si consideri il seguente frammento di programma, in cui `i` e `x` sono due variabili di tipo `integer`, mentre `c` è una variabile di tipo `char`:

```
FOR i := x DOWNTO -x DO
   FOR c := '0' TO '9' DO
      writeln(i,c)
```

a Nell'ipotesi che il valore contenuto inizialmente in `x` non sia negativo, indicare, in funzione del contenuto di `x`, quante volte verrà eseguita l'istruzione `writeln(i,c)`.
b Riscrivere il frammento di codice sostituendo i due cicli `FOR` con cicli `WHILE`. Nel ciclo esterno

`i` varia da `x` a `-x`, pertanto vi sono `2x+1` iterazioni. Per ognuna di esse, mediante il ciclo interno, l'istruzione `writeln(i,c)` viene ripetuta con `c` che varia sulla sequenza delle cifre. Pertanto l'istruzione `writeln(i,c)` verrà eseguita per un numero di volte pari a `2x+1` per il numero delle cifre, cioè `10(2x+1)`.

Il ciclo esterno può essere riscritto utilizzando lo schema:

```
i := x
WHILE i >= -x DO
   BEGIN
      ...ciclo interno...
      i := i - 1
   END
```

Il ciclo interno, a sua volta, può essere riscritto come:

```
c := '0';
WHILE c <= '9' DO
   BEGIN
      writeln(i, c);
      c := succ(c)
   END
```

Pertanto, il frammento di codice può essere riscritto come:

```
i := x
WHILE i >= -x DO
   BEGIN
   c := '0';
   WHILE c <= '9' DO
      BEGIN
         writeln(i, c);
         c := succ(c)
      END;
      i := i - 1
   END
```

Esercizio 5

Si considerino i seguenti programmi. Per ciascuno di essi indicare se ci saranno errori in fase di compilazione (in tal caso indicare il motivo), 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 d'ingresso per cui ciò avviene). Non si tenga conto degli errori d'esecuzione che si potrebbero avere durante la lettura dei dati.

- ```
  PROGRAM p1 (input, output);
     VAR x: integer;
  BEGIN
     readln(x);
     WHILE x >= 0 DO
        BEGIN
           writeln(x);
           x := x DIV 2
        END
  END.
  ```

- ```
  PROGRAM p2 (input, output);
     VAR x, y: real;
  BEGIN
     REPEAT
        readln(x)
     UNTIL x = trunc(x);
     FOR y := x TO x * x DO
        BEGIN
           writeln(x,y);
           x := x + 2
        END
  END.
  ```

- ```
  PROGRAM p3 (input, output);
      VAR a, z: char;
          b: boolean;
    BEGIN
      REPEAT
        readln(a);
        b := (a < 'a');
        b: = NOT b AND (a <= 'z')
      UNTIL b;
      FOR z := a TO 'b' DO
        writeln(z)
    END.
  ```

Il primo programma viene compilato correttamente ma, sugli input positivi, entra in un ciclo infinito.

Nel secondo programma si utilizza, come variabile di controllo del ciclo `FOR`, una variabile di tipo `real`. Pertanto vi è un errore di compilazione.

Il terzo programma viene compilato correttamente. Si osservi che si esce dal primo ciclo quando `b` contiene una lettera minuscola. Se la lettera è successiva, in ordine alfabetico, alla `b`, il ciclo `FOR` non verrà mai eseguito. In ogni caso, il ciclo `FOR` termina sempre.

Esercizio 6

Si considerino i seguenti frammenti di codice. Per ognuno di essi scrivere delle dichiarazioni di variabile, in modo che le istruzioni risultino corrette dal punto di vista della compatibilità dei tipi. Se ciò non fosse possibile spiegare il motivo.

- ```
  x := ord(a >= b);
  a := ord(a) >= ord(b)
  ```
- ```
  z := chr(succ(ord(z) MOD x))
  ```
- ```
  z := ord(z) / round(y)
  ```

Nella prima istruzione del primo frammento, si assegna il risultato della funzione `ord` alla variabile `x` che pertanto deve essere di tipo `integer` o `real`. La funzione `ord` è applicata al risultato di un confronto (riconoscibile dall'uso dell'operatore `>=`) tra `a` e `b`, che pertanto devono essere confrontabili tra loro. Nella seconda istruzione si assegna il risultato del confronto tra `ord(a)` e `ord(b)` ad `a`. Ricordando che `ord` può essere applicata a tutti i tipi scalari, segue che `a` e `b` devono essere di tipo `boolean`. Pertanto, è possibile fornire le sequenti dichiarazioni:

```
VAR
   a, b: boolean;
   x: integer;
```

Nel secondo frammento si assegna a `z` il risultato della funzione `chr`. La variabile `z` deve dunque essere di tipo `char`, ciò è anche compatibile con l'espressione `ord(z)` all'interno del lato destro dell'assegnamento. Per poter applicare correttamente `MOD`, la variabile `x` dovrà essere di tipo `integer`:

```
VAR
   x: integer;
   z: char;
```

Infine, nell'ultimo frammento, si assegna a `z` il risultato di una divisione reale. Pertanto `z` deve essere di tipo `real`. A destra dell'assegnamento viene applicata a `z` la funzione `ord`, che richiede un parametro di tipo `integer`. Dunque, in questo caso non è possibile definire correttamente i tipi.

Esercizio 7

Riscrivere il seguente frammento di programma utilizzando esclusivamente sequenza, selezione e cicli `REPEAT`.

```
WHILE (b <= a) OR (b <= 1000) DO
BEGIN
   a := a DIV 2;
   b := b * b
END
```

Ricordiamo che l'uscita da un ciclo `REPEAT` avviene quando la condizione diventa *vera*, mentre nel ciclo `WHILE`, quando la condizione diventa *falsa*. Pertanto, in prima approssimazione, possiamo riscrivere il ciclo, negando la condizione, come:

```
REPEAT
   a := a DIV 2;
   b := b * b
UNTIL NOT((b <= a) OR (b <= 1000))
```

La condizione `NOT((b <= a) OR (b <= 1000))` può essere riscritta come `(b > a) AND (b > 1000)`. Osserviamo tuttavia che le istruzioni del ciclo `REPEAT` vengono eseguite sempre almeno una volta, mentre quelle del ciclo `WHILE` possono essere eseguite anche zero volte (si pensi ad esempio al caso in cui i valori iniziali di `a` e di `b` siano rispettivamente `3000` e `4000`). Affinché le istruzioni del ciclo `REPEAT` non vengano mai eseguite, qualora la condizione del ciclo `WHILE` risulti falsa sin dall'inizio, effettuiamo inizialmente un controllo, mediante un'istruzione `IF`:

```
IF (b <= a) OR (b <= 1000)
   THEN
      REPEAT
         a := a DIV 2;
         b := b * b
       UNTIL (b > a) AND (b > 1000)
```