Lezione 10
2–3 novembre 1999

Argomenti trattati
- Introduzione ai sottoprogrammi in Pascal.
- Passaggio di parametri per valore e per riferimento.

10.1 Le procedure

Una *procedura* è una parte di programma a cui è associato un nome e che può essere attivata mediante una *chiamata*. Le procedure vengono dichiarate, all'interno del programma principale, dopo le variabili, e hanno la stessa struttura del programma principale: si aprono con un'intestazione, che inizia con la parola riservata `PROCEDURE` seguita dall'identificatore della procedura, ed eventualmente da una lista di parametri. Successivamente la procedura contiene una parte di dichiarazioni, e infine la parte di codice.

Svilupperemo ora alcuni semplici programmi che fanno uso di procedure.

Come primo esempio, presentiamo un programma che legge da `input` le misure dei lati di un parallelepipedo, e scrive in `output` il volume.

È immediato strutturare il programma nelle seguenti tre fasi principali:

1. lettura da `input` dei valori della larghezza, della profondità e dell'altezza del parallelepipedo;
2. calcolo del volume;
3. scrittura in `output` del valore del volume.

Anziché collocare direttamente nel programma il codice che svilupperemo per ciascuna delle tre fasi, costruiamo tre procedure, che chiameremo rispettivamente `leggi`, `calcolavolume` e `scrivi`. Compito della procedura `leggi` è quello di ricevere i dati da `input` e porli in tre variabili di nomi `larghezza`, `profondita` e `altezza`.

La procedura `calcolavolume`, dovrà occuparsi del calcolo del volume, sulla base dei valori contenuti nelle tra variabili appena menzionate. Il valore del volume verrà lasciato in una variabile di nome `volume`.

Infine, la procedura `scrivi` dovrà riportare in `output` il valore del volume.

Le procedure possono essere *chiamate* o *invocate* semplicemente utilizzando il loro nome. In Pascal, le procedure devono essere dichiarate *prima del codice programma principale* (dopo le dichiarazioni di variabili). Pertanto, il programma avrà la seguente struttura:

```
PROGRAM parallelepipedo (input, output);

   VAR
      larghezza, profondita, altezza, volume: integer;

   ... spazio per le procedure...

BEGIN {parallelepipedo}
   leggi;
   calcolavolume;
   scrivi
END. {parallelepipedo}
```

La prima istruzione del programma è una chiamata della procedura `leggi`. Pertanto, verrà eseguito il codice di tale procedura. Al termine, si tornerà nel programma principale, eseguendo l'istruzione successiva, cioè la chiamata della procedura `calcolavolume`. Anche in questo caso, si eseguirà il codice della procedura, rientrando, al termine di esso, al programma principale, e passando, con la medesima modalità, all'istruzione successiva, cioè alla chiamata della procedura `scrivi`. Al rientro da tale procedura, l'esecuzione termina in quanto viene raggiunta la fine del programma principale.

Sviluppiamo ora le tre procedure. Una procedura si apre con un'intestazione che inizia con la parola riservata `PROCEDURE`, seguita da un identificatore, il nome della procedura. Può esserci poi una lista di parametri e una serie di dichiarazioni. Dopo le dichiarazioni, viene scritto, racchiuso tra le parole riservate `BEGIN` ed `END`, il codice della procedura. Le procedure di questo esempio, sono estremamente elementari in quanto prive di parametri e prive di dichiarazioni. Ad esempio, il testo della procedura `calcolavolume` è semplicemente:

```
   PROCEDURE calcolavolume;
   BEGIN
      volume := larghezza * profondita * altezza
   END;
```

Quando viene chiamata la procedura `calcolavolume`, viene eseguito il codice che essa contiene (in questo caso un solo assegnamento, che utilizza le variabili del programma principale). Raggiunta la fine del codice, l'esecuzione riprende dal punto successivo a quello della chiamata.

Ecco il testo del programma, completato con le tre procedure:

```
PROGRAM parallelepipedo (input, output);

   VAR
      larghezza, profondita, altezza, volume: integer;

   PROCEDURE leggi;
   BEGIN {leggi}
      write('inserire i valori della larghezza, della profondita'' e dell''altezza ');
      readln(larghezza, profondita, altezza);
      WHILE (larghezza < 0) OR (profondita < 0) OR (altezza < 0) DO
         BEGIN
            write('Dati non corretti - Ripetere l''inserimento ');
            readln(larghezza, profondita, altezza)
         END {while}
   END; {leggi}
```

```
   PROCEDURE calcolavolume;
   BEGIN {calcolavolume}
      volume := larghezza * profondita * altezza
   END; {calcolavolume}

   PROCEDURE scrivi;
   BEGIN {scrivi}
      writeln('Il volume del parallelepipedo e'' ', volume : 1)
   END; {scrivi}

BEGIN {parallelepipedo}
   leggi;
   calcolavolume;
   scrivi
END. {parallelepipedo}
```

Questo esempio mostra come le procedure aiutino ad organizzare il codice di un programma in *moduli di piccola dimensione*, ognuno dei quali è destinato a svolgere uno specifico compito.

Forniamo ora un esempio in cui evidenziamo come le procedure permettano di scrivere *codice riutilizzabile* per effettuare le stesse operazioni su dati differenti.

Scriveremo un programma che legge da `input` le misure dei lati di tre diversi parallelepipedi, scrive man mano in `output` il volume di ciascun parallelepipedo e, infine, scrive in `output` il massimo tra i tre volumi. Possiamo strutturare il programma nelle seguenti fasi:

1. lettura delle misure dei lati del primo parallelepipedo;
2. calcolo del volume del primo parallelepipedo;
3. scrittura del volume del primo parallelepipedo;
4. lettura delle misure dei lati del secondo parallelepipedo;
5. calcolo del volume del secondo parallelepipedo;
6. scrittura del volume del secondo parallelepipedo;
7. lettura delle misure dei lati del terzo parallelepipedo;
8. calcolo del volume del terzo parallelepipedo;
9. scrittura del volume del terzo parallelepipedo;
10. calcolo e scrittura del massimo tra i tre volumi.

Introduciamo delle variabili di tipo `integer` di nomi `largh1, prof1, alt1, vol1, largh2, prof2, alt2, vol2, largh3, prof3, alt3, vol3` per memorizzare larghezza, profondità, altezza e volume di ciascuno dei tre parallelepipedi.

Per calcolare il volume del primo parallelepipedo, dovremmo effettuare un assegnamento come

```
vol1 := largh1 * prof1 * alt1
```

Analogamente, per calcolare il volume del secondo parallelepipedo, dovremmo scrivere

```
vol2 := largh2 * prof2 * alt2
```

e per il terzo

```
vol3 := largh3 * prof3 * alt3
```

In altre parole, in tutti e tre i casi, effettuiamo un'operazione della forma

```
volume := larghezza * profondita * altezza
```

dove al posto di `volume` sostituiamo una delle *variabili* `vol1`, `vol2`, `vol3`, al posto di `larghezza` sostituiamo uno dei *valori* di `largh1`, `largh2`, `largh3`, e così via.

Utilizzando il meccanismo di *passaggio dei parametri*, scriviamo un'unica procedura per il calcolo del volume di un parallelepipedo, in grado di operare, di volta in volta, sui valori e sulle variabili desiderate, che saranno specificate al momento della chiamata:

```
PROCEDURE calcolavolume (larghezza, profondita, altezza: integer; VAR volume: integer);
BEGIN {calcolavolume}
   volume := larghezza * profondita * altezza
END; {calcolavolume}
```

La procedura riceve tre *valori* di tipo `integer`, che pone in tre variabili private di nome `larghezza, profondita, altezza`, e una *variabile* di tipo `integer`, che nel codice della procedura viene indicata con il nome `volume`. La procedura assegna a quest'ultima variabile il prodotto dei tre valori ricevuti.

Nell'intestazione, dopo il nome della procedura, sono stati dichiarati i *parametri formali* della procedura. Questa procedura ha quattro parametri formali, tutti di tipo `integer`. I primi tre sono parametri *per valore*, l'ultimo, preceduro dalla parola riservata `VAR`, è un parametro *per riferimento* (o *per indirizzo*, o anche *per variabile*). Nel caso dei parametri per valore, al momento della chiamata viene passato un valore, con cui la procedura inizializza le proprie variabili locali corrispondenti. Nel caso del parametro per riferimento, al momento della chiamata viene passata una variabile su cui la procedura va ad agire direttamente.

Ad esempio, supponendo di avere nelle tre variabili `largh1, prof1, alt1`, rispettivamente i valori della larghezza, profondità ed altezza del primo parallelepipedo, per assegnare alla variabile `vol1` il volume corrispondente, possiamo effettuare la chiamata

```
calcolavolume(largh1, prof1, alt1, vol1)
```

In questo modo, vengono create le tre variabili private `larghezza`, `profondita` e `altezza` della procedura `calcolavolume`. Tali variabili vengono inizializzate con i valori dei *parametri attuali* corrispondenti, cioè dei parametri utilizzati nella chiamata, in questo caso i valori di `largh1`, `prof1` e `alt1`. Il quarto parametro formale, `volume`, è per riferimento: esso rappresenta pertanto un riferimento al parametro attuale utilizzato nella chiamata, in questo caso `vol1`. Ogni operazione contenuta nella procedura che riguardi `volume` andrà ad operare *direttamente* sulla variabile `vol1`. Pertanto, il risultato dell'espressione scritta a destra del simbolo di assegnamento verrà assegnato direttamente alla variabile `vol1`.

In modo simile, possiamo riscrivere la procedura di lettura `leggi`, utilizzando tre parametri per riferimento nei quali porre i valori letti, e la procedura `scrivi`, utilizzando un parametro per il valore da scrivere.

Sviluppiamo, infine, una procedura `calcolamax` per il calcolo e la scrittura in `output` del massimo tra i tre volumi calcolati. La procedura riceve come parametri i valori dei tre volumi, e con alcune istruzioni di selezione, determina il massimo tra i tre, ponendolo in una variabile `max`. Poiché questa variabile rappresenta una quantità utilizzata solamente all'interno della procedura, è opportuno che venga dichiarata come *variabile locale* dopo l'intestazione della procedura. In tal modo la variabile viene creata al momento della chiamata della procedura e distrutta la momento del rientro dalla stessa. Il codice della procedura è:

```
PROCEDURE calcolamax (a, b, c: integer);
   VAR
      max: integer;
```

```
BEGIN {calcolamax}
   IF (a >= b) AND (a >= c) THEN
      max := a
   ELSE IF b >= c THEN
      max := b
   ELSE
      max := c;
   writeln('Il volume massimo e'' ', max)
END; {calcolamax}
```

Segue il testo completo del programma:

```
PROGRAM parallelepipedo2 (input, output);

   VAR
      largh1, prof1, alt1, vol1, largh2, prof2, alt2, vol2, largh3, prof3, alt3, vol3: integer;


   PROCEDURE leggi (VAR larghezza, profondita, altezza: integer);
   BEGIN {leggi}
      write('inserire i valori della larghezza, della profondita'' e dell''altezza ');
      readln(larghezza, profondita, altezza);
      WHILE (larghezza < 0) OR (profondita < 0) OR (altezza < 0) DO
         BEGIN
            write('Dati non corretti - Ripetere l''inserimento ');
            readln(larghezza, profondita, altezza)
         END {while}
   END; {leggi}


   PROCEDURE calcolavolume (larghezza, profondita, altezza: integer; VAR volume: integer);
   BEGIN {calcolavolume}
      volume := larghezza * profondita * altezza
   END; {calcolavolume}


   PROCEDURE scrivi (volume: integer);
   BEGIN {scrivi}
      writeln('Il volume del parallelepipedo e'' ', volume : 1)
   END; {scrivi}


   PROCEDURE calcolamax (a, b, c: integer);
      VAR
         max: integer;
   BEGIN {calcolamax}
      IF (a >= b) AND (a >= c) THEN
         max := a
      ELSE IF b >= c THEN
         max := b
      ELSE
         max := c;
      writeln('Il volume massimo e'' ', max)
```

```
   END; {calcolamax}


BEGIN {parallelepipedo2}
   writeln(' *** Primo parallelepipedo ***');
   leggi(largh1, prof1, alt1);
   calcolavolume(largh1, prof1, alt1, vol1);
   scrivi(vol1);
   writeln;
   writeln(' *** Secondo parallelepipedo ***');
   leggi(largh2, prof2, alt2);
   calcolavolume(largh2, prof2, alt2, vol2);
   scrivi(vol2);
   writeln;
   writeln(' *** Terzo parallelepipedo ***');
   leggi(largh3, prof3, alt3);
   calcolavolume(largh3, prof3, alt3, vol3);
   scrivi(vol3);
   writeln;
   calcolamax(vol1, vol2, vol3)
END. {parallelepipedo2}
```

10.2 Metodi per il passaggio dei parametri in Pascal

Negli esempi precedenti abbiamo mostrato alcune caratteristiche delle procedure del Pascal. In particolare, abbiamo considerato procedure senza parametri e procedure con parametri.

Nel caso delle procedure senza parametri, la chiamata della procedura avviene semplicemente scrivendo l'identificatore della procedura stessa. L'esecuzione prosegue con il codice della procedura, per riprendere, alla fine della procedura, dall'istruzione successiva alla chiamata.

Nel caso di procedure con parametri, al momento della chiamata avviene il passaggio dei parametri. Piú precisamente distinguiamo tra:

- *Parametri formali.*
  Sono i parametri specificati nell'intestazione della procedura. Le sequenze di identificatori precedute dalla parola `VAR` individuano i *parametri per riferimento*, le altre i *parametri per valore*. Ad esempio, nell'intestazione:

  ```
  PROCEDURE p (x, y: char; VAR z, w: integer; t: integer; k: real);
  ```

  i parametri formali `x`, `y`, `t` e `k` sono per valore, mentre i parametri formali `z` e `w` sono per riferimento.

- *Parametri attuali.*
  Sono i parametri che vengono specificati, al momento della chiamata, dopo il nome della procedura. Essi vengono associati, in base alla loro posizione, ai parametri formali della procedura.

Chiaramente i parametri attuali devono corrispondere, per numero e per tipo, ai parametri formali. Piú precisamente:

- *Parametri per valore.*
  Il parametro attuale deve essere *un'espressione* di tipo compatibile con quello del parametro formale corrispondente. Ad esempio, se il parametro formale è di tipo `real`, il parametro

attuale può essere una qualunque espressione (eventualmente anche una sola variabile) che dia un risultato `integer` o `real`.

Il parametro per valore è una variabile locale della procedura, che viene inizializzata, al momento della chiamata, con il valore dell'espressione fornita come parametro attuale.

- *Parametri per riferimento (*o *per indirizzo,* o *per variabile).*
  Il parametro attuale deve essere *una variabile dello stesso tipo* specificato per il parametro formale. In questo caso, ogni operazione riguardante il parametro formale è applicata direttamente al parametro attuale.

  Attraverso i parametri per riferimento, una procedura può dunque fornire risultati al programma che l'ha chiamata.

10.3 Le funzioni

Abbiamo visto come, mediante le `PROCEDURE`, sia possibile strutturare un programma in sottoprogrammi. Esiste un secondo tipo di sottoprogrammi, detti `FUNCTION`. Le `FUNCTION`, oltre a comunicare con l'ambiente esterno ad esse tramite i parametri, restituiscono direttamente un risultato.

La dichiarazione di una funzione si apre con un'intestazione, che inizia con la parola riservata `FUNCTION`, seguita dal nome della funzione, dalla lista dei parametri e dal tipo del risultato. In particolare, le funzioni possono restituire risultati di tipo semplice o puntatori.

Le `FUNCTION` del Pascal possono essere pensate come funzioni in senso matematico: la lista dei parametri specifica il *dominio* della funzione, il tipo del risultato specifica il *codominio.*

Ad esempio, per calcolare il volume di un parallelepipedo associamo a tre numeri, che rappresentato le lunghezze dei lati, il loro prodotto, che rappresenta il volume. Supponendo che le lunghezze siano rappresentate da numeri di tipo `integer`, anche il risultato sarà di tipo `integer`. Pertanto possiamo definire una funzione con la seguente intestazione:

```
FUNCTION volume (larghezza, profondita, altezza: integer): integer;
```

I parametri formali `larghezza, profondita, altezza` rappresentano gli ingressi della funzione, il nome `integer` scritto dopo la lista dei parametri è il tipo del risultato restituito dalla funzione.

Questa funzione potrà essere richiamata all'interno di espressioni che coinvolgono valori `integer`, specificando tre parametri interi. Ecco alcuni esempi di istruzioni contenenti chiamate alla funzione:

```
z := volume(x, w, k);
x := 2 * volume(81, 18, x + y) + 3;
IF volume(w, k, w DIV k) <> 5 THEN ...
```

Nei tre casi, all'interno di un'espressione si incontra la chiamata della funzione. Pertanto, incontrando tale chiamata, l'esecuzione passa alla funzione. Quando l'esecuzione della funzione termina, viene ripreso il calcolo dell'espressione, sostituendo al posto della chiamata il risultato fornito dalla funzione.

La funzione `volume` può essere riscritta semplicemente come segue:

```
FUNCTION volume (larghezza, profondita, altezza: integer): integer;
BEGIN
   volume := larghezza * profondita * altezza
END;
```

L'unica istruzione della funzione ha la stessa forma di un assegnamento in cui, a sinistra dell'operatore di assegnamento, viene indicato il nome della funzione stessa. Mediante questa istruzione, la funzione restituisce a chi l'ha chiamata il proprio risultato, cioè il risultato dell'espressione `larghezza * profondita * altezza`.

Una `FUNCTION` *deve* sempre contenere un'istruzione in cui viene restituito un risultato, cioè un'istruzione di assegnamento al nome della funzione. Se, per effetto di istruzioni di selezione, la computazione si divide in piú rami, ogni ramo dovrà portare alla restituizione di un valore, come ad esempio nella funzione:

```
FUNCTION valoreassoluto (n: integer): integer;
BEGIN
   IF n < 0
      THEN valoreassoluto := -n
      ELSE valoreassoluto := n
END;
```

È scorretto restituire due volte un risultato , cioè avere sullo stesso ramo di computazione due assegnamenti al nome della funzione. Sebbene la restituzione di un valore venga effettuata mediante un assegnamento al nome della funzione, il nome stesso della funzione non può essere trattato come una variabile. Come vedremo in una delle prossime lezioni, l'uso del nome della funzione in un'espressione all'interno della funzione stessa, ad esempio a destra di un assegnamento, implica una *chiamata ricorsiva* della funzione.

Riportiamo il codice completo del programma relativo ai tre parallelepipedi, in il calcolo del volume viene effettuato mediante un `FUNCTION`:

```
PROGRAM parallelepipedo3 (input, output);

   VAR
      largh1, prof1, alt1, vol1, largh2, prof2, alt2, vol2, largh3, prof3, alt3, vol3: integer;


   PROCEDURE leggi (VAR larghezza, profondita, altezza: integer);
   BEGIN {leggi}
      write('inserire i valori della larghezza, della profondita'' e dell''altezza ');
      readln(larghezza, profondita, altezza);
      WHILE (larghezza < 0) OR (profondita < 0) OR (altezza < 0) DO
         BEGIN
            write('Dati non corretti - Ripetere l''inserimento ');
            readln(larghezza, profondita, altezza)
         END {while}
   END; {leggi}


   FUNCTION volume (larghezza, profondita, altezza: integer): integer;
   BEGIN {volume}
      volume := larghezza * profondita * altezza
   END; {volume}


   PROCEDURE scrivi (volume: integer);
   BEGIN {scrivi}
      writeln('Il volume del parallelepipedo e'' ', volume : 1)
   END; {scrivi}


   PROCEDURE calcolamax (a, b, c: integer);
      VAR
```

```
        max: integer;
   BEGIN {calcolamax}
      IF (a >= b) AND (a >= c) THEN
         max := a
      ELSE IF b >= c THEN
         max := b
      ELSE
         max := c;
      writeln('Il volume massimo e'' ', max)
   END; {calcolamax}


BEGIN {parallelepipedo3}
   writeln(' *** Primo parallelepipedo ***');
   leggi(largh1, prof1, alt1);
   vol1 := volume(largh1, prof1, alt1);
   scrivi(vol1);
   writeln;
   writeln(' *** Secondo parallelepipedo ***');
   leggi(largh2, prof2, alt2);
   vol2 := volume(largh2, prof2, alt2);
   scrivi(vol2);
   writeln;
   writeln(' *** Terzo parallelepipedo ***');
   leggi(largh3, prof3, alt3);
   vol3 := volume(largh3, prof3, alt3);
   scrivi(vol3);
   writeln;
   calcolamax(vol1, vol2, vol3)
END. {parallelepipedo3}
```

Le `FUNCTION`, come le `PROCEDURE`, possono avere sia parametri per valore, che parametri per riferimento e possono accedere a variabili globali, modificandole. Tuttavia, salvo casi eccezionali, è bene scrivere `FUNCTION` che utilizzino *solo parametri per valore* e che non modifichino variabili globali. In questo modo, l'unico effetto della `FUNCTION` è quello di produrre il proprio risultato, senza che vi siano altri effetti nascosti che rendono i programmi poco leggibili.

Esempio: Calcolo del massimo comun divisore

Scriviamo ora una funzione per il calcolo del massimo comun divisore tra due numeri interi. Questa funzione deve trasformare due valori interi in ingresso (cioè ricevuti tramite i parametri) in un valore intero da fornire in uscita (come risultato). In altre parole, possiamo scrivere una funzione `mcd` che riceva in ingresso due parametri di tipo `integer` e restituisca un valore `integer`. L'intestazione della funzione sarà:

```
FUNCTION mcd (a, b: integer): integer;
```

I parametri formali per valore `a` e `b` rappresentano gli ingressi della funzione, il tipo `integer` scritto dopo la lista dei parametri rappresenta in tipo del risultato restituito dalla funzione.

Riportiamo ora il testo della funzione, basata sull'algoritmo di Euclide, scritta, per semplicità, nell'ipotesi che, al momento della chiamata, il primo parametro attuale sia sempre positivo o uguale a zero e il secondo sia sempre positivo.

```
FUNCTION mcd (a, b: integer): integer;
```

```
   {restituisce il massimo comun divisore dei propri parametri}
   {Ipotesi: a >= 0 e b > 0}
   VAR
      resto: integer;
BEGIN {mcd}
   REPEAT
      resto := a MOD b;
      a := b;
      b := resto
   UNTIL resto = 0;
   mcd := a
END; {mcd}
```

Esempio: Semplificazione di frazioni

Costruiamo ora un programma che, dati in ingresso due numeri interi, che rappresentano rispettivamente il numeratore e il denominatore di una frazione, restituisce in uscita la frazione semplificata. Ad esempio, dati i numeri `4` e `12`, il programma dovrà restituire la frazione `1/3`.

Il programma può essere suddiviso in tre fasi fondamentali: una prima fase di lettura, una fase in cui viene calcolata la frazione semplificata e una fase di scrittura del risultato.

La struttura dati fondamentale del programma è costituita da due variabili, di tipo `integer`, che conterranno rispettivamente il numeratore e il denominatore della frazione considerata. Chiameremo pertanto tali variabili `numeratore` e `denominatore`.

Svilupperemo le tre fasi indicate sopra mediante tre procedure. Prima di sviluppare ciascuna procedura, specifichiamo in modo piú preciso l'architettura generale del programma definendo l'intestazione e il compito di ciascuna procedura.

- `PROCEDURE leggidati (VAR num, den: integer);`

  Compito di questa procedura è quello di ricevere da `input` i valori del numeratore e del denominatore e di restituirli al codice chiamante, tramite i due parametri, che vengono quindi passati per riferimento. Compito della procedura sarà anche quello di garantire che i due valori siano corretti, cioè che il denominatore sia diverso da zero.

- `PROCEDURE semplifica (VAR num, den: integer);`

  Questa procedura riceve nei due parametri il numeratore e il denominatore, e restituisce, sempre negli stessi parametri, i valori del numeratore e denominatore dopo la semplificazione. Affinché i risultati elaborati dalla procedura siano disponibili al programma chiamante, il passaggio dei parametri avviene per riferimento.

- `PROCEDURE stamparisultato (num, den: integer);`

  Questa procedura scrive in uscita la frazione il cui numeratore e denominatore sono specificati nei parametri. Non dovendo comunicare risultati al codice chiamante, il passaggio dei parametri può avvenire per valore.

A questo punto, dopo avere disegnato l'architettura generale del programma, passiamo a sviluppare ognuno dei blocchi specificati.

Sviluppo del blocco `PROCEDURE leggidati`

La procedura deve ricevere da `input` due valori interi, il numeratore e il denominatore. Nel caso in cui venga fornito per il denominatore valore `0`, la procedura, dopo avere segnalato l'errore, deve richiedere di ripetere l'inserimento dei dati. Si utilizzerà pertanto un ciclo, dal quale si uscirà quando i dati risultano corretti:

```
leggi num e den
WHILE den = 0 DO
   segnala l'errore e ripeti la lettura
```

È immediato passare alla codifica:

```
PROCEDURE leggidati (VAR num, den: integer);
   {acquisisce da input i dati, controllandone la correttezza}
   {i valori letti vengono posti nei parametri}

BEGIN {leggidati}
   write('Inserire numeratore e denominatore ');
   readln(num, den);
   WHILE den = 0 DO
   BEGIN
      write('Dati non validi: ripetere l''inserimento.');
      readln(num, den)
   END {while}
END; {leggidati}
```

Dopo l'intestazione della procedura è utile scrivere un breve commento che descriva il compito della procedura e come la procedura comunica con l'ambiente esterno (in questo caso tramite i parametri).

Sviluppo del blocco `PROCEDURE semplifica`

La semplificazione di una frazione avviene dividendo il numeratore e il denominatore per il loro massimo comun divisore. Avremo pertanto bisogno di un sottoprogramma in grado di calcolare il massimo comun divisore di due numeri. Lo schema della procedura sarà dunque:

```
calcola il massimo comun divisore di num e den
dividi num e den per il massimo comun divisore
```

Introduciamo una variabile locale `m` di tipo `integer` per memorizzare il massimo comun divisore. Per il suo calcolo possiamo servirci della `FUNCTION mcd` sviluppata in precedenza.

Ricordiamo che la funzione `mcd` è stata scritta nell'ipotesi che i parametri in ingresso non siano negativi, mentre i valori di `num` e `den` potrebbero esserlo. Per risolvere questo problema osserviamo che il procedimento di semplificazione dipende solo dai valori assoluti dei numeri e non dal loro segno. Pertanto stabiliamo all'inizio, in base ai segni di `num` e `den`, quale sarà il segno del risultato. A questo scopo introduciamo una variabile `negativo` di tipo `boolean` che varrà `true` se e solo se la frazione è negativa. Possiamo poi privare `num` e `den` del segno e procedere alla semplificazione, ponendo, alla fine, l'eventuale segno meno davanti a `num`:

```
negativo := valore della condizione "la frazione e' negativa"
priva i valori di num e den del segno
assegna a m il valore del massimo comun divisore di num e den
dividi num per m, inserendo l'eventuale segno meno
dividi den per m
```

Per il calcolo del valore assoluto di `num` e di `den` si può utilizzare la `FUNCTION valoreassoluto`, scritta in precedenza, o la funzione predefinita `abs` che, ricevendo un parametro `integer`, restituisce il valore `integer`, corrispondente al valore assoluto del proprio parametro (`abs` con un `real` restituisce il `real` corrispondente al valore assoluto del proprio parametro).

A questo punto, possiamo codificare la `PROCEDURE semplifica`, al cui interno inseriremo la `FUNCTION mcd` sviluppata in precedenza:

```
PROCEDURE semplifica (VAR num, den: integer);
   {semplifica la frazione fornita tramite i parametri}
   {il risultato viene restituito nei parametri}

   VAR
      negativo: boolean;
      m: integer;

...dichiarazione della FUNCTION mcd...

BEGIN {semplifica}
   negativo := num * den < 0;
   num := abs(num);
   den := abs(den);
   m := mcd(num, den);
   IF negativo THEN
      num := -num DIV m
   ELSE
      num := num DIV m;
   den := den DIV m
END; {semplifica}
```

Sviluppo del blocco `PROCEDURE stamparisultato`

Vista la semplicità dei compiti affidati a questa procedura, ne riportiamo immediatamente il codice:

```
PROCEDURE stamparisultato (num, den: integer);
BEGIN {stamparisultato}
   writeln('La frazione semplificata e'' ', num : 1, '/', den : 1)
END; {stamparisultato}
```

Il programma completo

Riportiamo il codice completo del programma. Si osservi l'innestamento a piú livelli di sottoprogrammi.

```
PROGRAM frazioni (input, output);
   {data in input una coppia di interi, che rappresentano}
   {il numeratore e il denominatore di una frazione}
   {restituisce in output la frazione semplificata}

   VAR
      numeratore, denominatore: integer;

   PROCEDURE leggidati (VAR num, den: integer);
      {acquisisce da input i dati, controllandone la correttezza}
      {i valori letti vengono posti nei parametri}

   BEGIN {leggidati}
      write('Inserire numeratore e denominatore ');
      readln(num, den);
      WHILE den = 0 DO
      BEGIN
```

```
         write('Dati non validi: ripetere l''inserimento.');
         readln(num, den)
      END {while}
   END; {leggidati}

   PROCEDURE semplifica (VAR num, den: integer);
      {semplifica la frazione fornita tramite i parametri}
      {il risultato viene restituito nei parametri}

      VAR
         negativo: boolean;
         m: integer;

      FUNCTION mcd (a, b: integer): integer;
         {restituisce il massimo comun divisore dei propri parametri}
         {Ipotesi: a >= 0 e b > 0}

         VAR
            resto: integer;

      BEGIN {mcd}
         REPEAT
            resto := a MOD b;
            a := b;
            b := resto
         UNTIL resto = 0;
         mcd := a
      END; {mcd}

   BEGIN {semplifica}
      negativo := num * den < 0;
      num := abs(num);
      den := abs(den);
      m := mcd(num, den);
      IF negativo THEN
         num := -num DIV m
      ELSE
         num := num DIV m;
      den := den DIV m
   END; {semplifica}

   PROCEDURE stamparisultato (num, den: integer);
   BEGIN {stamparisultato}
      writeln('La frazione semplificata e'' ', num : 1, '/', den : 1)
   END; {stamparisultato}

BEGIN {frazioni}
   leggidati(numeratore, denominatore);
   semplifica(numeratore, denominatore);
   stamparisultato(numeratore, denominatore)
END. {frazioni}
```

Esercizi

1. Si simuli "manualmente" l'esecuzione del seguente programma al fine di dedurre l'output prodotto. Si verifichi poi le risposte mandandolo in esecuzione sul proprio computer.

```
PROGRAM parametri (input,output);
VAR valore, variabile: integer;

   PROCEDURE piu5 (valore: integer; VAR variabile: integer);
   BEGIN {piu5}
      writeln('All''inizio dell''esecuzione di piu5: ',valore:4,variabile:4);
      valore := valore + 5;
      variabile := variabile + 5;
      writeln('Alla fine dell''esecuzione di piu5: ',valore:4,variabile:4)
   END; {piu5}

BEGIN {parametri}
   valore := 10;
   variabile := 20;
   piu5(valore,variabile);
   writeln('Al rientro, dopo l''esecuzione di piu5: ',valore:4,variabile:4)
END. {parametri}
```

2. Il seguente programma utilizza una procedura di nome `scambia` per scambiare il contenuto di due variabili:

```
PROGRAM p1 (input, output);
   VAR
      a, b: integer;

   PROCEDURE scambia (VAR x, y: integer);
      VAR
         t: integer;

   BEGIN {scambia}
      t := x;
      x := y;
      y := t
   END; {scambia}

BEGIN {p1}
   readln(a, b);
   scambia(a, b);
   writeln(a, b)
END. {p1}
```

   Si consideri ora il programma

```
PROGRAM p2 (input, output);
   VAR
      a: integer;

   PROCEDURE scambia (VAR x, y: integer);
```

```
      VAR
         t: integer;

   BEGIN {scambia}
      t := x;
      x := y;
      y := t
   END; {scambia}

BEGIN {p2}
   readln(a);
   scambia(a, a);
   writeln(a)
END. {p2}
```

Cosa produce in output?
Si sostituisca ora la procedura `scambia` dei due programmi precedenti con la seguente e si mandino in esecuzione i programmi sul proprio computer.

```
PROCEDURE scambia (VAR x, y: integer);

BEGIN {scambia}
   x := x + y;
   y := x - y;
   x := x - y
END; {scambia}
```

Mentre il comportamento di `p1` è il medesimo, quello di `p2` risulterà sorprendentemente differente. Si simuli manualmente l'esecuzione del programma per comprendere il motivo.

3. Abbiamo visto che nel passaggio per riferimento la procedura opera direttamente sul parametro attuale. È fondamentale osservare che il passaggio per riferimento è differente dal cosidetto passaggio per "valore/risultato", non presente in Pascal, ma disponibile in altri linguaggi. Nel passaggio per "valore/risultato", il parametro formale è una variabile locale della procedura. Al momento della chiamata, il valore del parametro attuale viene copiato nel parametro formale e al momento del rientro, il valore del parametro formale viene ricopiato nel parametro attuale. Quale sarebbe il comportamento del programma `p2` precedente, con la seconda procedura di scambio, se anziché utilizzare il passaggio per riferimento del Pascal, si utilizzasse il passaggio per "valore/risultato"?

4. Riscrivere il programma per la semplificazione delle frazioni sostituendo le `FUNCTION` con delle `PROCEDURE`.

5. Scrivere una `FUNCTION` Pascal con l'intestazione

   ```
   FUNCTION potenza (x, y: integer): integer;
   ```

   che restituisca il valore del primo parametro elevato al valore del secondo (si supponga che il secondo parametro non sia negativo).

6. Si consideri la seguente intestazione di procedura:

   ```
   PROCEDURE a (x: integer; VAR y, z: integer; w: char);
   ```

Individuare tra le seguenti chiamate quelle che non sono corrette, spiegando il motivo (le variabili `n`, `k`, `h` sono di tipo `integer`, `b` è di tipo `char`).

- `a(n + 1, k, h, b)`
- `a(n, k + 1, h, b)`
- `a(ord(b), k, h, chr(n))`
- `a(k DIV h, k, h, chr(n))`
- `a(k / h, k, h, chr(n))`

7. Si consideri la seguente intestazione di procedura:

```
PROCEDURE b (x: real; VAR z: real; w: integer);
```

Per ognuna delle seguenti chiamate della procedura `b`, scrivere delle dichiarazioni di variabili ed eventualmente di tipo, in modo che la chiamata risulti corretta. Se ciò non fosse possibile spiegare il motivo.

- `b(a, p, succ(a))`
- `b(x / y, y, x)`
- `b(x DIV y, y, x)`
- `b(x, y, x DIV y)`