Lezione 23
11–12 gennaio 2000

Argomenti trattati
- Procedure e funzioni come parametri.
- Pile.
- Code.

23.1 Procedure e funzioni come parametri

Consideriamo il tipo *lista di interi*, `tipolista`, definito come segue:

```
TYPE
   tipolista = ^nodolista;
   nodolista = RECORD
         info: integer;
         pros: tipolista
      END;
```

Supponiamo di dovere scrivere un programma in cui sia possibile effettuare alcune operazioni che modificano tutti gli elementi di una lista di numeri interi, come ad esempio:

- incremento di 1 dei valori contenuti nella lista;
- sostituzione di ogni valore contenuto nella lista con il doppio;
- decremento di 1 di ogni valore contenuto della lista.

Le tre operazioni indicate possono essere realizzate utilizzando il seguente schema, eseguito a partire dal primo elemento della lista:

```
WHILE non e' stata raggiunta la fine della lista DO
BEGIN
   operazioni sul campo info dell'elemento corrente
   spostamento sull'elemento successivo
END
```

Per ciascuna delle tre operazioni indicate potremmo dunque scrivere una procedura del tipo:

```
PROCEDURE operazioni (l: tipolista);

BEGIN
   WHILE l <> NIL DO
      BEGIN
         operazione su l^.info;
         l := l^.pros
      END
END;
```

Si noti che poiché le operazioni modificano il contenuto dei nodi, e non i puntatori, il puntatore `l` può essere passato per valore.

Utilizzando lo schema della procedura `operazioni`, potremmo ora scrivere tre procedure per realizzare le operazioni richieste. Tali procedure si differenziano solo nel nome e nell'istruzione che abbiamo indicato come `operazione su l^.info`.

In alternativa, possiamo scrivere un'unica procedura che riceva come parametro la lista e una variabile che indichi l'operazione richiesta. Al posto dell'istruzione `operazione su l^.info` possiamo scrivere un'istruzione `CASE` in cui viene selezionata l'esecuzione di una tra le possibili istruzioni. Ad esempio, se è stato definito il tipo:

```
TYPE
   tipooperazioni = (incremento, decremento, doppio);
```

la procedura potrebbe essere:

```
PROCEDURE operazioni (l: tipolista; oper: tipooperazioni);

{modifica i valori contenuti nella lista puntata da l, applicando l'operazione oper}

BEGIN {operazioni}
   WHILE l <> NIL DO
      BEGIN
         CASE oper OF
            incremento:
               l^.info := l^.info + 1;
            decremento:
               l^.info := l^.info - 1;
            doppio:
               l^.info := l^.info * 2
         END; {case}
         l := l^.pros
      END
END; {operazioni}
```

Se `lista` è una variabile globale contenente il puntatore al primo elemento della lista, l'incremento di 1 del valore contenuto in ogni nodo verrà effettuato mediante la chiamata:

```
operazioni(lista, incremento)
```

Le tre operazioni da effettuare sui singoli valori possono essere scritte all'interno di singole procedure (in questo caso non vi sono particolari vantaggi, visto che le operazioni sono estremamente semplici, ma per operazioni piú complesse, che non possano essere espresse in un'unica linea di codice, l'uso di sottoprogrammi migliora la leggibilità del codice):

```
PROCEDURE PiuUno (VAR dato: integer);
BEGIN
   dato := dato + 1
END;

PROCEDURE MenoUno (VAR dato: integer);
BEGIN
   dato := dato - 1
END;

PROCEDURE PerDue (VAR dato: integer);
BEGIN
   dato := dato * 2
END;
```

La procedura `operazioni` viene dunque riscritta come:

```
PROCEDURE operazioni (l: tipolista; oper: tipooperazioni);

{modifica i valori contenuti nella lista puntata da l, applicando l'operazione oper}

BEGIN {operazioni}
   WHILE l <> NIL DO
      BEGIN
         CASE oper OF
            incremento:
               PiuUno(l^.info);
            decremento:
               MenoUno(l^.info);
            doppio:
               PerDue(l^.info)
         END; {case}
         l := l^.pros
      END
END; {operazioni}
```

Osserviamo che le tre procedure `PiuUno`, `MenoUno` e `PerDue` hanno la stessa lista di parametri (un unico parametro di tipo `integer` passato per riferimento). Inoltre, le tre chiamate all'interno dell'istruzione `CASE` si differenziano solo per il nome della procedura chiamata. In questa situazione è possibile riscrivere la procedura `operazioni`, passando ad essa, come secondo *parametro*, la procedura che realizza l'operazione desiderata:

```
PROCEDURE operazioni (l: tipolista; PROCEDURE oper (VAR x: integer));

{modifica i valori contenuti nella lista puntata da l, applicando la procedura oper}

BEGIN {operazioni}
   WHILE l <> NIL DO
      BEGIN
         oper(l^.info);
         l := l^.pros
      END
END; {operazioni}
```

Con la nuova intestazione, `operazioni` è una procedura con due parametri formali. Il primo parametro è un valore di tipo `tipolista`. Il secondo parametro è una *procedura* che riceve un unico parametro di tipo `integer`, passato per riferimento. Con l'istruzione

```
oper(l^.info)
```

viene chiamata la procedura che è stata passata come parametro attuale al momento della chiamata di `operazioni`.

Ad esempio, si supponga di eseguire la chiamata

```
operazioni(lista, PiuUno)
```

Al momento dell'esecuzione dell'istruzione `oper(l^.info)`, verrà chiamata la procedura `PiuUno`, a cui verrà passata, per riferimento, la variabile `l^.info`. Pertanto, l'effetto di `operazioni(lista, PiuUno)` è quello di applicare la procedura `PiuUno` ad ogni valore contenuto in `lista`.

Analogamente, si possono applicare le operazioni di decremento e di raddoppio di ogni elemento della lista, passando come parametri le procedure `MenoUno` e `PerDue`.

Segue il listato di un programma di prova che legge una sequenza di numeri interi, terminata con 0, la memorizza in una lista in ordine inverso rispetto a quello di inserimento, incrementa tutti gli elementi di 1, poi li raddoppia, e infine li decrementa di 1. Dopo ogni operazione viene visualizzato il contenuto della lista.

```
PROGRAM liste (input, output);

   TYPE
      tipolista = ^nodolista;
      nodolista = RECORD
            info: integer;
            pros: tipolista
         END;

   VAR
      lista: tipolista;


   PROCEDURE InserInizio (VAR l: tipolista; x: integer);

   {Inserisce il valore di x all'inizio della lista puntata da l}

      VAR
         p: tipolista;

   BEGIN {InserInizio}
      new(p);
      p^.info := x;
      p^.pros := l;
      l := p
   END; {InserInizio}


   PROCEDURE lettura (VAR l: tipolista);
      VAR
         n: integer;
   BEGIN {lettura}
```

```
   l := NIL;
   readln(n);
   WHILE n <> 0 DO
      BEGIN
         InserInizio(l, n);
         readln(n)
      END
END; {lettura}


PROCEDURE ScriviLista (l: tipolista);

{Scrive in output il contenuto della lista puntata da l}
{versione iterativa}

BEGIN {ScriviLista}
   WHILE l <> NIL DO
      BEGIN
         writeln(l^.info);
         l := l^.pros
      END {while}
END; {ScriviLista}


PROCEDURE PiuUno (VAR dato: integer);
BEGIN
   dato := dato + 1
END;


PROCEDURE MenoUno (VAR dato: integer);
BEGIN
   dato := dato - 1
END;


PROCEDURE PerDue (VAR dato: integer);
BEGIN
   dato := dato * 2
END;


PROCEDURE operazioni (l: tipolista; PROCEDURE oper (VAR x: integer));

{modifica i valori contenuti nella lista puntata da l, applicando la procedura oper}

BEGIN {operazioni}
   WHILE l <> NIL DO
      BEGIN
         oper(l^.info);
         l := l^.pros
      END
```

```
   END; {operazioni}


BEGIN
   writeln('LETTURA');
   lettura(lista);
   writeln('SCRITTURA');
   ScriviLista(lista);
   writeln('INCREMENTO');
   operazioni(lista, PiuUno);
   ScriviLista(lista);
   writeln('RADDOPPIO');
   operazioni(lista, PerDue);
   ScriviLista(lista);
   writeln('DECREMENTO');
   operazioni(lista, MenoUno);
   ScriviLista(lista)
END.
```

Analogamente, è possibile passare come parametro a un sottoprogramma anche una `FUNCTION`, indicandone l'intestazione nella lista dei parametri formali.

Ad esempio, si supponga di avere definito un tipo `persona` che rappresenti, mediante un record, i dati anagrafici di una persona, e un tipo `tabpersona` in cui siano memorizzati, mediante un array, i dati relativi a piú persone. Dovendo costruire un programma in cui i dati delle persone memorizzate possano essere visualizzati, a scelta dell'utente, in ordine alfabetico o in ordine di età, si può costruire una sola procedura di ordinamento, che riceva come parametro una `FUNCTION` utilizzata per effettuare i confronti:

```
PROCEDURE ordina (VAR t: tabpersona; FUNCTION minore (a, b: persona): boolean);
```

La funzione `minore` deve restituire `true`, quando il record passato come primo parametro è minore del record passato come secondo parametro, rispetto al criterio desiderato.

Per l'ordinamento alfabetico, si scriverà una funzione con la seguente intestazione:

```
FUNCTION alfabetico (x, y: persona): boolean;
```

che restituisca `true` quando il cognome di `x` è minore del cognome di `y`. Pertanto, volendo ordinare un array `tabella` alfabeticamente, si userà la chiamata:

```
ordina(tabella, alfabetico)
```

Per l'ordinamento rispetto all'età si scriverà una funzione con l'intestazione:

```
FUNCTION PiuGiovane (x, y: persona): boolean;
```

che restituisca `true` quando il record `x` contiene i dati anagrafici di una persona piú giovane rispetto a quella i cui dati sono rappresentati nel record `y`. In questo caso, per ordinare l'array si utilizzerà la chiamata:

```
ordina(tabella, PiuGiovane)
```

Riassumendo, mentre col passaggio per valore e per riferimento, è possibile scrivere procedure, parametriche rispetto ai *dati* utilizzati, con il passaggio di sottoprogrammi è possibile parametrizzare anche le *azioni* svolte dalle procedure.

23.2 Pile

Abbiamo già introdotto il concetto di *pila* o *stack* nella Lezione 15, per descrivere il comportamento dinamico di un programma Pascal.

Ricordiamo che una struttura a pila è caratterizzata dal fatto che tutte le operazioni vengono effettuate dallo stesso estremo, che chiamiamo *cima* della pila. Pertanto è possibile aggiungere un elemento solo in cima alla pila e cancellare l'elemento che si trova in cima alla pila. Dunque, il primo elemento che può essere eliminato da una data pila è quello che è stato inserito per ultimo, da cui il nome LIFO (*Last In First Out*) con cui vengono indicate le pile.

Esempio: valutazione di espressioni in notazione postfissa

Un'espressione aritmetica in notazione postfissa (v. Lezione 21), come ad esempio 1 2 $*$ 3 $+$ 5 4 $-$ $*$ può essere valutata semplicemente utilizzando una pila di interi.

L'algoritmo per il calcolo non fa altro che scandire l'espressione da sinistra verso destra e applicare le seguenti regole:

- quando si legge un numero, lo si inserisce in cima alla pila;
- quando si legge un simbolo di operazione, si prelevano i due numeri che si trovano piú in alto sulla pila, si applica ad essi l'operazione, si inserisce il risultato sulla pila.

Se l'espressione è formata correttamente, alla fine di questo procedimento la pila conterrrà, come unico elemento, il risultato.

Mostriamo i passi per il calcolo dell'espressione 8 1 $-$ 4 3 5 $*$ $+$ $*$.

- Inizialmente la pila è vuota:

pila	*parte di espressione che resta da leggere*
	8 1 $-$ 4 3 5 $*$ $+$ $*$

- Lettura del numero 8. Il numero viene inserito in cima alla pila:

pila	*parte di espressione che resta da leggere*
8	1 $-$ 4 3 5 $*$ $+$ $*$

- Lettura del numero 1. Il numero viene inserito in cima alla pila:

pila	*parte di espressione che resta da leggere*
1	$-$ 4 3 5 $*$ $+$ $*$
8	

- Lettura del segno $-$. Si prelevano i due valori piú in alto nella pila, cioè 1 e 8 (la pila dunque resta vuota); si applica l'operazione ai due numeri (il primo operando è l'ultimo numero prelevato, cioè 8). Si inserisce il risultato dell'operazione in cima alla pila:

pila	*parte di espressione che resta da leggere*
7	4 3 5 $*$ $+$ $*$

- Lettura del numero 4:

pila	*parte di espressione che resta da leggere*
4	3 5 $*$ $+$ $*$
7	

- Lettura del numero 3:

pila	*parte di espressione che resta da leggere*
3	5 * + *
4	
7	

- Lettura del numero 5:

pila	*parte di espressione che resta da leggere*
5	* + *
3	
4	
7	

- Lettura del segno $*$:

pila	*parte di espressione che resta da leggere*
15	+ *
4	
7	

- Lettura del segno $+$:

pila	*parte di espressione che resta da leggere*
19	*
7	

- Lettura del segno $*$:

pila	*parte di espressione che resta da leggere*
133	

Implementazione di pile in Pascal

Accenniamo brevemente a come può essere realizzata una pila in Pascal. Una prima implementazione è ottenibile memorizzando gli elementi che costituiscono la pila in un vettore ed utilizzando una variabile per memorizzare l'indice corrispondente alla cima. Il vettore e la variabile vengono organizzati in un record. Ad esempio un tipo per rappresentare pile di interi può essere definito come segue:

```
TYPE
   tipopila = RECORD
         info: ARRAY [1..maxpila] OF integer;
         cima: 0..maxpila
      END;
```

Nella definizione precedente, `maxpila` è una costante, che quindi va definita prima, che indica il numero massimo di elementi memorizzabili nella pila. Se `p` è una variabile dichiarata di tipo `tipopila`, e `p.cima` contiene 0, la variabile `p` rappresenta la pila vuota. Se invece `p.cima` contiene 3, la variabile `p` rappresenta la pila costituita, dal basso verso l'alto, dai valori delle variabili `p.info[1]`, `p.info[2]` e `p.info[3]`.

Per aggiungere un elemento alla pila è sufficiente incrementarne il campo `cima` ed effettuare l'inserimento nella posizione di indice `cima` dell'array `info`. La cancellazione avviene in modo simmetrico.

Lo svantaggio principale di questa semplice implementazione sta nel fatto che occorre definire a priori il numero massimo di elementi che possono essere contenuti nella pila.

Quando si debbano scrivere programmi privi di vincoli sulla lunghezza della pila è necessario far ricorso a variabili dinamiche. In questo caso la pila può essere realizzata mediante una lista, in cui la cima della pila corrisponde al primo elemento della lista. L'implementazione di pile mediante liste è illustrata nell'esempio seguente.

Esempio

Si vuole costruire un ambiente per lo studio e la prova di sottoprogrammi che manipolano sequenze di interi. In base ad un menú, l'utente può scegliere di volta in volta quali operazioni effettuare manipolando una sequenza, inizialmente vuota. Le operazioni previste sono:

1. Inserimento di un nuovo elemento *all'inizio* della sequenza.
2. Visualizzazione del contenuto della sequenza.
3. Calcolo e visualizzazione della media dei valori contenuti nella sequenza.

Il programma *deve essere privo* di vincoli sulla lunghezza massima della sequenza trattata e deve risultare facilmente espandibile nel caso si vogliano introdurre nuove funzionalità.

Un programma come quello richiesto dal testo del problema, in cui l'utente possa attivare successivamente operazioni scelte sulla base di un menú, che ogni volta si ripresenta, può essere costruito basandosi sul seguente schema:

```
BEGIN
   fase iniziale
   fase operativa
   fase finale
END
```

Nella `fase iniziale` vengono effettuate alcune operazioni preliminari (in genere inizializzazioni di variabili), mentre nella `fase finale` vengono effettuate, se necessarie, alcune operazioni conclusive (nel programma che svilupperemo la fase finale è assente).

Nella `fase operativa` viene presentato un menú, in cui vengono indicate le possibili operazioni selezionabili (in questo caso le tre operazioni previste dal testo del problema), a cui viene aggiunta la possibilità di terminare l'esecuzione del programma. Ogni scelta è contrassegnata da un carattere.

Dopo avere visualizzato il menú, il programma attende che l'utente digiti il carattere corrispondente alla scelta effettuata. Sulla base di tale carattere, il programma attiva un sottoprogramma che esegue l'operazione selezionata. Al termine dell'esecuzione di tale operazione, viene ripresentato all'utente il menú al fine di permettere la scelta di una nuova operazione. Tutto questo viene eseguito fino a quando l'utente seleziona la terminazione del programma.

La fase operativa può dunque essere codificata basandosi su un ciclo:

```
REPEAT
   visualizza il menu'
   leggi la scelta dell'utente
   esegui il sottoprogramma corrispondente alla scelta effettuata
UNTIL l'utente sceglie di terminare l'esecuzione
```

La selezione del sottoprogramma da attivare avviene mediante un costrutto `CASE`, dopo avere verificato che il carattere digitato dall'utente corrisponda ad una delle scelte previste.

Nel nostro specifico caso, possiamo indicare le tre operazioni specificate nel testo con i caratteri `'1'`, `'2'` e `'3'`. La possibilità di uscire dal programma potrebbe essere indicata con il carattere

`'4'`. Tuttavia, nel caso si voglia introdurre nel programma una quarta funzionalità, occorrerebbe modificare il carattere utilizzato per indicare la terminazione. Per evitare questo inconveniente, scegliamo di indicare con `'0'` la scelta corrispondente all'uscita dal programma. Pertanto, una scelta è valida se si trova nell'insieme `['0'..'3']`. Per rendere il programma piú facilmente espandibile, utilizziamo una costante di nome `sceltamax` per indicare la scelta possibile con valore piú alto (in questo caso `'3'`). Per controllare la validità di una scelta ci si riferirà dunque all'insieme `['0'..sceltamax]`.

```
REPEAT

   {visualizzazione menu}
   writeln('   Operazioni disponibili:');
   writeln('1. Inserimento di un nuovo elemento all''inizio della sequenza');
   writeln('2. Visualizzazione del contenuto della sequenza');
   writeln('3. Calcolo della media dei valori della sequenza');
   writeln;
   writeln('0. Fine programma');
   writeln;

   {lettura della scelta effettuata dall'utente}
   write('   Digitare la cifra corrispondente all''operazione desiderata ');
   readln(scelta);

   {verifica ed esecuzione dell'operazione corrispondente alla scelta}
   IF scelta IN ['0'..sceltamax] THEN
      CASE scelta OF
         '0':  nessuna operazione da eseguire
         '1':  chiamata del sottoprogramma di inserimento
         '2':  chiamata del sottoprogramma di visualizzazione
         '3':  chiamata del sottoprogramma per il calcolo della media
      END {case}
   ELSE
      writeln('Selezione non valida')

UNTIL scelta = '0'
```

Passiamo ora a definire la struttura dati che utilizzeremo per rappresentare la sequenza. Osserviamo che, poiché l'inserimento di un nuovo elemento deve avvenire *all'inizio* della sequenza, quando si visualizza il contenuto della sequenza il primo elemento che si incontra è quello che è stato inserito per ultimo. In altre parole, gli elementi vengono visualizzati in ordine *contrario* rispetto all'ordine con cui sono stati inseriti (struttura LIFO). Pertanto è naturale risolvere il problema utilizzando una pila, in cui l'elemento in cima corrisponde al primo elemento della sequenza rappresentata.

Poiché non ci devono essere vincoli sulla lunghezza massima della sequenza trattata, rappresentiamo la pila facendo uso di una lista, che verrà gestita effettuando gli inserimenti *all'inizio*.

Introduciamo dunque le seguenti definizioni di tipo:

```
TYPE
   punt = ^nodo;
   nodo = RECORD
          info: integer;
          pros: punt;
      END;

   tipopila = punt;
```

Possiamo ora definire le intestazioni dei sottoprogrammi corrispondenti alle tre funzionalità richieste:

- `PROCEDURE inserimento (VAR s: tipopila)`
  legge un intero da `input` un numero intero e lo inserisce nella pila `s`, fornita come parametro.
- `PROCEDURE visualizzazione (s: tipopila)`
  scrive in `output` il contenuto della pila `s`, fornita come parametro.
- `PROCEDURE calcolomedia (s: tipopila)`
  calcola e scrive in `output` la media dei valori contenuti nella pila `s`, fornita come primo parametro.

Siamo ora in grado di scrivere il codice programma principale, con le relative dichiarazioni di costanti, tipi e variabili:

```
PROGRAM sequenze (input, output);

   CONST
      sceltamax = '3';

   TYPE
      punt = ^nodo;
      nodo = RECORD
            info: integer;
            pros: punt;
         END;

      tipopila = punt;

   VAR
      scelta: char;
      sequenza: tipopila;


...sottoprogrammi...


BEGIN {sequenze}
   {inizializza la sequenza come vuota}
   sequenza := NIL;

   REPEAT

      {visualizzazione menu}
      writeln;
      writeln('   Operazioni disponibili:');
      writeln('1. Inserimento di un nuovo elemento all''inizio della sequenza ');
      writeln('2. Visualizzazione del contenuto della sequenza');
      writeln('3. Calcolo della media dei valori della sequenza');
      writeln;
      writeln('0. Fine programma');
      writeln;
```

```
      {lettura della scelta effettuata dall'utente}
      write('   Digitare la cifra corrispondente all''operazione desiderata ');
      readln(scelta);

      {verifica ed esecuzione dell'operazione corrispondente alla scelta}
      IF scelta IN ['0'..sceltamax] THEN
         CASE scelta OF
            '0':
               ;
            '1':
               inserimento(sequenza);
            '2':
               visualizzazione(sequenza);
            '3':
               calcolomedia(sequenza)
         END {case}
      ELSE
         writeln('Selezione non valida')

   UNTIL scelta = '0'

END. {sequenze}
```

Passiamo ora alla codifica delle procedure che costituiscono il programma. Rappresentando la pila con una lista, la cima della pila corrisponde al primo elemento della lista. Dunque, l'inserimento di un nuovo elemento avviene *all'inizio* della lista. Pertanto possiamo sviluppare la procedura `inserimento` basandoci sulla procedura `InserInizio` presentata nella Lezione 19.

Le procedure `visualizzazione` e `calcolomedia` operano una scansione dell'intero contenuto della lista. In particolare, la procedura `calcolomedia` deve scandire la lista calcolando man mano la somma e il numero degli elementi. Pertanto la procedura può utilizzare due variabili `somma` e `n` di tipo `integer` per memorizzare tali valori, aggiornandoli all'interno del ciclo di scansione:

```
somma := 0;
n := 0;

WHILE la lista non e' finita DO
   BEGIN
      aggiungi a somma il valore contenuto nel nodo corrente
      incrementa n
      spostati sul nodo successivo
   END
```

All'uscita dal ciclo, viene calcolata e stampata la media, salvo nel caso in cui `n` valga 0, in cui viene fornito il messaggio `La sequenza e' vuota`.

Il codice completo del programma è riportato qui di seguito:

```
PROGRAM sequenze (input, output);

   CONST
      sceltamax = '3';

   TYPE
      punt = ^nodo;
```

```
   nodo = RECORD
          info: integer;
          pros: punt;
      END;

   tipopila = punt;

VAR
   scelta: char;
   sequenza: tipopila;



PROCEDURE inserimento (VAR s: tipopila);

{legge un intero da input e lo inserisce nella pila s}

   VAR
      x: integer;
      p: punt;

BEGIN {inserimento}

   {lettura del valore da inserire}
   write('Scrivi il numero da inserire nella sequenza ');
   readln(x);

   {creazione di un nuovo nodo contenente il numero letto}
   new(p);
   p^.info := x;

   {inserimento del nuovo nodo nella pila}
   p^.pros := s;
   s := p

END; {inserimento}



PROCEDURE visualizzazione (s: tipopila);

{visualizza il contenuto della pila puntata da s}

BEGIN {visualizzazione}

   IF s = NIL THEN
      writeln('La sequenza e'' vuota')
   ELSE
      BEGIN
         writeln('La sequenza contiene i seguenti elementi');
         REPEAT
            writeln(s^.info);
```

```
            s := s^.pros
         UNTIL s = NIL
      END

   END; {visualizzazione}



   PROCEDURE calcolomedia (s: tipopila);

   {calcola e visualizza la media dei valori memorizzati nella pila s}

      VAR
         media: real;
         somma, n: integer;

   BEGIN {calcolomedia}

      somma := 0;
      n := 0;

      WHILE s <> NIL DO
         BEGIN
            somma := somma + s^.info;
            n := n + 1;
            s := s^.pros
         END;

      IF n = 0 THEN
         writeln('La sequenza e'' vuota')
      ELSE
         BEGIN
            media := somma / n;
            writeln('La media degli elementi della sequenza e'' ', media : 1 : 1)
         END

   END; {calcolomedia}


BEGIN {sequenze}
   {inizializza la sequenza come vuota}
   sequenza := NIL;

   REPEAT

      {visualizzazione menu}
      writeln;
      writeln('   Operazioni disponibili:');
      writeln('1. Inserimento di un nuovo elemento all''inizio della sequenza ');
      writeln('2. Visualizzazione del contenuto della sequenza');
      writeln('3. Calcolo della media dei valori della sequenza');
      writeln;
```

```
      writeln('0. Fine programma');
      writeln;

      {lettura della scelta effettuata dall'utente}
      write('   Digitare la cifra corrispondente all''operazione desiderata ');
      readln(scelta);

      {verifica ed esecuzione dell'operazione corrispondente alla scelta}
      IF scelta IN ['0'..sceltamax] THEN
         CASE scelta OF
            '0':
               ;
            '1':
               inserimento(sequenza);
            '2':
               visualizzazione(sequenza);
            '3':
               calcolomedia(sequenza)
         END {case}
      ELSE
         writeln('Selezione non valida')

   UNTIL scelta = '0'

END. {sequenze}
```

23.3 Code

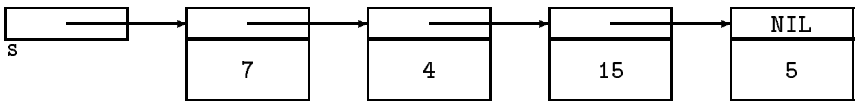
Consideriamo ora le strutture a *coda*. A differenza delle pile, in cui l'inserimento e l'accesso agli elementi avvengono sempre dalla stessa parte, nella coda gli elementi vengono prelevati da un lato e inseriti dall'altro. Si pensi, ad esempio, ad una coda di persone davanti allo sportello di un ufficio postale. In una coda, dunque, gli elementi vengono prelevati nello stesso ordine con cui vengono inseriti, da cui l'acronimo FIFO (*First In First Out*).

Anche le code possono essere realizzate sia facendo uso di vettori che facendo uso di liste. Presenteremo solo la realizzazione mediante liste.

Per mostrare l'implementazione di code mediante liste, modifichiamo il problema considerato nell'esempio precedente, richiedendo che gli inserimenti avvengano *alla fine* della sequenza. In questo caso, gli elementi verranno visualizzati nello *stesso ordine* secondo cui sono stati inseriti.

Una coda può essere realizzata con una lista, in cui gli inserimenti vengono effettuati *alla fine* della lista.

Si consideri ad esempio la seguente lista:

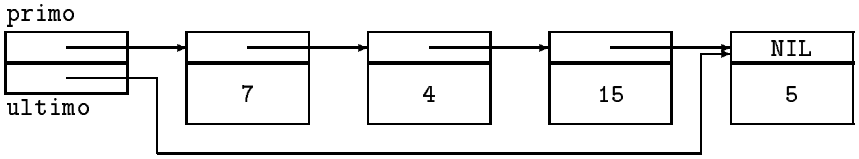


Per inserire un nuovo nodo *alla fine* della lista occorre:

- creare un nuovo nodo;
- attraversare l'intera lista fino a raggiungere l'ultimo nodo;

- collegare il nuovo nodo alla lista, facendo puntare ad esso il campo puntatore dell'ultimo nodo.

Questa tecnica è molto dispendiosa in termini di tempo. Infatti, ogni volta che si voglia inserire un nuovo elemento è necessario scandire l'intera lista. Per evitare di effettuare ogni volta questa scansione, si può semplicemente rappresentare la struttura con due puntatori, uno al primo, l'altro all'ultimo elemento:



La coda vuota viene invece rappresentata ponendo a `NIL` entrambi i puntatori `primo` e `ultimo`.

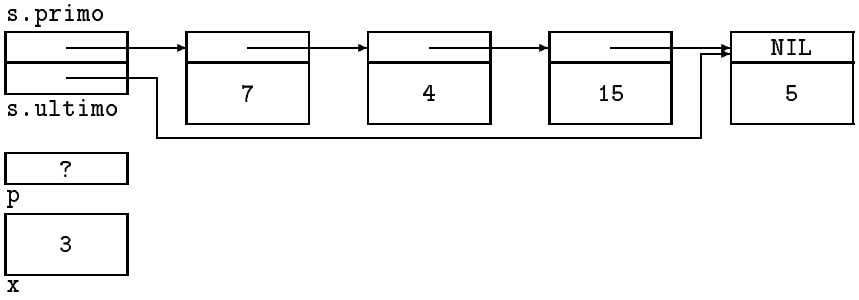
Per realizzare una coda possiamo organizzare i puntatori `primo` e `ultimo` in un record, e introdurre i seguenti tipi:

```
TYPE
   punt = ^nodo;
   nodo = RECORD
          info: integer;
          pros: punt;
      END;

   tipocoda = RECORD
          primo, ultimo: punt
      END;
```

Esaminiamo ora l'inserimento di un valore, contenuto in una variabile `x`, in una coda `s` di tipo `tipocoda`. Faremo come sempre uso di una variabile ausiliaria `p` di tipo `punt`.

Supponiamo che la situazione prima dell'inserimento sia quella rappresentata nella seguente figura:

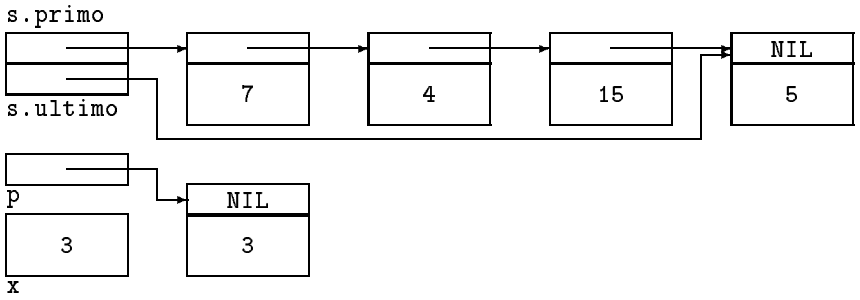


Per inserire un nuovo nodo contenente il valore di `x` nella coda, effettuiamo le seguenti operazioni:

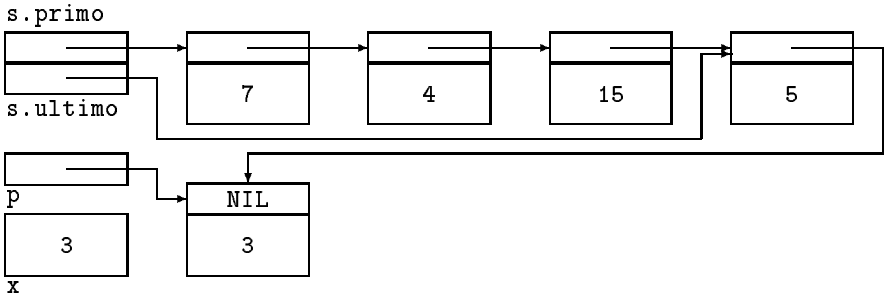
- Creazione del nuovo nodo e registrazione delle informazioni (compreso il valore `NIL` nel campo `pros`):

  ```
  new(p);
  p^.info := x;
  p^.pros := NIL
  ```

  Dopo l'esecuzione di queste operazioni, la memoria contiene:

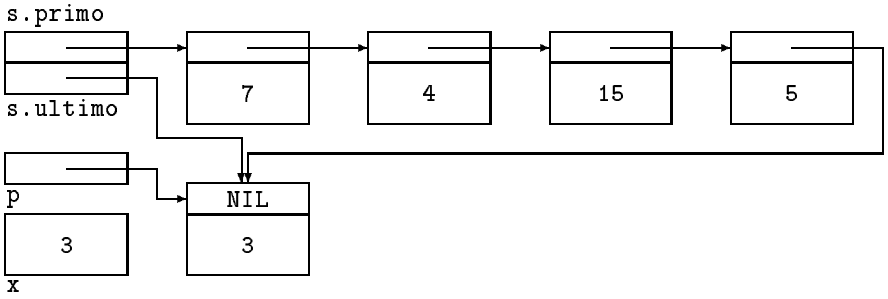


- Collegamento del nuovo nodo alla fine della lista (dopo il nodo puntato da `s.ultimo`, mediante l'assegnamento `s.ultimo^.pros := p`:



  A questo punto il nuovo nodo è stato inserito correttamente nella lista.

- Aggiornamento di `s.ultimo`, che deve puntare anch'esso al nuovo nodo, mediante l'assegnamento `s.ultimo := p`



Se la coda è vuota, entrambi i campi `primo` e `ultimo` contengono `NIL`. In questo caso l'inserimento avviene semplicemente facendo puntare sia `primo` che `ultimo` al nuovo nodo creato, cioè scrivendo gli assegnamenti

```
s.primo := p;
s.ultimo := p
```

Riportiamo di seguito una procedura per la lettura di un intero da `input` e per il suo inserimento alla fine di una coda passata alla procedura come parametro per riferimento:

```
PROCEDURE inserimento (VAR s: tipocoda);

{legge un intero da input e lo inserisce alla fine della coda s}
```

```
   VAR
      x: integer;
      p: punt;

BEGIN {inserimento}

   {lettura del valore da inserire}
   write('Scrivi il numero da inserire nella sequenza ');
   readln(x);

   {creazione di un nuovo nodo contenente il numero letto}
   new(p);
   p^.info := x;
   p^.pros := NIL;

   {inserimento del nuovo nodo in fondo alla coda}
   WITH s DO
      IF primo = NIL THEN {se la coda e' vuota...}
         BEGIN
            primo := p;
            ultimo := p
         END
      ELSE
         BEGIN
            ultimo^.pros := p;
            ultimo := p
         END

END; {inserimento}
```

Il problema considerato all'inizio può essere facilmente risolto basandosi sullo stesso schema del programma costruito nel caso delle pile. In particolare, si utilizza la procedura di inserimento riportata qui sopra. Per le procedure di visualizzazione della sequenza e di calcolo della media sono sufficienti semplici modifiche.

Il listato completo del programma è il seguente:

```
PROGRAM sequenze (input, output);

   CONST
      sceltamax = '3';

   TYPE
      punt = ^nodo;
      nodo = RECORD
            info: integer;
            pros: punt;
         END;

      tipocoda = RECORD
            primo, ultimo: punt
         END;

   VAR
```

```
   scelta: char;
   sequenza: tipocoda;



PROCEDURE inserimento (VAR s: tipocoda);

{legge un intero da input e lo inserisce alla fine della coda s}

   VAR
      x: integer;
      p: punt;

BEGIN {inserimento}

   {lettura del valore da inserire}
   write('Scrivi il numero da inserire nella sequenza ');
   readln(x);

   {creazione di un nuovo nodo contenente il numero letto}
   new(p);
   p^.info := x;
   p^.pros := NIL;

   {inserimento del nuovo nodo in fondo alla coda}
   WITH s DO
      IF primo = NIL THEN {se la coda e' vuota...}
         BEGIN
            primo := p;
            ultimo := p
         END
      ELSE
         BEGIN
            ultimo^.pros := p;
            ultimo := p
         END

END; {inserimento}



PROCEDURE visualizzazione (s: tipocoda);

{visualizza il contenuto della coda puntata da s}

   VAR
      p: punt;

BEGIN {visualizzazione}

   p := s.primo;
```

```
      IF p = NIL THEN
         writeln('La sequenza e'' vuota')
      ELSE
         BEGIN
            writeln('La sequenza contiene i seguenti elementi');
            REPEAT
               writeln(p^.info);
               p := p^.pros
            UNTIL p = NIL
         END

   END; {visualizzazione}



   PROCEDURE calcolomedia (s: tipocoda);

   {calcola e visualizza la media dei valori memorizzati nella coda s}

      VAR
         p: punt;
         media: real;
         somma, n: integer;

   BEGIN {calcolomedia}

      p := s.primo;
      somma := 0;
      n := 0;

      WHILE p <> NIL DO
         BEGIN
            somma := somma + p^.info;
            n := n + 1;
            p := p^.pros
         END;

      IF n = 0 THEN
         writeln('La sequenza e'' vuota')
      ELSE
         BEGIN
            media := somma / n;
            writeln('La media degli elementi della sequenza e'' ', media : 1 : 1)
         END

   END; {calcolomedia}


BEGIN {sequenze}
   {inizializza la sequenza come vuota}
   sequenza.primo := NIL;
   sequenza.ultimo := NIL;
```

```
  REPEAT

     {visualizzazione menu}
     writeln;
     writeln('   Operazioni disponibili:');
     writeln('1. Inserimento di un nuovo elemento alla fine della sequenza');
     writeln('2. Visualizzazione del contenuto della sequenza');
     writeln('3. Calcolo della media dei valori della sequenza');
     writeln;
     writeln('0. Fine programma');
     writeln;

     {lettura della scelta effettuata dall'utente}
     write('   Digitare la cifra corrispondente all''operazione desiderata ');
     readln(scelta);

     {verifica ed esecuzione dell'operazione corrispondente alla scelta}
     IF scelta IN ['0'..sceltamax] THEN
        CASE scelta OF
           '0':
              ;
           '1':
              inserimento(sequenza);
           '2':
              visualizzazione(sequenza);
           '3':
              calcolomedia(sequenza)
        END {case}
     ELSE
        writeln('Selezione non valida')

  UNTIL scelta = '0'

END. {sequenze}
```

Esercizi

1. Modificare il programma `liste`, inserendo un menú che permetta all'utente la selezione di una delle operazioni previste. Aggiungere anche altre operazioni, come il cambio di segno o l'azzeramento di tutti i valori contenuti nella lista.

2. Scrivere un programma che legga (a scelta dell'utente da `input` o da un file), un elenco di record contenti i dati anagrafici di alcune persone, e scriva (a scelta dell'utente in `output` o su di un file) l'elenco ordinato, secondo un criterio selezionato dall'utente, tra i seguenti:

   - in ordine alfabetico;
   - in ordine di età a partire dalla persona piú giovane;
   - in ordine di età a partire dalla persona piú anziana.

3. Scrivere un programma che calcoli il valore di un'espressione aritmetica scritta in notazione

postfissa, utilizzando una pila. Costruire il programma in due versioni: nella prima versione la pila viene implementata con un array, nella seconda con una lista.

4. Costruire (in versione ricorsiva e iterativa) una procedura che ricevendo come unico parametro il puntatore a una coda di interi, elimini da tale coda tutti gli elementi che si trovano in posizione pari.

5. Costruire (in versione ricorsiva e iterativa) una procedura che ricevendo come unico parametro il puntatore a una coda di interi, elimini da tale coda tutti gli elementi che si trovano in posizione dispari.

6. (Dal tema d'esame del 4 aprile 1997.) Si vuole costruire un ambiente per lo studio e la prova di sottoprogrammi che manipolano sequenze di nomi. In base ad un menù, l'utente può scegliere di volta in volta quali operazioni effettuare manipolando una sequenza, inizialmente vuota. Le operazioni previste sono:

   (a) Inserimento di un nuovo nome *alla fine* della sequenza.

   (b) Stampa del contenuto della sequenza.

   (c) Scorrimento circolare all'indietro della sequenza: tutti gli elementi vengono spostati all'indietro di una posizione, eccetto il primo che passa in ultima posizione. Ad esempio, la sequenza contenente `pippo pluto topolino paperino paperone` dovrà essere trasformata nella sequenza `pluto topolino paperino paperone pippo`.

   (d) Rimozione di un elemento dalla sequenza: dato in ingresso un nome, il programma lo elimina dalla sequenza, se presente.

   Il programma *deve essere privo* di vincoli sulla lunghezza massima della sequenza trattata e deve risultare facilmente espandibile nel caso si vogliano introdurre nuove funzionalità. È possibile fissare un limite superiore alla lunghezza dei nomi trattati (ad esempio 15 caratteri).

7. Scrivere un programma che legga un file di testo `t` presente su memoria di massa e crei un file `f` contenente, in ordine alfabetico, tutte le parole presenti in `t` con l'indicazione del numero di occorrenze. Ci si può basare sul programma costruito per risolvere l'esercizio 17 della Lezione 21. Le componenti di `t` saranno `RECORD` con due campi: la parola e il suo numero di occorrenze.

8. Scrivere un programma che legga un file di testo `t` presente su memoria di massa e crei un file `f` contenente, in ordine alfabetico, un indice analitico di `f`, cioè un elenco di tutte le parole presenti in `t` con l'indicazione dei numeri delle linee in cui appaiono. Ci si può basare sul programma costruito per risolvere l'esercizio 18 della Lezione 21. In questo caso nel file `t` occorre memorizzare la prima parola, seguita dai numeri delle linee in cui appare, la seconda parola, seguita dai numeri delle linee in cui appare, e così via. Si noti che il numero di occorrenze di ciascuna parola non può essere limitato a priori. Il problema può essere risolto utilizzando, come componenti di `t`, `RECORD` con varianti, in cui in alternativa si può memorizzare una parola o la posizione di una parola.