Lezione 6
14–15 ottobre 1999

Argomenti trattati
- Tipi semplici: il tipo `char`, il tipo `boolean`, tipi subrange.
- Funzioni sui tipi scalari.
- Il tipo `real`.
- Le procedure standard `write` e `writeln`.
- La selezione: il costrutto `CASE`.

6.1 Il tipo `char`

Il tipo `char` è un tipo scalare predefinito del Pascal, le cui costanti sono tutti i caratteri del set utilizzato dalla macchina (molto spesso l'insieme dei caratteri ASCII, *American Standard Code for Information Interchange*). Una variabile di tipo `char` può dunque assumere come valore un singolo carattere. Spesso i valori di tipo `char` vengono rappresentati utilizzando 8 bit. In questo modo è possibile rappresentare 256 caratteri differenti.

Riportiamo sotto una parte della tabella ASCII che associa un codice numerico a ciascun carattere. All'interno della tabella sono riportati i caratteri, il codice corrispondente a ciascun carattere si ottiene sommando i numeri indicati all'inizio della riga e della colonna che contengono il carattere (ad esempio, il codice di `H` è 64+8=72).

Le costanti di tipo `char` vengono sempre indicate tra apici, come `'a'`, `';'`, `'1'` (da non confondere con la costante `1` del tipo `integer`). I valori di tipo `char` sono ordinati, in base al set di caratteri utilizzato. Pertanto è possibile effettuare confronti, utilizzando gli usuali operatori `<`, `<=`, `>`, `>=`, `=` e `<>`.

Se, ad esempio, abbiamo dichiarato

```
VAR x, y: char;
```

dopo gli assegnamenti

```
x := 'Z'
y := 'F'
```

la condizione `x < y` risulta falsa, in quanto il valore contenuto in `x` non precede, nella tabella ASCII, quello contenuto in `y`.

Per scrivere programmi che utilizzino il tipo `char`, non è necessario conoscere la tabella ASCII. Al contrario, è bene che i programmi siano indipendenti dalla codifica ASCII, facilitando così la loro trasportabilità su macchine che utilizzano differenti codifiche. Una caratteristica importante da ricordare, comune alla maggior parte delle codifiche utilizzate, è che le lettere minuscole hanno codifiche consecutive (nel caso ASCII da 97 a 122), così le maiuscole e le cifre. Questo, come vedremo, permette di effettuare alcune conversioni, *senza* la necessità di scrivere esplicitamente i codici.

	32	48	64	80	96	112
0		0	@	P	`	p
1	!	1	A	Q	a	q
2	”	2	B	R	b	r
3	#	2	C	S	c	s
4	$	4	D	T	d	t
5	%	5	E	U	e	u
6	&	6	F	V	f	v
7	´	7	G	W	g	w
8	(	8	H	X	h	x
9	)	9	I	Y	i	y
10	*	:	J	Z	j	z
11	+	;	K	[	k	{
12	,	<	L	\	l	|
13	-	=	M	]	m	}
14	.	>	N	^	n	~
15	/	?	O	_	o	del

6.2 Il tipo `boolean`

Il tipo `boolean` è un tipo scalare predefinito del Pascal, contenente solo le due costanti `false` e `true`, utilizzate per rappresentare, rispettivamente, i due valori di verità *falso* e *vero*. Ad una variabile di tipo `boolean` può dunque essere assegnato il valore di verità di una condizione o, in altre parole, il risultato di un'espressione booleana. Ad esempio, il programma che determina il minimo tra due numeri, presentato nella lezione precedente, può essere riscritto come segue, introducendo una variabile di tipo `boolean` per memorizzare il risultato del confronto:

```
PROGRAM min (input, output);

{scrive in output il valore del minimo tra due interi letti da input}

VAR
   x, y, m: integer;
   xminimo: boolean;

BEGIN
   {leggi i numeri}
   readln(x, y);

   {determina il minimo}
   xminimo := x < y;
   IF xminimo THEN m := x
             ELSE m := y;
```

```
   {scrivi il risultato}
   writeln('Il minimo e'' :',m)
END.
```

L'assegnamento `xminimo := x < y` viene valutato nel solito modo: prima si calcola il valore dell'espressione che si trova a destra del simbolo `:=` di assegnamento. In questo caso l'espressione è la condizione `x < y`. Se tale condizione è vera, alla variabile `xminimo` verrà assegnato valore `true`, altrimenti verrà assegnato valore `false`.

Il tipo `boolean` dispone di due operatori binari indicati con `AND` e `OR` (corrispondenti, rispettivamente, alle operazioni di *congiunzione* e *disgiunzione*) e di un operatore unario indicato con `NOT` (*negazione*), che applicati a valori di tipo `boolean`, restituiscono un risultato di tipo `boolean`.

In particolare, il risultato della congiunzione è *vero* solo quando *entrambi* gli operandi hanno valore *vero*; il risultato della disgiunzione è *vero* quando *almeno uno* degli operandi ha valore *vero*. La negazione scambia *vero* con *falso*. In altre parole, il comportamento dei tre operatori è rappresentato dalle seguenti *tabelle di verità*.

x	y	x AND y
false	false	false
false	true	false
true	false	false
true	true	true

x	y	x OR y
false	false	false
false	true	true
true	false	true
true	true	true

x	NOT x
false	true
true	false

Utilizzando gli operatori `AND`, `OR` e `NOT` è possibile costruire espressioni booleane complesse. L'operatore `NOT` ha la massima priorità, cioè, in assenza di parentesi, viene applicato per primo, `OR` la minima. Pertanto, se `a, b, c` sono di tipo `boolean`, l'espressione `NOT a AND b OR a AND c` è equivalente a `((NOT a) AND b) OR (a AND c)`.

Costruiamo ora la tabella di verità dell'espressione `NOT(a AND b) OR (a AND c)`, a partire dalle tabelle dei tre operatori. A sinistra della tabella elenchiamo i valori che possono assumere le variabili `a, b, c`. A destra, sotto ogni operatore, riportiamo i valori di verità corrispondenti ai valori elencati a sinistra: calcoliamo prima di tutto i valori dell'espressione `a AND b`, riportandoli sotto il primo `AND`; su di essi applichiamo l'operatore `NOT`, scrivendo nella colonna sottostante i risultati; calcoliamo poi il valore di `a AND c`, riportando il risultato sotto il secondo `AND`; infine, applichiamo l'operatore `OR` ai valori scritti nelle colonne sotto il `NOT` e sotto il secondo `AND`, ottenendo così il valore dell'intera espressione, riportato nella colonna sotto l'operatore `OR`:

a	b	c	NOT	(a AND b)	OR	(a AND c)
false	false	false	true	false	true	false
false	false	true	true	false	true	false
false	true	false	true	false	true	false
false	true	true	true	false	true	false
true	false	false	true	false	true	false
true	false	true	true	false	true	true
true	true	false	false	true	false	false
true	true	true	false	true	true	true

Dati due valori booleani x e y, valgono le seguenti uguaglianze, dette *Leggi di De Morgan*:

- `NOT(`x `AND` y`) = NOT` x `OR NOT` y
- `NOT(`x `OR` y`) = NOT` x `AND NOT` y

Dimostrare per esercizio le Leggi di De Morgan, costruendo le tabelle di verità dei lati sinistri e destri delle due uguaglianze.

Le due costanti `false` e `true` del tipo `boolean` sono tra loro ordinate. In particolare, `false` è minore di `true`.

Esempio

Si consideri una variabile `x` di tipo `integer`. Vogliamo esprimere la condizione "il valore assoluto di `x` è minore di 4" o, in altre parole, il valore di `x` è compreso nell'intervallo tra -4 e +4, estremi esclusi. Dunque il valore di `x` deve verificare *entrambe* le condizioni: essere maggiore di -4 *ed* essere minore di +4. La condizione è pertanto

```
(x > -4) AND (x < 4)
```

Consideriamo ora la condizione "il valore assoluto di `x` *non* è minore di 4". Ciò significa che il valore di `x` non deve trovarsi all'interno dell'intervallo indicato in precedenza, cioè deve essere minore o uguale a -4 *o* maggiore o uguale a 4. La condizione è dunque

```
(x <= -4) OR (x >= 4)
```

Si osservi che le due condizioni sono una la negazione dell'altra. È possibile passare dalla negazione della prima condizione alla seconda (e dalla negazione della seconda alla prima) mediante le leggi di De Morgan. Infatti, la negazione della prima condizione è:

```
NOT((x > -4) AND (x < 4))
```

da cui si ottiene

```
NOT(x > -4) OR NOT(x < 4)
```

I due `NOT` possono essere eliminati negando le condizioni considerate (si ricordi che `NOT(x > y)` equivale a `x <= y`, mentre `NOT(x >= y)` equivale a `x < y`). In questo modo si ottiene

```
(x <= -4) OR (x >= 4)
```

6.3 Subrange

È possibile definire un *subrange* di qualunque tipo scalare già definito, indicando gli estremi dell'intervallo considerato. Pertanto, con le definizioni

```
TYPE
   weekend = sab..dom;
   cifra = '0'..'9';
   indice = 0..9;
   vero = true..true;
```

si definiscono subrange dei tipi `giorno`, `char`, `integer` e `boolean`, rispettivamente. Sulle variabili di tipo subrange è possibile effettuare tutte le operazioni consentite sul tipo di partenza. L'unico vincolo è la restrizione sull'insieme di valori che possono essere assunti.

6.4 Funzioni sui tipi scalari

I tipi enumerativi, i tipi predefiniti `integer`, `char`, `boolean`, insieme ai tipi subrange, formano la famiglia dei *tipi scalari*. Questi tipi possiedono tre funzioni comuni, `pred`, `succ` e `ord`.

- La funzione `pred` applicata a un valore di un tipo scalare ne restituisce il *predecessore* (non è definita per il primo elemento del tipo).
- La funzione `succ` applicata a un valore di un tipo scalare ne restituisce il *successore* (non è definita per l'ultimo elemento del tipo).

- La funzione `ord` applicata a un valore di un tipo scalare restituisce un `integer` che rappresenta la posizione del valore all'interno del tipo.

Ad esempio, `pred(1)` è `0`, `pred('1')` è `'0'`, `pred('c')` è `'b'`. Se `x` è una variabile di tipo `integer`, `pred(x)` è `x-1` e `succ(x)` è `x+1`. Se abbiamo definito

```
TYPE
   giorno = (lun, mar, mer, gio, ven, sab, dom);
```

allora `pred(gio)` è `mer`. Per i tipi enumerativi, la funzione `ord` restituisce la posizione del valore all'interno del tipo, contando a partire da zero. Dunque `ord(lun)` è `0`, `ord(mar)` è `1`, e così via.

Per il tipo `integer`, la funzione `ord` restituisce il valore stesso a cui è stata applicata.

Per il tipo `boolean`, `ord(false)` vale zero, mentre `ord(true)` vale uno. Dunque `succ(false)` è `true`, mentre `pred(true)` è `false`.

Per il tipo `char`, la funzione `ord` restituisce la codifica del carattere nel set di caratteri utilizzati. Ad esempio, per i caratteri ASCII, si ha che `ord('A')=65`, `ord('a')=97`, `ord('0')=48`.

È inoltre disponibile la funzione `chr` che riceve come argomento un valore di tipo `integer` e restituisce come risultato il carattere corrispondente. Ad esempio, nel caso dei caratteri ASCII, `chr(65)` è uguale ad `'A'`.

Segnaliamo inoltre la funzione `odd`, che applicata a un valore di tipo `integer` restituisce `true` se il valore è dispari, `false` altrimenti. Altre funzioni relative al tipo `integer` sono indicate piú avanti.

6.5 Esempi

Allo scopo di illustrare l'uso delle variabili di tipo `boolean`, presentiamo un programma che legge un carattere e indica se esso sia una lettera, una cifra, oppure né lettera né cifra.

Il programma è organizzato come segue:

- inizialmente viene letto da `input` il carattere da esaminare;
- si valutano dunque tre espressioni booleane, che verificano, rispettivamente, che il carattere cada nell'intervallo delle lettere minuscole, delle lettere maiuscole e delle cifre; i risultati di questi controlli vengono memorizzati in variabili di tipo `boolean`;
- in base ai valori delle tre variabili booleane si scrive in `output` il messaggio opportuno.

Ecco il testo del programma:

```
PROGRAM lettere (input,output);

VAR
   c: char;
   minuscola, maiuscola, cifra: boolean;

BEGIN {lettere}

   {lettura del carattere}
   readln(c);

   {pone la variabile minuscola a true se e solo se il carattere e' una
    lettera minuscola}
   minuscola := (c >= 'a') AND (c <= 'z');
```

```
   {pone la variabile maiuscola a true se e solo se il carattere e' una
    lettera maiuscola}
   maiuscola := (c >= 'A') AND (c <= 'Z');

   {pone la variabile cifra a true se e solo se il carattere e' una cifra}
   cifra := (c >= '0') AND (c <= '9');

   {seleziona il messaggio da fornire in output, in base ai valori
    delle variabili minuscola, maiuscola, cifra}
   IF maiuscola OR minuscola THEN writeln(c,' e'' una lettera')
   ELSE IF cifra THEN writeln(c,' e'' una cifra')
   ELSE writeln(c,' non e'' ne'' una lettera ne'' una cifra')

END. {lettere}
```

Si noti che sono stati effettuati direttamente confronti tra valori di tipo `char`. Ad esempio, nell'espressione

```
(c >= 'a') AND (c <= 'z')
```

si controlla che il carattere memorizzato nella variabile `c` sia successivo o uguale ad `'a'` e precedente o uguale a `'z'`. Poiché, come abbiamo detto, nella maggior parte dei codici utilizzati, le codifiche delle lettere minuscole sono consecutive, questa espressione sarà vera se e solo se `c` contiene una lettera minuscola. Si sarebbe anche potuto utilizzare l'espressione

```
(ord(c) >= ord('a')) AND (ord(c) <= ord('z'))
```

che però introduce quattro conversioni da `char` a `integer`, che sono del tutto inutili, e fastidiose da leggere.

Le funzioni `chr` e `ord` sono utili per effettuare conversioni tra caratteri e tra caratteri e numeri. Abbiamo già osservato che nella tabella ASCII le lettere minuscole si trovano in un blocco contiguo (dal codice 97 al 122), le maiuscole in un altro blocco (dal 65 al 90), le cifre in un altro ancora (dal 48 al 57). Se ad esempio la variabile `c`, di tipo `char`, contiene una lettera minuscola, l'espressione

```
chr(ord(c) - ord('a') + ord('A'))
```

ha come risultato la lettera maiuscola corrispondente. Infatti, `ord(c)` fornisce il codice della carattere memorizzato in `c`. Se questo è una lettera minuscola, `ord(c) - ord('a')` è uguale alla distanza di questa lettera dalla `'a'` (ad esempio la `'d'` ha distanza 3). Aggiungendo tale distanza al codice della `'A'` maiuscola, si ottiene il codice della lettera maiuscola corrispondente. Infine, applicando la funzione `chr`, si trasforma il codice nel carattere. Dunque, se `c` contiene una lettera minuscola, l'assegnamento

```
c := chr(ord(c) - ord('a') + ord('A'))
```

sostituisce il contenuto di `c` con la maiuscola corrispondente. Lasciando indicati `ord('a')` e `ord('A')` nell'espressione, la conversione è corretta anche nel caso in cui i caratteri vengano rappresentati con un codice diverso da quello ASCII. Se invece si sostituiscono ad `ord('a')` e `ord('A')`, scrivendo `chr(ord(c) - 97 + 65)` o, addirittura, `chr(ord(c) - 32)`, il programma risulterà corretto solo per macchine che utilizzano il codice ASCII, e quindi dovrà essere modificato se trasportato su macchine differenti.

Con un ragionamento analogo, si può osservare che se una variabile `c` di tipo `char` contiene una cifra, cioè uno dei caratteri da `'0'` a `'9'`, l'espressione

```
ord(c) - ord('0')
```

ha come risultato il valore `integer` corrispondente a tale cifra.

6.6 Il tipo `real`

Oltre ai tipi `integer`, `char` e `boolean`, il linguaggio Pascal mette a disposizione il tipo predefinito `real`, per la rappresentazione dei numeri reali. Sono disponibili i seguenti operatori binari tra valori di tipo `real`, che forniscono un risultato di tipo `real`: `+, -, *, /`. Essi denotano rispettivamente le operazioni di somma, sottrazione, moltiplicazione, divisione reale. Inoltre sono disponibili gli usuali operatori unari `+` e `-`, per indicare numeri positivi o negativi. Le costanti di tipo `real` vengono indicate con sequenze di cifre, eventualmente precedute dal segno e seguite dalla parte decimale, come `-48.24`, `765.12`. Al fine di poter rappresentare in poco spazio numeri molto grandi o molto piccoli, è possibile inoltre utilizzare la *notazione scientifica*, detta anche in *virgola mobile*, evidenziando la *mantissa* e l'*esponente*, preceduto dal carattere `E`. Ad esempio, la costante `0.000012`, cioè $1.2 \cdot 10^{-5}$, può essere rappresentata piú brevemente come `1.2E-5`.

Poiché la rappresentazione in un computer deve necessariamente utilizzare un numero finito di bit, mentre vi sono numeri reali la cui rappresentazione richiederebbe un numero infinito di bit, nell'aritmetica dei reali vi è perdita di precisione. In calcoli successivi e ripetuti, piccole perdite di precisione possono essere via via amplificate fino ad ottenere risultati notevolmente distanti da quelli corretti. Lo studio delle problematiche riguardanti il trattamento automatico di numeri reali verrà svolto nei corsi di *Analisi Numerica* e *Calcolo Numerico*.

Nelle espressioni che coinvolgono valori di tipo `real`, è sempre possibile utilizzare valori `integer`, che verranno automaticamente convertiti al tipo `real`. In ogni caso, *il risultato sarà di tipo* `real`, e *non* potrà essere assegnato a variabili di tipo `integer`. Ad esempio, nell'assegnamento:

```
x := y / z
```

è presente l'operatore di divisione reale. Pertanto, il suo risultato sarà di tipo `real`. Dunque, la variabile `x` dovrà essere anch'essa di tipo `real` (anche nel caso in cui il risultato della divisione sia un numero senza la virgola). D'altra parte, le variabili `y` e `z` possono essere sia di tipo `real` che di tipo `integer`. Nel caso una di esse (o entrambe) sia di tipo `integer`, il suo valore verrà convertito nella rappresentazione `real` per il calcolo dell'espressione.

Consideriamo ora l'assegnamento

```
x := y + z
```

in cui viene utilizzato l'operatore `+`, disponibile sia per il tipo `integer` che per il tipo `real` (come si vedrà nel corso di *Architettura degli elaboratori*, gli algoritmi utilizzati dal processore per sommare due valori di tipo `real` e due valori di tipo `integer` sono profondamente differenti). Possiamo avere le seguenti situazioni:

- La variabile `x` è di tipo `integer`: in questo caso le può essere assegnato solo un valore di tipo `integer`. Pertanto `y + z` deve essere un'espressione che ha come risultato un `integer`, il che implica che entrambe le variabili `y` e `z` siano di tipo `integer`.
- La variabile `x` è di tipo `real`: in questo caso è possibile assegnare a `x` valori sia di tipo `real` che di tipo `integer` (che verranno automaticamente convertiti nella rappresentazione `real` al momento dell'assegnamento). Sono pertanto possibili le seguenti situazioni:
  - Entrambe le variabili `y` e `z` sono di tipo `real`: in questo caso l'operatore `+` denota un'operazione tra `real`, con risultato `real`, e pertanto non viene effettuata alcuna conversione di rappresentazione.
  - Entrambe le variabili `y` e `z` sono di tipo `integer`: in questo caso l'operatore `+` denota un'operazione tra `integer`, con risultato `integer`. Dopo avere eseguito l'operazione, il risultato viene convertito nel tipo `real` per effettuare l'assegnamento.

– Una variabile è di tipo `integer`, l'altra di tipo `real`: in questo caso, prima di effettuare la somma, il valore di tipo `integer` viene convertito nella corrispondente rappresentazione `real`, quindi viene effettuata una somma tra `real`.

In pratica, mentre è possibile, mediante conversioni implicite, utilizzare valori di tipo `integer`, dove siano richiesti valori di tipo `real`, non è possibile fare il viceversa. Esistono tuttavia due funzioni che permettono di convertire esplicitamente valori di tipo `real` in valori di tipo `integer`. La funzione `trunc` effettua la conversione eliminando la parte decimale del numero; la funzione `round` effettua invece l'arrotondamento all'intero piú vicino. Piú precisamente:

$$\texttt{round(r)} = \begin{cases} \texttt{trunc(r + 0.5)} & \text{se } \texttt{r} \geq 0 \\ \texttt{trunc(r - 0.5)} & \text{se } \texttt{r} < 0 \end{cases}$$

Osserviamo infine che il tipo `real` *non* è un tipo scalare. Pertanto non è possibile utilizzare con esso le funzioni `pred`, `succ` e `ord`, non è possibile definire subrange e, come vedremo, non è possibile utilizzare variabili di tipo `real` come variabili di controllo in cicli `FOR`. È invece possibile confrontare valori di tipo `real` mediante gli usuali operatori di confronto `<`, `<=`, ecc.

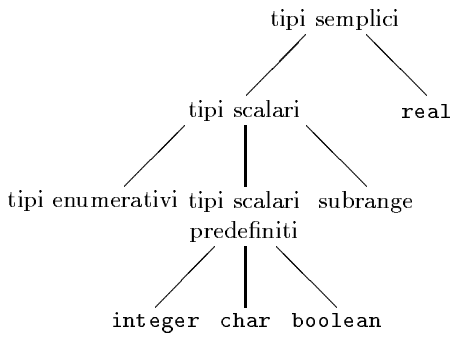
Le seguenti funzioni predefinite, ricevono un parametro di tipo `real` e restituiscono un risultato di tipo `real`:

- `sin(x)`: seno di `x`, dove `x` è espresso in radianti;
- `cos(x)`: coseno di `x`, dove `x` è espresso in radianti;
- `arctan(x)`: arcotangente di `x`, in radianti;
- `ln(x)`: logaritmo naturale di `x`;
- `exp(x)`: esponenziale di `x`;
- `sqrt(x)`: radice quadrata di `x`.

Le seguenti funzioni predefinite, con un parametro di tipo `real` restituiscono un risultato di tipo `real`, on un parametro di tipo `integer` restituiscono un risultato di tipo `integer`:

- `abs(x)`: valore assoluto di `x`;
- `sqr(x)`: `x` elevato al quadrato.

La seguente figura riassume i tipi semplici disponibili in Pascal:



6.7 Le procedure predefinite `write` e `writeln`

Negli esempi, abbiamo utilizzato le procedure predefinite **`write`** e **`writeln`** per scrivere in output. Entrambe le procedure riportano in output i valori dei propri parametri. La procedura **`write`**, dopo avere scritto in output, lascia il cursore sulla stessa riga, pertanto una **`write`** o una **`writeln`** successiva inizieranno a scrivere da quel punto. La procedura **`writeln`** fa invece posizionare il cursore all'inizio della riga successiva.

Le procedure **`write`** e **`writeln`** possono ricevere come parametri valori dei tipi **`char`**, **`integer`**, **`real`**, **`boolean`**, o stringhe di caratteri. Nella stessa **`write`** o **`writeln`** è possibile scrivere in output differenti valori, separandoli con virgole, come ad esempio nella chiamata:

```
writeln('Il carattere ',c,' e'' una lettera')
```

dove la variabile **`c`** è di tipo **`char`**, e **`'Il carattere '`** e **`' e'' una lettera'`** sono stringhe di caratteri (si ricorda che i valori dei caratteri e delle stringhe vengono rappresentati tra apici; per rappresentare il carattere apice si fa uso di due apici).

È possibile specificare il numero di caratteri utilizzati per scrivere ciascun parametro. Questo risulta utile quando si voglia, ad esempio, incolonnare dei risultati. In particolare:

- La scrittura di un valore di tipo **`char`** occupa esattamente 1 carattere. Pertanto, se ad esempio la variabile **`c`** contiene il carattere **`'a'`**, l'istruzione

  ```
  writeln('Il valore di c e'' uguale a',c)
  ```

  produrrà in uscita:

  ```
  Il valore di c e' uguale aa
  ```

  È possibile utilizzare un numero di caratteri maggiore, specificando, dopo il nome della variabile, un'*ampiezza di campo*. In questo caso, il valore della variabile verrà preceduto da un numero opportuno di spazi bianchi. Ad esempio, l'istruzione

  ```
  writeln('Il valore di c e'' uguale a',c:3)
  ```

  produrrà in uscita:

  ```
  Il valore di c e' uguale a  a
  ```

  dove il valore della variabile **`c`** è preceduto da due spazi bianchi. Analoghe convenzioni valgono per le stringhe.

- I valori di tipo **`integer`** vengono normalmente stampati utilizzando un numero fissato di caratteri, che dipende dall'implementazione (in molte implementazioni è 12). Il numero viene allineato a destra, preceduto da spazi bianchi. Ad esempio, se la variabile **`x`** di tipo **`integer`** contiene il valore **`12345`**, l'istruzione

  ```
  writeln('Il valore di x e'' uguale a',x)
  ```

  produrrà in uscita il messaggio:

  ```
  Il valore di x e' uguale a       12345
  ```

  in cui il numero è preceduto da 7 spazi bianchi. Anche in questo caso è possibile specificare l'ampiezza di campo. Ad esempio, l'istruzione

```
writeln('Il valore di x e'' uguale a',x:7)
```

produrrà in uscita il messaggio:

```
Il valore di x e' uguale a  12345
```

in cui il numero è preceduto da due spazi bianchi. Nel caso in cui l'ampiezza di campo sia minore del numero di caratteri necessari per stampare l'intero, essa viene automaticamente aumentata, allineando il numero a sinistra. Ad esempio, per produrre numeri allineati a sinistra, anziché a destra, si può utilizzare ampiezza di campo uguale a 1. Pertanto l'istruzione

```
writeln('Il valore di x e'' uguale a',x:1)
```

produrrà in uscita il messaggio:

```
Il valore di x e' uguale a12345
```

in cui il valore di `x` non è preceduto da alcuno spazio (naturalmente, in questo caso è opportuno inserire uno spazio alla fine della stringa data come primo parametro).

- I valori di tipo `real` vengono normalmente stampati in notazione scientifica, con un numero di cifre per mantissa ed esponente che dipende dall'implementazione (in molte implementazioni con 6 cifre decimali nella mantissa e 2 per l'esponente).

  È possibile specificare una o due ampiezze di campo. Con un'unica ampiezza di campo, il numero viene scritto sempre in notazione scientifica, utilizzando il numero di caratteri specificato nell'ampiezza di campo. Con due ampiezze di campo il numero viene rappresentato in notazione decimale, utilizzando il numero totale di caratteri specificato dalla prima ampiezza, con un numero di cifre decimali specificato nella seconda ampiezza. Ad esempio, l'esecuzione del seguente programma (compilato utilizzando HP-Pascal)

```
PROGRAM stampareali (output);

VAR
   x: real;

BEGIN
   x := 12345.6789;
   writeln('Il valore di x e'' uguale a', x);
   writeln('Il valore di x e'' uguale a', x:15);
   writeln('Il valore di x e'' uguale a', x:15:3)
END.
```

ha prodotto in output:

```
Il valore di x e' uguale a 1.23457E+04
Il valore di x e' uguale a  1.2345679E+04
Il valore di x e' uguale a      12345.679
```

(si notino gli arrotondamenti).

6.8 La selezione: il costrutto CASE

L'istruzione **CASE** permette di selezionare l'esecuzione di un'istruzione, tra piú istruzioni possibili, in base al valore di un'espressione detta *selettore*.

Il seguente programma riceve in ingresso due numeri interi e verifica se essi possono rappresentare una coppia (giorno,mese). In caso affermativo il programma calcola il numero dei giorni che mancano alla fine del mese. Si notino i costrutti **IF** innestati per calcolare il numero dei giorni nel mese, in base al mese considerato (per semplicità non si è tenuto conto degli anni bisestili).

```
PROGRAM date (input, output);

VAR
   giorno, mese, giorninelmese, giornimancanti: integer;

BEGIN {date}
   {lettura dati}
   write('Inserire il numero del giorno e il numero del mese ');
   readln(giorno, mese);

   {controllo data}
   IF (mese < 1) OR (mese > 12) THEN
      writeln('Mese non valido')
   ELSE
      BEGIN
         {calcolo del numero di giorni del mese}
         IF (mese = 11) OR (mese = 4) OR (mese = 6) OR (mese = 9) THEN
            giorninelmese := 30
         ELSE IF mese = 2 THEN
            giorninelmese := 28
         ELSE
            giorninelmese := 31;

         {controllo sul giorno}
         IF (giorno < 1) OR (giorno > giorninelmese) THEN
            writeln('Giorno non valido')
         ELSE {calcolo dei giorni mancanti}
            BEGIN
               giornimancanti := giorninelmese - giorno;
               writeln('Mancano ', giornimancanti : 1, ' giorni alla fine del mese')
            END
      END {else}
END. {date}
```

Ecco il programma riscritto utilizzando un costrutto **CASE** per il calcolo del numero dei giorni:

```
PROGRAM date (input, output);

VAR
   giorno, mese, giorninelmese, giornimancanti: integer;

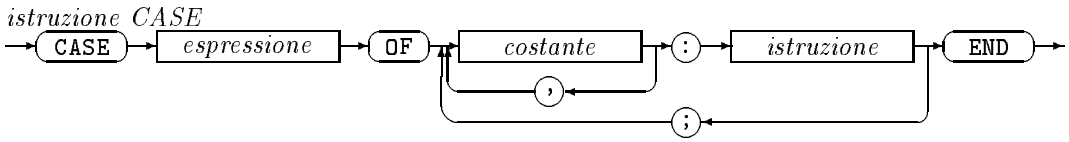
BEGIN {date}
   {lettura dati}
    write('Inserire il numero del giorno e il numero del mese ');
```

```
    readln(giorno, mese);

   {controllo data}
    IF (mese < 1) OR (mese > 12) THEN
        writeln('Mese non valido')
    ELSE
        BEGIN
         {calcolo del numero di giorni del mese}
           CASE mese OF
               4, 6, 9, 11:
                   giorninelmese := 30;
               2:
                   giorninelmese := 28;
               1, 3, 5, 7, 8, 10, 12:
                   giorninelmese := 31
           END; {case}

         {controllo sul giorno}
           IF (giorno < 1) OR (giorno > giorninelmese) THEN
               writeln('Giorno non valido')
           ELSE {calcolo dei giorni mancanti}
               BEGIN
                   giornimancanti := giorninelmese - giorno;
                   writeln('Mancano ', giornimancanti : 1, ' giorni alla fine del mese')
               END
        END {else}
END. {date}
```

La sintassi dell'istruzione `CASE` è data dalla seguente carta:



Al momento dell'esecuzione, viene valutata l'espressione utilizzata come *selettore*, cioè l'espressione scritta tra le due parole `CASE` e `OF`, che *deve* essere di un tipo scalare. Viene quindi eseguita l'istruzione preceduta dalla costante uguale al risultato dell'espressione selettore. L'esecuzione prosegue poi da ciò che segue la parola `END` che chiude il costrutto `CASE`. Si ricordi che:

- se il valore assunto dal selettore non è indicato tra le costanti elencate prima delle varie istruzioni, si avrà un errore in esecuzione (alcune implementazioni non standard del Pascal prevedono una clausula `ELSE` o `OTHERWISE` per i valori non elencati);
- la stessa costante non può essere elencata piú di una volta;
- come sempre, è possibile raggruppare piú istruzioni tra le parole riservate `BEGIN` ed `END`;
- in molte implementazioni del Pascal è possibile elencare anche intervalli. Ad esempio, se la variabile `ch` è di tipo `char`, è possibile scrivere:

```
CASE ch OF
   'a'..'z','A'..'Z': writeln('il carattere e'' una lettera');
   '0'..'9': writeln('il carattere e'' una cifre')
   ...
END
```

Esercizi

1. Costruire un programma che legga da input un carattere. Se il carattere è una lettera maiuscola, il programma deve scrivere in uscita la lettera minuscola corrispondente; se il carattere è una lettera minuscola, il programma deve scrivere la lettera maiuscola corrispondente; infine, se il carattere non è una lettera, il programma deve riportarlo in uscita senza modifiche.

2. Sia `c` una variabile di tipo `char`. Esprimere, senza utilizzare l'operatore `NOT` la condizione "`c` contiene una lettera minuscola" e la condizione "`c` non contiene una lettera minuscola".

3. Costruire le tabelle di verità associate alle espressioni

   - `(a AND NOT b) OR NOT a`
   - `NOT a OR b AND (a OR c)`
   - `a AND NOT(c OR b)`

   dove `a`, `b` e `c` sono di tipo `boolean`.

4. Scrivere una condizione equivalente a `NOT((a > 10) AND (a <= 20))` che non contenga l'operatore `NOT`.

5. Scrivere una condizione equivalente a `NOT((a = 5) OR NOT(b < 10))` che non contenga l'operatore `NOT`.

6. Siano `x, y, z` variabili di tipo `boolean`, e `a` una variabile di tipo `char`. Individuare, tra le seguenti condizioni quelle che sono sempre vere e quelle che sono sempre false. Per ognuna delle rimanenti condizioni, indicare dei valori delle variabili che vi compaiono che rendano la condizione vera e dei valori che rendano la condizione falsa:

   - `(x = x) OR (x = NOT x)`
   - `(x = x) AND (x = NOT x)`
   - `(x = y) OR (x = NOT y)`
   - `(x = y) AND (x = NOT y)`
   - `z OR NOT z`
   - `z AND NOT z`
   - `x OR (y AND NOT x)`
   - `x OR (y OR NOT x)`
   - `x AND (y AND NOT x)`
   - `x AND (y OR NOT x)`
   - `(a >= 'A') AND (a <= 'Z')`
   - `(a >= 'A') OR (a <= 'Z')`
   - `(a < 'A') AND (a > 'Z')`
   - `(a < 'A') OR (a > 'Z')`
   - `x OR (a <> 'a')`
   - `x AND (a <> 'a')`
   - `x OR (a <> a)`
   - `x AND (a <> a)`
   - `x OR (a >= 'A') AND (a <= 'Z')`
   - `(x OR (a >= 'A')) AND (a <= 'Z')`

- `x OR (a >= 'A') OR (a <= 'Z')`
- `x AND (a >= 'A') OR (a <= 'Z')`
- `x AND ((a >= 'A') OR (a <= 'Z'))`

7. Per ognuna delle seguenti espressioni, indicare i tipi che possono assumere le variabili, e, in funzione di essi, il tipo del risultato; si evidenzino inoltre eventuali conversioni implicite di tipo:

   - `x + y - z`
   - `x + y / z`
   - `x + y DIV z`
   - `x + y MOD z`
   - `x > y`
   - `x > succ(y)`
   - `x > y + 1`
   - `x = ord (y)`
   - `x = chr(y)`
   - `succ(chr(y))`
   - `x = (a > b)`
   - `x = ord(a AND b)`
   - `x = ord(x)`
   - `x = NOT x`

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

   - ```
     c:= a / b;
     d:= a MOD b;
     e:= c + d;
     ```
   - ```
     c:= a DIV b + a MOD b;
     d:= a / b + a MOD b;
     e:= c + d;
     ```
   - ```
     a := b + 1;
     c := a DIV b;
     d := a / b;
     ```
   - ```
     a := x = y;
     z := x - y;
     b := NOT a;
     ```
   - ```
     a := chr(z);
     b := a <> 'a';
     ```
   - ```
     z := y > ord(w);
     w := chr(ord('a')+ord(z));
     ```
   - ```
     IF p THEN q := a/b
          ELSE c := a DIV b;
     ```

9. Scrivere e fare eseguire dei semplici programmi di prova, per verificare il comportamento delle procedure `write` e `writeln` utilizzando differenti profondità di campo.
     ```