3.8 Salti condizionati e incondizionati

L'istruzione `break` consente di interrompere l'esecuzione del `case`, provocando un salto del flusso di esecuzione alla prima istruzione successiva. Una seconda possibilità di uso di `break` è quella di forzare la terminazione di un'iterazione `for`, `while` o `do-while`, provocando un salto alla prima istruzione successiva al ciclo.

Riprendiamo il programma per il calcolo della somma e la ricerca del massimo dei numeri immessi dall'utente. L'istruzione di controllo del ciclo ne bloccava l'esecuzione se veniva immesso in ingresso il valore zero o quando fossero già stati inseriti dieci valori. Scorporiamo quest'ultima verifica affidandola a un'altra istruzione:

```
for(i=1; numero!=0; i++) {
  printf("Inser. intero positivo: \t");
  scanf("%d", &numero);
  if(numero>max)  max=numero;
  somma+=numero;
  if(i==10) break;
}
```

Naturalmente era migliore la soluzione precedente:

```
for(i=1; numero!=0 && i<=10; i++)
```

L'istruzione `break` provoca l'uscita solo dal ciclo più interno. Per esempio la sequenza di istruzioni

```
for(j=1; j<=3; j++) {
  somma = 0;
  max = 0;
  numero = 1;
  for(i=1; numero!=0; i++) {
    printf("Inser. intero positivo: \t");
    scanf("%d", &numero);
    if(numero>max)  max=numero;
    somma+=numero;
    if(i==10) break;
  }
  printf("Somma: %d\n", somma);
  printf("Maggiore: %d\n", max);
}
```

ripete per tre volte il calcolo della somma e la ricerca del massimo visti precedentemente.

L'istruzione `continue` lavora in modo simile al `break`, con la differenza che anziché forzare l'uscita dal ciclo provoca una nuova iterazione a partire dall'inizio, saltando le istruzioni che rimangono dal punto in cui si trova `continue` alla fine del ciclo, come nel seguente esempio.

```
for(i=1; numero!=0 && i<=10; i++) {
       printf("Inser. intero positivo: \t");
       scanf("%d", &numero);
```

```
    if(numero<0) continue;
    if(numero>max)  max=numero;
    somma+=numero;
  }
```

Se il numero immesso dall'utente è negativo vengono saltate le ultime due istruzioni del ciclo.

Per analogia presentiamo la funzione `exit`, che fa parte della libreria standard `stdlib.h` e che provoca l'immediata terminazione del programma e il ritorno al sistema operativo. Normalmente la funzione viene chiamata non passandole nessun argomento, il che significa *terminazione normale*. Altri argomenti consentono di indicare che si è verificato un particolare tipo di errore e il destinatario di questa comunicazione dovrebbe essere un processo di livello superiore in grado di gestire la condizione anomala. Nell'esempio precedente, in caso di immissione da parte dell'utente del valore zero, il programma sarebbe terminato:

```
if(numero<0) exit();
```

Generalmente `exit` viene inserita dopo la verifica negativa di una condizione indispensabile per il proseguimento dell'esecuzione. Per ipotesi, un programma relativo a un gioco potrebbe richiedere la presenza di una scheda grafica nel sistema per essere eseguito:

```
if(!scheda_grafica()) exit;
gioco();
```

`scheda_grafica` è una funzione definita dall'utente che ha valore vero se almeno una delle schede grafiche richieste è presente e falso in caso contrario. In quest'ultimo caso viene eseguito `exit` e il programma ha termine ▬.

L'istruzione `goto` richiede un'*etichetta* – un identificatore C valido – seguito da un carattere due punti. Tale identificatore deve essere presente nell'istruzione `goto`:

```
i=1;
ciclo:
  i++;
  printf("Inser. intero positivo: \t");
  scanf("%d", &numero);
  if(numero>max)  max=numero;
  somma+=numero;
if(numero!=0 && i<=10) goto ciclo;
```

Le ragioni della programmazione strutturata, tra cui pulizia ed eleganza del codice, sconsigliano l'uso generalizzato di `break`, `continue` ed `exit`, e "proibiscono" quello del `goto`.