CORSO DI PROGRAMMAZIONE
A.A. 2014-15

Dispensa 6

Laboratorio

Dott. Mirko Ravaioli
e-mail: mirko.ravaioli@unibo.it

6.2 Operatori relazionali

Gli Gli operatori relazionali del C vengono utilizzati per confrontare espressioni e rispondere a comande come "x è più grande di 100" o "y è più piccola di x"

Un'espressione contenente un operatore relazionale viene valutata secondo un valore di verità cioè veto (valore 1 o false) o falso (valore 0 o false).

Operatore	***Simbolo***	***Domanda sottointesa***	***Esempio***
Uguale	==	L'operando x è uguale all'operando y?	X ==y
Maggiore di	>	L'operando x è maggiore dell'operando y?	X > y
Minore di	<	L'operando x è minore dell'operando y?	X < y
Maggiore o uguale a	>=	L'operando x è maggiore o uguale dell'operando y?	X >= y
Minore o uguale a	<=	L'operando x è minore o uguale dell'operando y?	X<= y
Diverso	!=	L'operando x è diverso dall'operando y?	X != y

6.3 Istruzione if

L'istruzione if valuta un'espressione e dirige conseguentemente il flusso di esecuzione del programma. La sintassi (ridotta) dell'istruzione if è la seguente:

```
if(espressione)
{
      istruzioni
}
```

Se il valore di verità dell'espressione è vero viene conseguita l'istruzione presente nel corpo dell'istruzione (tra parentesi graffe). In caso contrario il flusso di esecuzione salta all'istruzione che segue quella contenuta nell'if. Si potrebbe dire che l'esecuzione dell'istruzione dipende dal risultato dell'espressione.

```
#include <stdio.h>

int x, y;

int main()
{
      /*Input dei valori da tastiera*/

      printf("\n Inserire un valore per x: ");
      scanf("%d",&x);
      printf("\n Inserire un valore per y: ");
      scanf("%d",&y);

      /*Confronta i valori e stampa i risultati*/

      if (x == y)
      {
            printf("x è uguale a y");
      }

      if (x > y)
```

```
        {
            printf("x è maggiore di y");
        }

        if (x < y)
        {
            printf("x è minore di y");
        }

        return 0;
    }
```

La clausola else

L'istruzione if supporta la clausola else, che ha la seguente sintassi:

```
if (espressione)
{
    istruzione1;
}
else
{
    istruzione2;
}
```

Dove la clausole *else* è facoltativa. L'*espressione* è valutata; se è vera (se cioè *espressione* ha un valore diverso da zero), è eseguita *istruzione1*. Se è falsa (*espressione* è zero) e se c'è una clausola *else*, viene invece eseguita *istruzione2*.

Stante la natura facoltativa della clausola *else* in un costrutto *if-else*, si crea ambiguità quando un else è omesso da una sequenza di *if* annidiata. Per convenzione, il compilatore abbina else al più vicino *if* precedente che ne sia privo. Per esempio:

```
if (n > 0)
{
    if (a > b)
    {
        z = a;
    }
    else
    {
        z = b;
    }
}
```

la clausola *else* va con l'istruzione *if* più interna, come segnala la rientranza. Se si vuole un risultato diverso, è necessario ricorrere alle parentesi graffe per imporre il giusto annidiamento:

```
if (n > 0)
{
    if (a > b)
    {
        z = a;
    }
}
else
{
    z = b;
}
```

Vediamo alcuni esempi, schematizziamo con le lettere maiuscole A, B, C, D… dei blocchi di istruzioni di codice:

Esempio 1:

```
A
if (exp)
{
      B
}
C
```

- Se exp restituisce un valore vero (o un numero diverso da zero) la sequenza delle istruzioni è A, B, C
- Se exp restituisce un valore falso (o un numero numero uguale a zero) la sequenza delle istruzioni è A, C. Questo perché solo se la condizione è vera si accede al corpo dell'istruzione *if*.

Esempio 2:

```
A
if (exp)
{
      B
}
else
{
      C
}
D
```

- Se exp restituisce un valore vero (o un numero diverso da zero) la sequenza delle istruzioni è A, B, D.
- Se exp restituisce un valore falso (o un numero numero uguale a zero) la sequenza delle istruzioni è A, C, D.

Esempio 3:

```
A
if (exp1)
{
      B
      if (exp2)
      {
            C
      }
      else
      {
            D
      }
      E
}
else
{
      F
      if (exp3)
      {
            G
      }
      H
}
I
```

EXP1	EXP2	EXP3	SEQUENZA ISTRUZIONI
V	V	V	A B C E I
V	V	F	A B C E I
V	F	F	A B D E I
F	F	F	A F H I
V	F	V	A B D E I
F	F	V	A F G H I
F	V	F	A F G H I
F	V	V	A F G H I

Consideriamo il seguente costrutto:

```
if (espresione)
      istruzione
else if (espresione)
      istruzione
else if (espresione)
      istruzione
else if (espresione)
      istruzione
else
      istruzione
```

Questa sequenza di istruzioni *if* è il modo più generale di analizzare un ventaglio di possibilità in C. Le espressioni sono sempre valutate nell'ordine, se una di esse è vera, l'istruzione con cui è associata viene eseguita, e ciò conclude l'esecuzione dell'intero costrutto. Come sempre il codice per ogni istruzione può essere un'istruzione singola o un blocco (più istruzioni tra graffe).

L'ultima clausola *else* si occupa del caso "nessuno dei precedenti" in cui, non essendo soddisfatta nessuna delle condizioni, occorre procedere d'ufficio. Può succedere che nessuna azione esplicita sia prevista per questo caso e allora il passo:

```
else
      istruzione
```

Può essere tralasciato, oppure lo si può riservare alla gestione degli errori, per rilevare una condizione "impossibile".

Vediamo un esempio: presi in ingresso 3 numeri stamparli in ordine crescente:

```
#include <stdio.h>

int main()
{
      int num1, num2, num3;
      int a, b, b;

      printf("\n Inserire primo valore: ");
      scanf("%d",&a);
      printf("\n Inserire secondo valore: ");
      scanf("%d",&b);
      printf("\n Inserire terzo valore: ");
      scanf("%d",&c);
```

```
        if (a < b)
        {
              if (a < c)
              {
                    num1 = a;
                    if (c < b)
                    {
                          num2 = c;
                          num3 = b;
                    }
                    else
                    {
                          num2 = b;
                          num3 = c;
                    }
              }
              else
              {
                    num1 = c;
                    num2 = a;
                    num3 = b;
              }
        }
        else
        {
              if (b < c)
              {
                    num1 = b;
                    if (c < a)
                    {
                          num2 = c;
                          num3 = a;
                    }
                    else
                    {
                          num2 = a;
                          num3 = c;
                    }
              }
              else
              {
                    num1 = c;
                    num2 = b;
                    num3 = a;
              }
        }

        printf("numeri in ordine crescente: %d, %d, %d",num1,num2,num3);

        return 0;
  }
```

6.4 Operatore condizionale

L'operatore condizionale è l'unico operatore ternario disponibile in C. Questo significa che è l'unico a lavorare su tre operandi senapati. La sua sintassi è la seguente:

```
exp1 ? exp2 : exp3;
```

Se exp1 è vera (cioè non nulla), l'intera espressione assume il valore di exp2. Se exp1 è falsa (ovvero il suo valore è zero), l'intera espressione assume il valore di exp3. Ad esempio l'istruzione seguente assegna a x il valore 1 se y è vera, mentre se y è falsa assegna ad x il valore 100:

```
x = y ? 1 : 100;
```

Analogamente, per assegnare a x il valore più grande tra quelli contenuti in x e in y:

```
z = (x > y) ? x : y ;
```

L'operatore condizionale funziona in maniera analoga a l'istruzione if. L'istruzione precedente si sarebbe potuta scrivere anche così:

```
if (x > y)
      z = x;
else
      z = y;
```

L'istruzione condizionale non può essere utilizzata al posto di qualsiasi istruzione if…else, ma quando è possibile il codice risultante è più conciso. L'operatore condizionale può essere utilizzato in alcune situazioni in cui l'istruzione if non è utilizzabile, come all'interno di un 'istruzione printf():

```
printf("il valore superiore è %d: " ((x > y) ? x : y));
```

6.5 Istruzione switch

L'istruzione *switch* esamina un ventaglio di possibilità. E devia di conseguenza il flusso dell'esecuzione. Ogni condizione esaminata esprime la coincidenza del valore di un'espressione con una costante intera.

```
switch (espressione)
{
      case espressione-costante:
            istruzioni;
            break;
      case espressione-costante:
            istruzioni;
            break;
      case espressione-costante:
            istruzioni;
            break;
      default:
            istruzioni
}
```

Ogni caso (clausola *case*) è etichettato da una o più costanti (o espressioni costanti) intere. Se uno di tali casi coincide con il valore dell'espressione, l'esecuzione prosegue dal caso in questione. Tutte le espressioni delle clausole *case* devono essere diverse. La clausola *default* è eseguita se nessuna delle altre è soddisfatta, ed è facoltativa; in sua assenza, se gli altri casi non sono soddisfatti non accade nulla: l'istruzione *switch* non ha alcun effetto. I casi e la clausola *default* possono trovarsi in un qualunque ordine.

L'istruzione *break* provoca l'uscita immediata dal costrutto *switch*. Poiché i casi fungono solo da etichette, quando l'esecuzione del codice relativo a un caso è finita, essa passa al successivo, a meno che non si decida esplicitamente di fare altrimenti. Questa gestione del flusso è detta a cascata. Le istruzioni break e

return sono le più comuni vie d'uscita da *switch*. Un istruzione *break* impone anche l'uscita immediata dai cicli *while*, *for* e *do*, come si vedrà nel seguito delle lezioni.

L'esecuzione a cascata presenta vantaggi e svantaggi. Il lato positivo è la possibilità di associare molti casi una singola azione, ma implica anche che ogni caso termini con un break per evirare il passaggio in cascata al successivo: tale meccanismo, infatti, non è robusto ed espone il brano *switch* a una possibile disintegrazione in caso di modifica del programma. Fatto salvo l'uso di più etichette in una singola computazione, l'esecuzione a cascata va usata con parsimonia e corredata con appositi commenti.