Dal Pascal al C

Parte 3^a

La programmazione strutturata

Le istruzioni di un algoritmo, che rappresenta il procedimento risolutivo di un problema, sono organizzate secondo schemi classificabili in tre costrutti fondamentali, chiamati **strutture di controllo**.

- Istruzioni organizzate in **sequenza**
- Istruzioni che vengono eseguite in **alternativa** con altre
- Istruzioni che vengono eseguite in **ripetizione**

La sequenza in C

La **sequenza** si rappresenta costruendo un blocco di istruzioni (il **blocco C**), ognuna terminante con il punto e virgola, delimitato all'inizio ed alla fine da una coppia di parentesi graffe aperta e chiusa.

All'interno del **blocco C** troviamo:

- (1) un insieme eventualmente vuoto di dichiarazioni di tipi e variabili, che sono visibili solo all'interno del blocco;
- (2) una sequenza eventualmente vuota di istruzioni.

Per quel che riguarda la nidificazione di blocchi e la visibilità degli identificatori in essi definiti, valgono le stesse regole che regolano la visibilità tra gli identificatori definiti in Pascal in procedure nidificate.

Se le dichiarazioni di variabili contengono una inizializzazione, tale inizializzazione viene eseguita ogni volta che il blocco viene eseguito.

Se una variabile definita all'interno del blocco è **static** essa conserva il suo valore anche quando il blocco cessa di essere eseguito; per cui se il blocco viene eseguito di nuovo, il valore iniziale della variabile è quello che la variabile aveva all'uscita della precedente esecuzione. L'inizializzazione di una variabile **static**, se presente viene eseguita una sola volta.

La struttura di alternativa (1)

Per la **selezione** nel linguaggio C sono disponibili due istruzioni condizionali, corrispondenti alla istruzione **if** e alla istruzione **case** del Pascal.

La sintassi dell'istruzione **if** del C è la seguente:

```
if (condizione)
    { istruzioni-a; }
else
    { istruzioni-b; }
```

Si noti che la condizione è racchiusa tra parentesi tonde e la mancanza della parola **then** presente nel linguaggio Pascal.

Se la **condizione** è **vera** (diversa da **0**), vengono eseguite le **istruzioni-a**, altrimenti vengono eseguite le **istruzioni-b**.

Nel caso in cui dopo la **condizione** o dopo l'**else** ci sia un'unica istruzione, le parentesi graffe possono essere omesse:

```
if (condizione) istruzione-a;
else istruzione-b;
```

Se si ha una selezione ad una sola via, si omette il ramo **else**:

```
if (condizione) istruzione-a;
```

L'istruzione **if** nel Pascal e nel C

```
if (a=c) and (b=c)
  then write('equilatero')
  else if (a=b) or (b=c)
    then write('isoscele')
    else write('scaleno');
```

```
if (a==c && b==c)
  printf("equilatero");
else if (a==b || b==c)
  printf("isoscele");
else printf("scaleno");
```

Nell'esempio precedente si vede come, in una struttura di alternativa, dopo la **condizione** o dopo l'**else**, può essere presente un gruppo di istruzioni, contenente a sua volta altre strutture **if** interne alla struttura più esterna. In questo caso si parla di **strutture nidificate (if nidificato)**.

La struttura di alternativa (2)

Nel linguaggio C la **struttura di scelta multipla** è realizzata dall'istruzione **switch**:

```
switch (variabile) {  
  case valore-1:  
    istruzioni-1;  
    break;  
  case valore-2:  
    istruzioni-2;  
    break;  
  . . . . .  
  case valore-n:  
    istruzioni-n;  
    break;  
  default:  
    istruzioni;  
    break;  
}
```

Dopo la parola **switch**, è indicato il nome della **variabile** (**selettore**) di cui si deve controllare il valore per decidere quale strada seguire tra quelle possibili. Accanto ai valori previsti, per ogni **case**, devono essere scritte l'istruzione o il blocco di istruzioni da eseguire nel caso che la **variabile** assuma quei valori. Se nessuno dei valori elencati corrisponde al valore di **variabile**, vengono eseguite le istruzioni scritte dopo **default**. Di solito, uno solo dei casi tra quelli previsti deve essere eseguito: è possibile interrompere il controllo degli altri casi inserendo l'istruzione **break** che provoca l'uscita dalla struttura **switch**.

La struttura di **selezione multipla** è una **struttura derivata** dalla struttura fondamentale binaria, in quanto potrebbe essere rappresentata utilizzando una successione di controlli nidificati del tipo:

if(condizione) ... else ...

La scelta multipla nel Pascal e nel C

```
case mese of
  2: giorni:=28;
  1,3,5,7,8,10,12: giorni:=31;
  4,6,9,11: giorni:=30
end;
```

```
switch (mese) {
  case 2: giorni=28;
           break;
  case 1:
  case 3:
  case 5:
  case 7:
  case 8:
  case 10:
  case 12: giorni=31;
           break;
  case 4:
  case 6:
  case 9:
  case 11: giorni=30;
}
```

Istruzioni ripetitive

Nel C sono disponibili due istruzioni ripetitive: l'istruzione **while** e l'istruzione **for**. Il comportamento dell'istruzione **repeat** del Pascal può essere ottenuta attraverso una forma particolare dell'istruzione **while**.

La sintassi dell'istruzione **while** nel C è la seguente:

```
while (condizione) {  
    istruzioni;  
}
```

Il comportamento dell'istruzione **while** del C è identico a quello dell'istruzione **while** del Pascal. Si osservi che il risultato della **condizione** può essere di tipo intero, carattere, numero con separatore decimale o puntatore. Le istruzioni all'interno del ciclo vengono eseguite se il risultato di **condizione** è diverso da **0**.

L'istruzione **while** nel Pascal e nel C

```
somma:=0;  
i:=1;  
while i<= 10 do  
begin  
    somma:=somma+i;  
    i:=i+1  
end;
```

```
somma=0;  
i=1;  
while (i<= 10)  
{  
    somma+=i;  
    i++;  
}
```

L'istruzione **do ... while** del C

Il comportamento dell'istruzione **do ... while** del C è analogo a quello dell'istruzione **repeat ... until** del Pascal. La sua sintassi è:

```
do {  
    istruzioni;  
} while (condizione);
```

Le istruzioni tra **do** e **while** vengono ripetute tante volte, mentre la **condizione** scritta dopo **while** si mantiene **vera**; in altre parole la ripetizione termina quando la **condizione** scritta dopo **while** diventa **falsa**. Questo comportamento, dal punto di vista logico, è esattamente opposto a quello della istruzione **repeat ... until** del Pascal.

Il **repeat...until** del Pascal e il **while...do** del C

```
repeat  
  read(ch)  
until ch = ' ';
```

```
do  
  ch=getchar();  
while (ch!=' ');
```

Lo stesso comportamento si può ottenere attraverso un programma C più compatto utilizzando gli operatori di assegnazione all'interno della **condizione** di una istruzione **while** propria:

```
while ((ch=getchar())!=' ');
```

da notare l'uso delle parentesi a causa della priorità maggiore dell'operatore **!=** rispetto all'operatore di assegnazione.

L'istruzione **for**

L'istruzione **for** del C permette di specificare cicli dal controllo molto più complesso della istruzione **for** del Pascal. La sintassi è la seguente:

```
for (i=inizio; i<=fine; i++) {  
    istruzioni;  
}
```

In realtà l'istruzione **for** del C corrisponde ad una istruzione **while** in cui sono messe in evidenza le fasi di inizializzazione delle variabili di controllo e la fase di modifica del loro valore all'interno del ciclo. La semantica è equivalente a quella della seguente istruzione **while**:

```
i=inizio;  
while (i<=fine) {  
    istruzioni;  
    i++;  
}
```

L'istruzione **for** nel Pascal e nel C

```
for a:=10 downto 1 do write(a*a);
```

```
for (a=10; a>=1; a--) printf(“%d”, a*a);
```

Confronto tra Pascal e C

(calcolo di x diviso y usando il metodo delle sottrazioni successive)

```
resto:=x; quoziente:=0;  
while resto>=y do  
begin  
    quoziente:=quoziente+1;  
    resto:=resto -y  
end;
```

```
for (resto=x, quoziente=0; resto>=y; quoziente=quoziente+1, resto=resto -y);
```

oppure più sinteticamente:

```
for (resto=x, quoziente=0; resto>=y; quoziente++, resto-=y);
```

Istruzioni di salto incondizionato

Il C ha un repertorio di istruzioni di salto incondizionato più ricco di quello messo a disposizione dal Pascal. Infatti non solo è disponibile una istruzione **goto** analoga a quella del Pascal, ma sono disponibili anche istruzioni che permettono di alterare il normale funzionamento di un ciclo.

- L'istruzione **goto** del C ha lo stesso effetto dell'istruzione **goto** del Pascal. Come nel Pascal l'istruzione è seguita da un identificatore di etichetta e la sua esecuzione provoca il salto incondizionato all'istruzione etichettata. È comunque sempre possibile ed è buona regola scrivere programmi che **non utilizzano** l'istruzione **goto**.
- Istruzioni di **uscita da un ciclo**. In C sono disponibili due istruzioni che consentono di alterare il normale controllo di un ciclo: l'istruzione **break** e l'istruzione **continue**. Nel Pascal **non esistono** istruzioni analoghe, che d'altra parte rendono i programmi non strutturati. Il loro utilizzo andrebbe quindi limitato a quei pochi casi in cui la programmazione strutturata renda il codice particolarmente inefficace o poco leggibile. L'istruzione **break** nella istruzione **switch** provoca l'uscita immediata dal ciclo che la contiene. L'istruzione **continue** provoca un salto alla parte di codice che valuta l'espressione di controllo nelle istruzioni **while** ed un salto alla parte di codice che modifica le variabili di controllo nell'istruzione **for**.

Esempi con le istruzioni **break** e **continue**

```
for (k=0;k<6;k++)
{
    if (k==3)
        continue;
    printf(“%d”,k);
}
```

```
k=0;
while (1) {
    if (k==3)
        break;
    printf(“%d”,k);
    k++;
}
```

L'esempio di sinistra scrive sullo schermo i numeri **0 1 2 4 5**; l'esempio di destra scrive sullo schermo i numeri **0 1 2**. Nel primo caso l'esecuzione della istruzione **continue** forza l'esecuzione della successiva iterazione del ciclo **for**; nel secondo l'istruzione **break** provoca l'interruzione del ciclo **while**.

Array

Il C consente di definire e gestire array in modo abbastanza simile al Pascal. Ci sono però alcune limitazioni che nascono dalla natura del C, più vicina del Pascal al linguaggio macchina, ed alcune differenze sintattiche.

Dichiarazioni di **array** nel Pascal e nel C

```
program dichiarazionearray;  
var a: array[1..10] of integer;  
    b: array['a'..'z'] of integer;  
    c: array[0..100, 0..100] of real;  
begin  
    a[2]:=100;  
    b['a']:=100;  
    c[2,5]:=1.1  
end.
```

```
main()  
{  
int a[10];  
int b[26];  
float c[101][101];  
    a[1]=100;  
    b[0]=100;  
    c[2][5]=1.1;  
}
```

Commento ai due programmi precedenti

- Dichiarazione dell'array **a**
Nel **Pascal** il tipo indice specifica l'estremo inferiore e l'estremo superiore dell'intervallo entro il quale è possibile indirizzare gli elementi dell'array.
Nel **C** vale la regola generale che tutti gli array sono indirizzati da interi a partire da 0 (ad esempio per accedere al secondo elemento occorre in C usare l'indice 1). La dichiarazione **int a[10]** dichiara che **a** è un array di dieci elementi interi, in cui il primo elemento ha indice **0** e l'ultimo ha indice **9**.
- Dichiarazione dell'array **b**
Visto che nel **C** non esiste il tipo intervallo e visto che l'indice di partenza di un array è sempre lo **0**, non resta che specificare che l'array è composto di **26** elementi. Per accedere alla posizione corrispondente ad '**a**' occorre accedere alla posizione **0**. La dichiarazione **int b['z']** sarebbe stata sintatticamente corretta, ma avrebbe generato un array di **122** elementi, dato che **122** è il codice ASCII di '**z**'. La dichiarazione **int b['z'-'a']**, definendo così un array di **26** elementi.
- Dichiarazione dell'array **c**
Essendo **c** un array bidimensionale, deve essere dichiarato in **C** come array di array. La sintassi della dichiarazione assomiglia alla seguente del **Pascal**:

```
var c: array [0..100] of array [0..100] of real;
```

```
      . . . . .  
c[2][5]:=1.1;
```

Inizializzazione di array

Una variabile di tipo array, come qualunque altra variabile del C, può essere inizializzata al momento della dichiarazione.

```
#include<stdio.h>

main() {
char a[5]={ 'g', 'a', 't', 't', 'o' };
int primi[5]={ 1, 2, 3, 5, 7 };
    printf(“%c”, a[1]);
    printf(“%d”, primi[4]);
}
```

Nell'esempio la prima istruzione **printf** stampa il carattere 'a' mentre la seconda stampa l'intero 7.

Inizializzazione di array multidimensionali

```
int matrice_diagonale [3][3] = {  
  { 1, 0, 0 } ,  
  { 0, 1, 0 } ,  
  { 0, 0, 1 }  
};
```

```
int matrice_diagonale [3][3] = {  
  1, 0, 0, 0, 1, 0, 0, 0, 1  
};
```

```
int primi [ ] = { 1, 2, 3, 5, 7 };
```

```
int matrice_diagonale_incompleta [3][3] = {  
  { 1, 0 } ,  
  { 0, 1, 0 } ,  
  { 0 }  
};
```

Confronto con il Pascal

- In C non esiste il concetto di **packed array**, tipico del Pascal.
- In C non viene effettuato alcun controllo sugli errori semantici statici e dinamici relativi agli array. Ad esempio se **a** è un array di 10 elementi, la presenza dell'istruzione **a[20] = 1** non causa alcun problema di compilazione e viene comunque eseguita; naturalmente ciò può causare effetti difficilmente controllabili sul programma.
- In C non è possibile scrivere l'istruzione **a = b** se **a** e **b** sono due variabili di tipo array, anche se i tipi di **a** e **b** coincidono. Non è neppure possibile confrontare array tramite operatori relazionali.
- In C gli elementi di un array sono memorizzati in locazioni contigue di memoria mentre nel Pascal non c'è alcuna garanzia che questo avvenga.

Stringhe

Un array di caratteri in cui i caratteri considerati significativi sono memorizzati consecutivamente a partire dall'indice **0** e contenente il carattere **'\0'** dopo l'ultimo carattere significativo è detto in C **stringa**.

In C è possibile inizializzare un array di caratteri usando una costante stringa:

```
#include<stdio.h>
main () {
char a[8]="gatto";
    printf("%c", a[1]);
    printf("%s", a);
}
```

L'inizializzazione **a[8]="gatto"**, oltre a scrivere i caratteri **'g','a','t','t','o'** negli elementi di **a** con indici compresi tra **0** e **4**, inserisce nell'elemento di indice **5** il carattere **'\0'**. In questo caso, visto che per la inizializzazione è usata una costante stringa, il C considera l'array **a** come una stringa.

L'esempio mostra anche come la funzione **printf**, specificando il formato tramite **%s**, possa essere usata per stampare una stringa. Vengono così stampati tutti i caratteri dell'array compresi tra quello di indice **0** ed il primo **'\0'** incontrato.

Contrariamente a quanto accade nel Pascal, le stringhe non possono essere confrontate con operatori relazionali. Le librerie standard contengono molte funzioni per la manipolazione di stringhe.