

PROGETTO 5

Scheda progetto	
Nome progetto	Potenza in N
Problema	Calcolo di una potenza con base ed esponente positivi.
Algoritmo creato	Potenza tra interi
Teoria di Pascal	- L'iterazione con condizione a posteriori - L'iterazione con condizione a priori

Questo progetto, finalmente, avrà una vera utilità: alla fine, infatti, saremo in grado di avere un programma capace di calcolare una qualsiasi potenza con base ed esponente positivi. Sarà inoltre la prima volta in cui creeremo un algoritmo *vero*: fino ad ora, infatti, avevamo utilizzato funzioni e procedure già predisposte nel linguaggio Pascal, mentre ora dovremo necessariamente lavorare anche nella progettazione, in quanto non esiste una funzione specifica al calcolo di una potenza.

Analisi del problema

Ora possiamo prevedere i casi in cui il nostro programma subirà degli errori di semantica nell'esecuzione, e possiamo quindi utilizzare opportune selezioni. Lavorando con variabili **longint** è infatti importante prevedere quando una delle due variabili è negativa, o quando l'esponente e la base sono uguali a zero.

Ecco la tabella di valori:

A	B	A ^B
0	0	imp.
0	1	0
1	0	1
1	1	1
0	x≠0	0
x≠0	0	1
x	1	x
1	x	1
-2	3	no
2	-3	no
-3	-4	no
2	3	8
4	2	16

Questa tabella risulta abbastanza complessa, all'apparenza, in quanto ci sono una marea di casi analoghi da prevedere (tutti i numeri diversi da zero elevati alla zero danno uno, tutti i numeri elevati a uno danno il numero stesso, ecc...).

In realtà dobbiamo ricordare **sempre** che il computer è stupido, ma il suo lavoro lo sa svolgere egregiamente. Cerchiamo di chiarire: se noi non prevedessimo ogni singola cosa, e inserissimo i valori 0 e 0, il programma cesserebbe di funzionare. Analogamente, se inserissimo un valore qualunque e 1, il computer, eseguendo il nostro algoritmo, arriverebbe sempre allo stesso risultato, ossia il numero di partenza. Con tutto questo volevamo affermare che, alla fin fine, i casi da prevedere sono solamente quelli che faranno cessare di funzionare il programma, quindi quelli di 0^0 e, nel nostro caso, di valori negativi (il programma continuerebbe a funzionare ugualmente, ma a noi interessa che lavori solo con numeri interi. Visto che lavoreremo con metodo indiretto, dovremo prevedere anche l'esponente uguale a 0.

Ricapitolando, i casi da prevedere sono:

1. $A = 0$ e $B=0$
2. $A < 0$
3. $B < 0$
4. $B=0$

Creazione dell'algoritmo

Ricordiamo la definizione di potenza:

Dati due numeri A e B, e chiamati rispettivamente BASE ed ESPONENTE, si dice potenza fra A e B e si legge "A elevato alla B", il prodotto di tanti fattori uguali alla base presi tante volte quanto lo indica l'esponente.

Ad esempio:

$$3^5 = 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3$$

Ecco allora che possiamo iniziare a ragionare su come creare l'algoritmo: dovremo leggere due numeri da tastiera (che saranno la base e l'esponente) e moltiplicare il primo per se stesso tante volte quanto lo indica il secondo. Potremo, ad esempio, assumere una variabile *conta* e assegnarle un valore che aumenta di uno ogni volta. Quando questa variabile eguaglia l'esponente, allora potremo smettere di moltiplicare la nostra base.

Ecco un algoritmo per il calcolo di una potenza:

1. Leggi A
2. Leggi B
3. **Se** A = 0
 - a. **Allora Se** B=0
 - i. Scrivi "Potenza impossibile"
4. **Se** A < 0
 - a. **Allora** scrivi "Dati non giusti"
5. **Se** B < 0
 - a. **Allora** scrivi "Dati non giusti"
6. **Se** B = 0
 - a. **Allora se** A≠0
 - i. **Allora** scrivi "Il risultato è 1"
7. **Se** tutte le condizioni di prima sono false
 - a. Assegna a C il valore di 1
 - b. Assegna a *conta* il valore di 0
 - c. **Ripeti** le seguenti istruzioni **finché** *conta* = C
 - i. Assegna a C il valore di (C•A)
 - ii. Assegna a *conta* il valore di (conta+1)

L'algoritmo è leggermente complicato e prevede un'anticipazione su alcuni lavori che svolgeremo in dettaglio nei progetti seguenti: gli operatori logici. In questo progetto utilizzeremo spesso l'operatore logico **and**, che restituisce un valore vero solo e solo se tutte le parti sono vere. In una condizione di selezione, delle condizioni legate con **and** fanno eseguire le istruzioni solo se **tutte** le condizioni sono soddisfatte.

Utilizzeremo anche **not** che, posto prima di una condizione, la fa risultare vera solo se quella originale era falsa e viceversa.

Nel punto 6, la condizione, lunghissima ma necessaria con le poche conoscenze che abbiamo, sarebbe:

((not ((A=0) and (B=0))) and (not (A<0)) and (not (B<0)) and (not (b=0)))

E' importante, a questo punto, che vi fermiate un secondo e cerciate di capire e leggere correttamente questa istruzione.

"Non è vero che A e B sono uguali a zero. Inoltre A e B non sono minori di zero e B non è uguale a zero."

A questo punto tutto è pronto: possiamo vedere il codice.

Creazione del programma in Pascal

Già da questo progetto, le parole riservate non saranno più in grassetto, ma in scrittura normale, fatta eccezione per quelle nuove che incontreremo ora per la prima volta.

```
program potenza_n;
uses crt;
var a,b,c,conta: longint;
begin
write ('Inserisci la base      ');
readln (a);
write ('Inserisci l''esponente  ');
readln (b);
if a=0 then
begin
if b=0 then
writeln ('Potenza impossibile');
end;
if b=0 then
begin
if a<>0 then
writeln ('Risultato:      1');
end;
if a<0 then
writeln ('Dati non giusti');
if b<0 then
writeln ('Dati non giusti');
if ((not ((A=0) and (B=0))) and (not (A<0)) and (not (B<0)) and
(not (b=0))) then
begin
conta:=0;
c:=1;
repeat
c:=c*a;
conta:=conta+1;
until conta=b;
writeln ('Risultato:      ',c);
end;
repeat until keypressed;
end.
```

Analisi del programma

Cerchiamo di capire cosa succede nel nostro programma: evitiamo di analizzare tutte le selezioni, in quanto ne abbiamo già parlato abbondantemente nel Progetto 4, e soprassediamo momentaneamente anche a quella lunghissima condizione piena di operatori logici, visto che l'abbiamo utilizzata solamente perché dobbiamo lavorare in un campo numerico strettissimo (l'insieme N è l'insieme di tutti i numeri naturali, ossia interi assoluti o positivi).

Puntiamo quindi la nostra attenzione su quello che veramente ci interessa: la serie di istruzioni compresa tra *repeat* e *until*. Se ben ricordate avevamo già accennato ad una struttura simile, quando, nel Progetto 1 approfondimento 3, avevamo visto l'istruzione *repeat until keypressed*, che serviva a far eseguire l'istruzione successiva solamente quando veniva premuto un qualunque pulsante della tastiera. Traducendo le due parole in italiano otterremmo che *repeat* significa *ripeti* e *until* significa *fino a che*. Questa struttura, infatti, serve a far ripetere una serie di istruzioni fino a quando non si verifica una condizione. Questa struttura di ripetizione ciclica è detta **iterazione** e, nei paragrafi seguenti, impareremo ad usarla in vari casi.

Un'ultima cosa va detta, anche se si tratta di una semplice curiosità. Quando si vuole scrivere un apostrofo con il comando *write* è necessario digitare due apici singoli consecutivi. (``).

TEORIA: L'iterazione

L'iterazione è una struttura predefinita di Pascal che consente di realizzare dei **cicli** detti **loop**, ossia consente la ripetizione di istruzioni per un numero finito di volte (se il ciclo fosse infinito si creerebbe un errore in fase di compilazione).

Esistono tre tipi di iterazioni:

- **Iterazione con condizione a priori**
- **Iterazione con condizione a posteriori**
- **Iterazione enumerativa**

Qualunque sia il tipo scelto, tuttavia, si devono rispettare delle condizioni fondamentali nella scrittura di un'iterazione:

1. Ogni iterazione deve essere **controllata da una condizione**, che determina la permanenza o l'uscita dal ciclo;
2. Prima di entrare nel ciclo si deve **inizializzare una variabile**, che comparirà nella condizione di controllo;
3. Nel ciclo devono essere presenti le **istruzioni che modifichino il valore della variabile** di controllo, in modo da poter far finire il ciclo.

In questo Progetto abbiamo visto l'iterazione con condizione a posteriori e vedremo quella con condizione a priori, nei prossimi Progetti studieremo la terza.

TEORIA: L'iterazione con condizione a posteriori

Partiamo analizzando il nome di questa iterazione: la nostra condizione, a quanto pare, verrà testata solo dopo aver eseguito le istruzioni del ciclo. Quindi saranno eseguite almeno una volta. In questo tipo di iterazione si utilizzano i termini riservati:

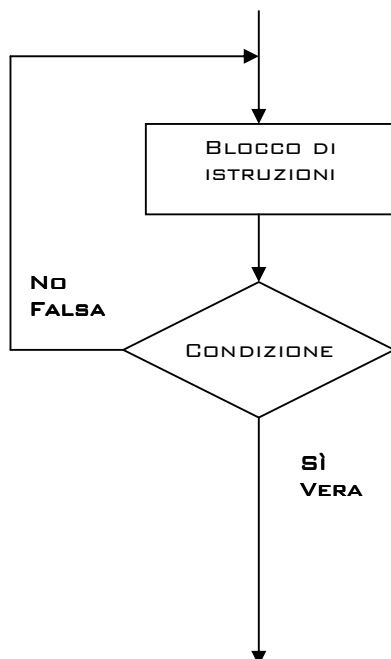
- **Repeat:** Indica l'inizio dell'iterazione e precede la serie di istruzioni del ciclo
- **Until:** Precede la condizione che determinerà l'uscita dal ciclo e segue le istruzioni.

Lo schema sarà quindi:

```
repeat
istruzione 1;
istruzione 2;
until (condizione);
```

La serie di istruzioni viene eseguita finché la condizione non risulta vera: viene eseguita, quindi, soltanto se questa è falsa.

Si può rappresentare l'iterazione con diagramma di flusso:



Si nota facilmente che il blocco di istruzioni viene eseguito almeno una volta, in quanto la condizione non viene testata all'inizio o prima del ciclo, ma solo dopo la prima esecuzione.

Inoltre l'iterazione prosegue solo se la condizione risulta falsa. Quando risulta vera, infatti, si esce dal ciclo.

Per questo motivo si parla anche di **iterazione per falso**.

Per controllare la condizione prima dell'esecuzione del ciclo è necessario inserire l'iterazione all'interno di uno schema di selezione con *if...then...else*.

```
if not(condizione) then
begin
repeat
blocco di istruzioni;
until (condizione);
end;
```

L'iterazione con condizione a posteriori segue dunque dei passaggi ben precisi:

1. Viene eseguito il ciclo
2. Viene verificata la condizione
 - a. Se è falsa il ciclo ricomincia
 - b. Se è vera si esce dal ciclo
3. Il ciclo continua ad essere eseguita fino a quando la condizione non diventa vera
4. Quando ciò succede, si esce dal ciclo

Prima di finire un'ultima precisazioni: a prescindere dal numero di istruzioni inserite nel ciclo, non è mai necessario annidare nuovamente *begin* e *end*; .

TEORIA: L'iterazione con condizione a priori

In questo progetto abbiamo parlato solo di iterazione con condizione a posteriori: esistono altri due tipi di iterazione: quello enumerativo, molto utile, di cui si parlerà diffusamente nel Progetto 7, e quella con condizione a priori. Quest'ultima è poco usata, in quanto è leggermente più difficile da usare e progettare. Ecco un esempio:

```
var a: integer;
begin
  readln (a);
  while (n>0) do
  begin
    writeln (n);
    n:=n-1;
  end;
end.
```

Notiamo subito i termini riservati utilizzati:

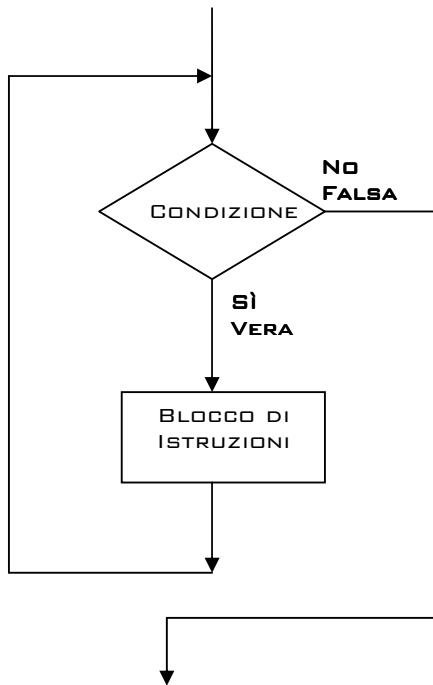
- **While:** *mentre*, precede la condizione.
- **Do:** *esegui*, precede la serie di istruzioni, e dovrà essere seguito da un *begin* se le istruzioni sono più di una, oppure potrà essere collegato direttamente se l'istruzione è una sola.

Gli schemi potranno essere dunque 2:

```
while (condizione) do
  istruzione;

while (condizione) do
  begin
    istruzione 1;
    istruzione 2;
    istruzione 3;
    istruzione n;
  end;
```

La serie di istruzioni viene eseguita solamente se la condizione risulta vera. Lo schema a blocchi è il seguente...



Osservando il diagramma, si nota come il blocco di istruzioni non venga eseguito per forza. Prima, infatti, viene testata la condizione e solo dopo aver avuto un risultato vero si passa all'esecuzione del ciclo.

La condizione, come già detto, deve risultare vera perché venga eseguito il blocco, e questo giustifica il secondo nome di questo tipo di iterazione: **iterazione per vero**.

Quando la condizione risulta falsa, si esce dal ciclo e si prosegue con le istruzioni successive. Bisogna ricordare sempre di modificare una variabile in modo che, prima o poi, questa faccia risultare falsa la condizione.

La progettazione di un'iterazione deve avvenire attentamente, onde evitare che il programma si trovi in un **loop infinito**, ossia che continui ad eseguire istruzioni all'infinito. L'iterazione è molto utile nella progettazione in linguaggio Pascal, ma bisogna ricordare di non esagerare nell'utilizzo delle sue funzioni.

TEORIA: Iterazioni nidificate

E' possibile, ovviamente, nidificare le iterazioni all'interno di se stesse.

Non è necessario annidare iterazioni di diverso tipo, è possibile ad esempio annidare una struttura *repeat...until* in una *while...do*, come accade nell'esempio seguente.

```
var a,n: longint;
begin
  readln (a,n);
  while n>0 do
    begin
      repeat
        writeln (a);
        a:=a-1;
      until a>0;
      writeln (n);
      n:=n-1;
    end;
end.
```

E' possibile costruire senza troppe i diagrammi di flusso per le iterazioni nidificate.

Appunti finali

Questo progetto è finito. Per una volta abbandoneremo la struttura iterativa e torneremo a quella selettiva. Nel Progetto 6, infatti, si parlerà di una selezione diversa da quella *if...then...else*.

Proposte

1. Scrivere i multipli di un numero inserito da tastiera ma minori di un altro inserito da tastiera.
2. Scrivere tutte le potenze di un numero m con esponente minore di n .
3. Modificare il programma del Progetto 3 in modo che continui a chiedere l'inserimento del divisore fino a quando questo non risulta diverso da zero. L'iterazione, in questo caso, deve essere accompagnata ad una selezione.
4. Scrivere i divisori di n , ricordando che un numero x è divisore di y se $y \bmod x$ è uguale a zero.
5. Scrivere il prodotto dei primi n numeri dispari.
6. Scrivere la somma dei primi n numeri pari.
7. Scrivere la somma dei numeri pari minori di n .

Riepilogo

Termini riservati nuovi	
do	Esegui, fai (struttura iterativa)
repeat	Ripeti (struttura iterativa)
until	Fino a che... (struttura iterativa)
while	Mentre (struttura iterativa)

Progetto creato da Squall1988

<http://www.pslife.net>

© PiEsseLife - 2003