

# PROGETTO 10

Scheda progetto	
<b>Nome progetto</b>	13_17
<b>Problema</b>	Dato un numero qualsiasi naturale, scrivere tutti i numeri minori ad esso finché non si incontra un multiplo di 13 o di 17
<b>Algoritmo creato</b>	Scrivere dei numeri in successione fino a quando non si incontra un multiplo di 13 o di 17
<b>Teoria di Pascal</b>	- Operatore OR - Operatore XOR - Tautologie e contraddizioni - Origine della selezione

Siamo arrivati al termine del nostro percorso di logica. Con questo progetto, infatti, saremo completamente in grado di lavorare con gli operatori logici, inserendoli in tutte le possibili condizioni. Nei progetti precedenti li avevamo visti nella struttura selettiva, mentre questa volta vedremo come possano agire anche in quella iterativa. Infine ci butteremo in cose estranee alla natura del Progetto, ma comunque importanti in logica e probabilmente utili nella programmazione futura. Finiremo anche il nostro percorso di logica, parlando di tautologie e contraddizioni una volta per tutte, analizzando tavole di verità complesse e parlando dell'origine della struttura selettiva. Ovviamente la logica è molto più complessa di quella che andremo ad analizzare noi, in quanto è possibile parlare di logica dei predicati, di dimostrazioni logiche, ma tutte queste cose non riguardano molto il nostro lavoro e quindi le trascureremo.

## **Analisi del problema**

Il testo risulta leggermente complicato ad una lettura veloce:

*Dato un numero qualsiasi naturale, scrivere tutti i numeri minori ad esso finché non si incontra un multiplo di 13 o di 17*

Parliamone con più calma. Dobbiamo prendere un numero qualsiasi letto quindi da tastiera e scrivere tutti i suoi numeri minori (fino ad arrivare al massimo a zero) finché non si incontra un multiplo di 13 o 17. Ad esempio, inserendo il numero 28, dovremmo scrivere 28 27 e fermarci qui, visto che il numero successivo, ossia 26, è divisibile per 13. Effettivamente bisogna prestare attenzione in questo caso, perché ogni minimo errore ci porterebbe in errore logico (leggete le lezioni di teoria per capire meglio di cosa parlo). Non perdiamo troppo tempo in descrizioni inutili, visto che comunque si tratta solo di leggere attentamente il testo. Ecco una

Numero	Scrivi
12	Troppo basso
27	27
36	36 35

tabella di valori:

Questa tabella sicuramente non è molto ampia, ma rende comunque l'idea del nostro programma e ci fa vedere che dovremo prevedere il caso in cui il numero inserito sia minore di 13 (il valore più basso fra 13 e 17). Potremmo sicuramente lasciar perdere, ma questa

volta abbiamo pensato di non lasciare tutto all'intelligenza di chi utilizza il programma: è il momento di vedere come un programma possa essere articolato.

## **Creazione dell'algoritmo**

Ecco l'algoritmo:

1. Leggi A
2. Se  $A < 13$ 
  - a. Scrivi "Numero troppo basso"
3. Altrimenti

- a. Ripeti queste istruzioni finché ( $A \bmod 13 = 0$ ) or ( $A \bmod 17 = 0$ )
  - i. Scrivi A
  - ii.  $A = A - 1$
- 4. Scrivi "Terminato"

A cosa serve il passo 4? Beh, effettivamente a niente, ma pensiamo all'inserimento di un valore già multiplo di 13 o 17, come 26. Il programma non scriverebbe nulla e noi rimarremmo piuttosto esterrefatti, quindi il passo 4 ci renderà consapevoli del fatto che non è avvenuto alcun errore, solo il numero inserito ha fatto terminare il ciclo fin da subito.

### Creazione del programma in Pascal

Ecco il listato. Prestate molta attenzione e leggete attentamente il testo. Questa volta abbiamo inserito le righe nel programma, in modo da rendere più facile i commenti. Ovviamente i numeri di riga non devono essere copiati.

```

1. - | program tredici_diciassette;
2. - | uses crt;
3. - | var a: integer;
4. - | begin
5. - |   write ('Inserisci un numero: ');
6. - |   readln (a);
7. - |   if a<13 then writeln ('Il numero e'' troppo basso')
8. - |   else
9. - |     begin
10.- |       if (a mod 13 = 0) or (a mod 17 = 0) then
11.- |         else
12.- |           begin
13.- |             repeat
14.- |               writeln (a);
15.- |               a := a-1;
16.- |             until (a mod 13 = 0) or (a mod 17 = 0);
17.- |           end;
18.- |         end;
19.- |       writeln ('Esecuzione terminata');
20.- |     repeat until keypressed;
21.- |   end.

```

### Analisi del programma

Guardiamo un secondo quello che succede nelle righe 10-11. Viene testata una condizione che non avevamo scritto nell'algoritmo. Tuttavia abbiamo scelto di utilizzare una iterazione con condizione a posteriori. Con questo tipo di loop le istruzioni vengono eseguite almeno una volta. Ma se noi inseriamo il valore 13 non bisogna scrivere nulla! Ecco il motivo di quel test. Ma guardiamo meglio il codice

```

if (a mod 13 = 0) or (a mod 17 = 0) then
else

```

Cosa succede dopo il then??? Assolutamente nulla. Questa struttura vuota serve solo per dire al programma che non deve eseguire nessuna istruzione se si verifica la condizione. In teoria dovrebbe esserci un punto e virgola, ma visto che dopo c'è un else non viene messo. In questo listato compare anche l'operatore di disgiunzione inclusiva, meglio conosciuto come OR.

### TEORIA: Operatore OR

L'operatore OR è detto di disgiunzione inclusiva. Si tratta in pratica di un operatore che rende vera una proposizione se anche solo una di quelle che la compongono è vera. Su 3000 proposizioni, basta che anche solo una sia vera che la loro unione tramite OR diventa vera. La disgiunzione inclusiva si riflette in insiemistica con l'unione tra insiemi.

Dati due insiemi  $A$  e  $B$ , si dice unione tra  $A$  e  $B$  e si scrive  $A \cup B$  l'insieme degli elementi di  $A$  o di  $B$ .

La congiunzione  $\cup$  indica il fatto che devono essere presi tutti gli elementi di tutti e due gli insiemi. La tavola di verità di OR è questa.

A	B	A or B
V	V	V
V	F	V
F	V	V
F	F	F

Si dice che il vero è l'**elemento assorbente**, mentre il falso è l'elemento **neutro**.

### TEORIA: Operatore XOR

Abbiamo visto che OR indica una disgiunzione inclusiva. Cosa significa il secondo termine, ossia il **inclusivo**? Semplicemente si indica che le due proposizioni possono coesistere. Facciamo un esempio pratico.

*Puoi mangiare una mela o una banana.*

Se noi intendiamo che si può mangiare una mela e non una banana, una banana e non una mela, ma anche una mela e una banana, allora avremo un

*Puoi mangiare una mela or una banana*

Invece lo XOR dice che le proposizioni non possono coesistere, nel senso che puoi mangiare una mela o una banana, ma non tutte e due. Si parla quindi di **disgiunzione esclusiva**.

*Puoi mangiare una mela xor una banana*

L'italiano non evidenzia questa distinzione e in effetti certe volte si specifica oralmente l'inclusività o l'esclusività delle proprie proposizioni.

*Prendi una mela o una banana, ma non tutte e due!*

*Prendi una mela o una banana, volendo anche tutte e due!*

Concludiamo con la tavola di verità di XOR.

A	B	A xor B
V	V	F
V	F	V
F	V	V
F	F	F

La tavola rispecchia esattamente quanto avevamo sostenuto in presenza, ossia il fatto che  $A \text{ xor } B$  è vero se le due proposizioni sono discordi, ossia non hanno valore uguale. E' invece falsa se hanno valori concordi, siano essi veri o falsi. Non abbiamo dedicato un progetto intero allo xor perché il suo utilizzo in generale è veramente raro e anche in logica spesso viene tralasciato.

Un ultimo appunto: i matematici, quando parlano di disgiunzione in generale, intendono quella di tipo inclusivo, ossia OR. E' importante ricordare tutto questo perché anche noi, nel corso della guida, utilizzeremo solo il termine disgiunzione, sottintendendo il termine inclusiva.

### Lavorare con proposizioni complesse

Abbiamo visto le tavole di verità di

- A and B
- A or B
- not A (e di conseguenza not B)
- A xor B

Tutto qui è molto semplice, ma come fare per risolvere espressioni logiche come

$(\text{not } A \text{ or } B) \text{ and } (B \text{ or not } A)$

Semplice, si eseguono tutte le operazioni richieste seguendo l'ordine in cui vengono incontrate, rispettando le parentesi e le precedenti degli operatori unari (not) su quelli binari. Si può fare riferimento alla tabella pubblicata nel Progetto 8.

Ecco la risoluzione di questa proposizione.

A	B	Not A	Not A or B	B or not A	(not A or B) and (B or not A)
V	V	F	V	V	V
V	F	F	F	F	F
F	V	V	V	V	V
F	F	V	V	V	V

Tutto sembra piuttosto semplice, vero?? Nelle proposte alla fine del progetto troverete anche delle proposizioni complesse da risolvere.

### Proprietà degli operatori logici AND e OR

Questa è solo una tabella del tutto inutile nel Pascal ma comunque necessaria nella risoluzione di proposizioni complesse.

Proprietà	OR	AND
Associativa	$(A \text{ or } B) \text{ or } C = A \text{ or } (B \text{ or } C)$	$(A \text{ and } B) \text{ and } C = A \text{ and } (B \text{ and } C)$
Commutativa	$A \text{ or } B = B \text{ or } A$	$A \text{ and } B = B \text{ and } A$
Distributiva	$(A \text{ and } B) \text{ or } C =$ $= (A \text{ or } C) \text{ and } (B \text{ or } C)$	$(A \text{ or } B) \text{ and } C =$ $= (A \text{ and } C) \text{ or } (B \text{ and } C)$
Idempotenza	$A \text{ and } A = A$	$A \text{ or } A = A$

Queste proprietà possono, in alcuni casi, semplificare il compito di risoluzione.

### Tautologie e contraddizioni

Abbiamo già parlato nello scorso progetto di tautologie e di contraddizioni. Le contraddizioni sono proposizioni che restituiscono sempre un valore falso. Con **tautologie** intendiamo invece proposizioni che restituiscono sempre vero. Ecco un paio di esempi di tautologie e contraddizioni.

- Tautologie
  - o  $A \text{ or } (\text{not } A)$
  - o  $\text{not } A \text{ or } A$
- Contraddizioni
  - o  $A \text{ and not } A$
  - o  $\text{not } (A \text{ or } B) \text{ and } A$

### Origine della selezione: l'implicazione

Partiamo da un esempio: un giovane studente è interessato ad iscriversi ad un torneo di corsa, pertanto si presenta al botteghino delle iscrizioni e domanda quali siano le condizioni necessarie per essere ammessi alla corsa. Il responsabile risponde con questa frase:

"Possono partecipare gli atleti non minorenni oppure autorizzati dai genitori"

Non iniziate a chiedervi il motivo per cui dica "non minorenni" invece di "maggioirenni", questo piccolo trucchetto ci serve per raggiungere il nostro scopo.

Se riportiamo la risposta del responsabile nel campo della logica, possiamo ottenere due proposizioni:

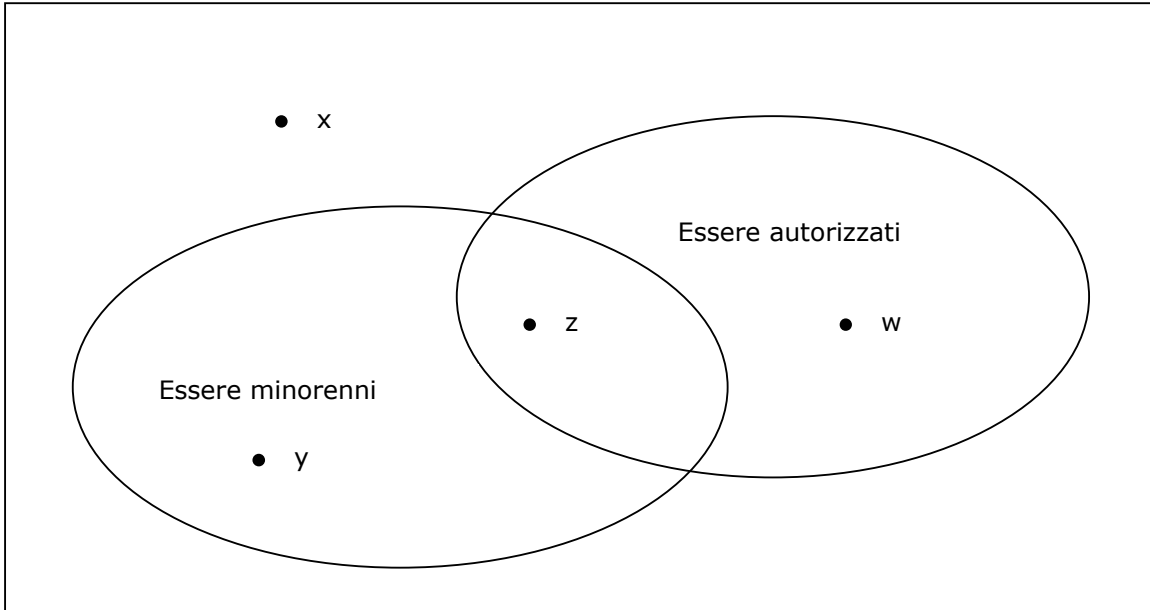
A => Essere minorenni

B => Essere autorizzati dai genitori

Pertanto la risposta del responsabile, tradotta in logica, si tramuta in

**not A or B**

Osserviamo ora gli insiemi dell'esempio e proseguiamo con il ragionamento:



Individui X

Essi non sono minorenni e non sono nemmeno autorizzati dai genitori.  
Possono partecipare alla corsa.

Individui Y

Essi sono minorenni e non sono nemmeno autorizzati dai genitori.  
Non possono partecipare alla corsa.

Individui Z

Essi sono minorenni, ma sono autorizzati dai genitori.  
Possono partecipare alla corsa.

Individui W

Essi non sono minorenni e sono autorizzati dai genitori.  
Possono partecipare alla corsa.

Abbiamo quindi visto che gli unici individui che possono gareggiare sono quelli dell'insieme

$$\bar{A} \cup B$$

che in sostanza equivale all'espressione di not A or B

Ecco la tabella di verità di questa proposizione:

<b>A</b>	<b>B</b>	<b>Not A</b>	<b>Not A or B</b>
V	V	F	V
V	F	F	F
F	V	V	V
F	F	V	V

Ed eccoci così giunti al punto di passare da questa condizione a quella dell'implicazione: invece di rispondere "non minorenni o autorizzati", il responsabile avrebbe potuto dire

"Se l'atleta è minorenne, allora deve essere autorizzato dai genitori"

Che, già a prima vista, equivale all'espressione precedente.  
In logica si dice che A implica B e si scrive

$A \rightarrow B$

Non ci soffermiamo troppo su questo concetto, perché questa struttura in logica è profondamente diversa da quella della programmazione e rischieremmo solamente di fare confusione. Concludiamo semplicemente dicendo che la tabella di verità dell'implicazione equivale a quella di not A or B.

### Proposte

1. Scrivere i successivi multipli di un numero naturale, fino a quando non se ne incontra uno maggiore di 200 o divisibile per 7.
2. Se un numero è "dispari minore di 50" oppure multiplo di 11 raddoppiarlo, altrimenti scrivere i suoi precedenti fino ad ottenerne uno minore di 10 oppure multiplo di 13.

### Riepilogo

Termini riservati nuovi	
<b>Or</b>	Disgiunzione inclusiva
<b>Xor</b>	Disgiunzione esclusiva

Progetto creato da Squall1988

<http://www.pslife.net>

© PiEsseLife - 2003