PASCAL ZERO

Settembre 2002
Enrico Centenaro `enrico@matematiche.org`

Indice

PASCAL ZERO *

Enrico Centenaro†

1999-word, 2002-LaTeX

Sommario

Il Pascal è un esempio di linguaggio di programmazione ad alto livello. Ci sono molti altri linguaggi di programmazione e ognuno ha i suoi vantaggi e svantaggi.

L'animo di questo scritto è quello di far conoscere le basi del linguaggio Pascal . Inizieremo con alcuni esempi che ci serviranno per introdurre i concetti della programmazione in Pascal . Non trattenetevi dallo sperimentare con i programmi che presenteremo, non prendete tutte le cose alla lettera e chiedete[1] quando avete dei dubbi: è il miglior modo per imparare.

1 Iniziamo

Cominciamo con lo scrivere un piccolo programma e mostrare come funziona. Il Pascal comprende certe istruzioni e parole chiave che noi scriveremo in lettere maiuscole, mentre il resto sarà scritto in lettere minuscole. I principianti è bene che facciano particolare attenzione a come viene scritto il programma, specialmente alla punteggiatura e alla spaziatura.

1.1 Esempio (my_prog.pas)

Utilizzando il Pascal che hai a disposizione, scrivi questo programma, sostituisci `A Programmer` con il tuo nome.

```
PROGRAM my_first; {short demonstration program}
 VAR              {by A Programmer}
  index : INTEGER;
 BEGIN
  FOR index := 1 TO 10 DO
  BEGIN
    WRITELN(index)
  END
END.
```

Salva il programma col nome **my_prog.pas**.

Dopo aver scritto un programma e averlo digitato, il passo successivo è la sua *compilazione*. Un *compilatore* controlla il programma cercando eventuali errori, cioè errori di digitazione delle istruzioni dal Pascal o errori nella sintassi.

*Questo documento è stato scritto utilizzando Latex *www.latex-project.org* (vedi [4] e [5]

†I programmi che vengono usati in questo documento sono tratti da uno scritto di John Brewer. Tutti i sorgenti sono disponibili all'indirizzo `http://www.matematiche.org/docs/odp2/progs.zip`

[1] In verità non vi sono domande stupide... molte risposte lo sono.

Se vengono riscontrati degli errori, questi devono essere corretti[2] prima di andare avanti. Questa fase è solitamente la più difficoltosa e spesso serve parecchi tempo affinché un programma funzioni correttamente, un consiglio: fare attenzione alla punteggiatura!

Dopo aver compilato il programma siete pronti a farlo *eseguire* e vedere se compie quello che avevate preventivato[3]...

Ebbene, eseguite il programma **my_prog** e notate cosa succede.

Adesso torniamo indietro al programma che abbiamo scritto e cerchiamo di capire *cosa abbiamo scritto*. Concentriamo la nostra attenzione riga per riga.

```
PROGRAM my_first;            {short demonstration program}
```

Tutti i programmi Pascal iniziano con la parola chiave `PROGRAM` seguita da un nome e da un punto e virgola. La parte restante della linea è racchiusa fra parentesi graffe. Tutto quello che viene inserito fra parentesi graffe viene ignorato dal computer. La ragione la comprendi nell'esempio stesso: quello che leggi ti indica quello che viene fatto dal programma in quella posizione. Per tale motivo spesso queste aggiunte vengono chiamate anche *commenti*[4]. Quando scrivi un programma aggiungi sempre dei commenti, perché lo rendono più chiaro e leggibile.

```
VAR                          {by A Programmer}
```

`VAR` è l'abbreviazione di variabile. Se il tuo programma usa dei numeri, o delle variabili, allora questo è il posto dove devono essere dichiarate le variabili, cioè come si chiamano e quale tipo di informazione potranno rappresentare. Come in precedenza il commento che termina la riga è ignorato dal computer.

```
  index : INTEGER;
```

... il nostro programma utilizza solamente una variabile. Essa si chiama `index` e rappresenta un numero intero o naturale.

Ora possiamo passare all'inizio del programma principale. Esso inizia sorprendentemente[5] con...

```
BEGIN
```

Notate che adesso che quello che leggete è rientrato rispetto il margine sinistro della pagina, in questo modo si ottiene un effetto grafico sol programma nel suo insieme. Questa tecnica di scrittura si chiama indentazione e viene utilizzata esclusivamente per ragioni estetiche e per rendere più chiaro e leggibile il programma.

```
  FOR index := 1 TO 10 DO
```

Questo è l'inizio di quello che viene comunemente chiamato *ciclo* o *loop*. Un ciclo è una istruzione (o insieme di istruzioni) che vengono eseguite ripetutamente[6].

```
  BEGIN
```

Questa istruzione definisce l'inizio della o delle e istruzioni del loop, che sono:

```
    WRITELN(index)
```

[2] Ogni versione del Pascal lo fa in modo più o meno completo.
[3] Un altro paio di maniche!
[4] Se non trovate le parentesi graffe nella vostra tastiera, le potete sostituire con `(*` e `*)`
[5] ...ovviamente sto scrivendo con tono ironico: ricordo che *begin* significa *inizio*
[6] E questo è quello che il computer è adatto a fare)

... in questo caso il ciclo contiene una sola istruzione. Esse serve a *stampare* il valore della variabile `index`. Si noti l'indentazione.

```
END
```

Questa istruzione marca la fine del gruppo di istruzioni del ciclo.

```
END.
```

Notate che questo `END` termina col punto, esso serve a segnalare la fine del programma.

Facciamo adesso un paio di cambiamenti al programma:

- Carichiamo il programma **my_prog.pas** e cambiamo la linea 6 con `FOR index := 1 TO 20 DO`, ricompiliamo il programma e eseguiamo, cosa succede?
- Cambiamo la linea 8 con `WRITELN('hello world')`, ricompiliamo il programma e eseguiamo, cosa succede?
- Cambiamo la linea 8 con `WRITELN(index, ' volte 3 è', index*3)` ricompiliamo il programma e eseguiamo, cosa succede?

Fino ad ora abbiamo visto che un programma Pascal è fatto da un certo numero di righe di istruzioni. Per poter eseguire il programma bisogna compilare il file contenente le istruzioni.

Questo potrebbe sembrare un po' strano, ma per adesso non preoccupiamoci più di tanto, vediamo piuttosto qualche altro esempio.

1.2 Esempio (stamps.pas)

Dal menù `File` del nostro programma scegliamo `New` e digitiamo il seguente programma.

```
PROGRAM stamps;          {calculates the cost of stamps} VAR {by A
Programmer}
  cost   : REAL;
  number : INTEGER;
  total  : REAL;
BEGIN
  cost := 0.65;
  WRITELN('enter the number of stamps');
  READLN(number);
  total := number * cost;
  WRITELN('you ordered ', number, ' stamps');
  WRITELN('the total cost is ', total)
END.
```

Come al solito salviamo e compiliamo il programma ed eventualmente correggiamo gli errori.

Dovrebbe essere evidente che il risultato della esecuzione del programma non è proprio quello che ci si aspettava. Per capire perché, vediamo più da vicino il funzionamento del programma. Inizialmente il programma utilizza tre numeri, o variabili, chiamate `cost`, `number` e `total`. La variabile chiamata `number` rappresenta un numero intero perché non avrebbe senso avere 0.7 francobolli. Le altre variabili chiamate `cost` e `total` non essendo sempre numeri interi sono descritte come tipo `real` che significa che possono avere una parte decimale. I numeri che vedete sono rappresentazioni scientifiche dei numeri (che possono sembrare piuttosto astruse), per ovviare a questo inconveniente basta sostituire la penultima riga con:

```
WRITELN('the total cost is ', total :5:2);
```

Quello che abbiamo aggiunto e solo un po' di *formattazione*, facendo in modo che quando il numero `total` viene stampato, esso occuperà 5 spazie e due di questi saranno utilizzati per la parte decimale.

Il programma che abbiamo appena scritto è differente dal precedente, perché in questo l'utente deve inserire dalla tastiera dei dati. Il programma altrimenti non funziona. Questo tipo di programmi si chiamano *interattivi* come la maggior parte del software che utilizzate. Un possibile miglioramento del programma potrebbe essere quello di dare la possibilità di decidere il valore dei francobolli, che adesso è fissa a 65 centesimi (`cost := 0.65;`). Come fare? Basterà sostituire la settima riga con due righe simili alla ottava e alla nona. Provare!

Esercizio

2 Ideare un Programma

Quando dovete realizzare un programma, la prima cosa da fare è decidere esattamente cosa il programma dovrà fare. Ciò vi può sembrare un tantino ovvio, ma se fate errori adesso sarà difficile che il programma soddisfi le vostre esigenze. Ci sono quattro importanti passaggi da considerare:

1. *analizzare il problema*, decidere come il programma risolverà il problema
2. *codificare il programma* - scrivere il codice in Pascal
3. *testare il programma* - controllare se i risultati prodotti sono corretti
4. *valutare la soluzione* - il programma fa quello che ci si aspettava? lo si può migliorare?

Chiariamo questi concetti attraverso un esempio.

2.1 Esempio (bus.pas)

Una ditta di trasporti richiede un programma per calcolare il costo di un viaggio conoscendo il costo per miglio e la lunghezza del viaggio. Per esempio, se la ditta ha una tariffa di \$0.75 per miglio e il percorso è lungo 15 miglia il costo totale sarà $0.75 \times 15 = \$11.25$.

2.1.1 Trovare una soluzione

Molti programmi sono strutturati secondo un semplice schema:

1. informazioni sull'input
2. elaborazione delle informazioni
3. output dei risultati

Noi useremo questo approccio per risolvere questo esempio. Iniziamo con:

1. ottenere le informazioni dall'utente
2. calcolare il costo del viaggio
3. stampare il risultato

identificatore	rate	miles	cost
tipo	real	integer	real

Tabella 1: Tabella tipi

Ora prendiamo in considerazione ognuno di questi passaggi e vediamo nel dettaglio come deve essere realizzato. Il primo passo è quello di ottenere dall'utente le informazioni richieste. Di quali informazioni abbiamo bisogno? Ricordate che l'utente deve capire cosa vuole il programma altrimenti egli non sa cosa fare...

1. ottenere le informazioni dall'utente
   (a) chiedere all'utente di inserire la tariffa per miglio
   (b) leggere la tariffa per miglio
   (c) chiedere all'utente di inserire la lunghezza del percorso in miglia
   (d) leggere la lunghezza del percorso

   Si noti che abbiamo scomposto il primo punto in ulteriori tre punti che hanno raffinato il procedimento. Facciamo la stessa cosa anche per i punti successivi.

2. calcolare il costo del viaggio
   (a) calcolare $costo = rata \times miglia$

   Il passaggio successivo è l'output dei dati:

3. output dei risultato
   (a) stampare la lunghezza del percorso
   (b) stampare la quota per miglio
   (c) stampare il costo

Riassumiamo i passaggi che abbiamo scritto:

```
1.1 chiedere all'utente di inserire la tariffa per miglio
1.2 leggere la tariffa per miglio
1.3 chedere all'utente la lunghezza del percorso in miglia
1.4 leggere la lunghezza del percorso

2.1 calcolare costo=rata\times miglia

3.1 stampare la lunghezza del percorso
3.2 stampare la quota per miglio
3.4 stampare il costo
```

É importante che si comprenda il lavoro che abbiamo fatto: non c'è niente di superfluo. E' così che si procede per realizzare un programma!

2.1.2 Scrivere il programma

Il programma opera con tre numeri - il numero delle miglia, la quota per miglio e il costo del viaggio. Dobbiamo decidere i) come chiamare queste variabili, ii) quale tipo di numeri rappresentano.

Potremmo fare le scelte rappresentate nella tabella 1.

Ricordiamo che i numeri interi sono `integer` e i numeri che invece hanno una parte decimale sono chiamati `real`. Scriviamo il codice.

```
PROGRAM bus_journey; {to calculate journey cost}
VAR {by A Programmer}
  rate  : REAL;
  miles : INTEGER;
  cost  : REAL;
BEGIN
  WRITELN('enter the rate per mile ');
  READLN(rate);
  WRITELN('enter the number of miles ');
  READLN(miles);
  cost := rate * miles;
  WRITELN('number of miles = ', miles);
  WRITELN('rate per mile = $', rate);
  WRITELN('cost of journey = $', cost)
END.
```

Si noti quanto è simile al progetto che abbiamo fatto (quello che viene talvolta chiamato anche *pseudocodifica*)

Adesso riprendi il computer, digita il programma e salvalo col nome di **bus.pas**. Compilalo ed eseguilo.

Un'altra volta vediamo che i risultati non sono proprio quello che ci si aspettava, ma abbiamo già risolto un problema simile... Editiamo il file **bus.pas** e modifichiamo la terzultima e la penultima riga come riportato di seguito:

```
WRITELN('rate per mile = $', rate :5:2);
WRITELN('cost of journey = $', cost :5:2);
```

Compila ed esegui il programma.

Perché non è necessario modificare la linea che segue? Esercizio

```
WRITELN('number of miles = ', miles);
```

Facciamo qualche altra modifica al programma, editiamo la linea 7 nel seguente modo:

```
WRITE('enter the rate per mile ');
```

Abbiamo utilizzato `WRITE` al posto di `WRITELN`. Modifichiamo anche la riga 9 nel seguente modo:

```
WRITE('enter the number of miles ');
```

Compiila ed esegui il programma. Descrivi le differenze che le modifiche hanno apportato. Quale preferisci? Salva questa versione del programma. Esercizio

2.1.3 Verifica del funzionamento (meglio *testing*)

Adesso che il programma 'gira' , dobbiamo verificare che effettivamente elabora i dati secondo le nostre esigenze e come lo avevamo ideato. Per esempio potremmo abbozzare una tabella (vedi 2) e verificare varie possibilità. Cosa succede se inseriamo 6.5 come numero di miglia? Dopotutto si suppone che sia un numero intero...

Copia e completa la tabella 2. Esercizio

La valutazione del programma dovrebbe descrivere il suo funzionamento, evidenziare i limiti del suo utilizzo e suggerire eventuali miglioramenti. Per esempio:

rate	miles	risposta attesa	risultato o commento
2	50		
0.5	100		
0	100		
3	6.5		
3	mmm		

Tabella 2: tabellaverifica

Il programma fornisce risultati corretti se si utilizzano dei numeri interi. Se l'utente inserisce valori con decimali quando specifica la lunghezza del percorso allora si verifica un 'run-time error' che ferma l'esecuzione del programma. Questo succede anche quanto l'utente inserisce dei valori che non sono numerici.

2.1.4 Valutazione

La valutazione dovrebbe dire se il programma fa quello per cui era stato pensato e come potrebbe essere migliorato. Per esempio:

Il programma calcola correttamente il costo di un viaggio basandosi sulle informazioni che l'utente fornisce. Ci sono degli errori 'Runt-ime' se l'utente non inserisce correttamente il tipo di dati richiesti. Sarebbe bene migliorare il programma in modo da segnalare gli errori di inserimento dei dati e permettere un nuovo inserimento, in altre parole fare una migliore gestione degli errori.

2.2 Esempio (fencing.pas)

Un giardiniere richiede un programma che calcoli la lunghezza della rete che serve a recintare un giardino quadrato, note la lunghezza e la larghezza.

2.2.1 Trovare una soluzione

Usando uno schema simile a quello adottato nell'esempio precedente, possiamo iniziare con:

1. ottenere le informazioni dall'utente
2. calcolare la lunghezza della recinzione
3. stampare il risultato

Come abbiamo fatto in precedenza sviluppiamo nel dettaglio i punti che hanno scomposto il nostro problema. Di quali informazioni abbiamo bisogno? Come possiamo calcolare la lunghezza della recinzione?

1. ottenere le informazioni dall'utente
   (a) chiedere all'utente di inserire la lunghezza del giardino
   (b) leggere la lunghezza
   (c) chiedere all'utente di inserire la larghezza del giardino
   (d) leggere la larghezza

Nome	Tipo della variabile
`length`	
`width`	
`total`	

Tabella 3: tabella tipi

2. calcolare la lunghezza della recinzione
   (a) calcolare $lunghezza\ recinzione = 2 \times lunghezza + 2 \times larghezza$
3. output dei risultato
   (a) stampare la lunghezza del giardino
   (b) stampare la larghezza del giardino
   (c) stampare il totale

Riassumiamo i passaggi che abbiamo scritto:

```
1.1 chiedere all'utente di inserire la lunghezza del giardino
1.2 leggere la lunghezza
1.3 chiedere all'utente di inserire la larghezza del giardino
1.4 leggere la larghezza

2.1 calcolare lunghezza recinzione=2 x lunghezza + 2 x larghezza

3.1 stampare la lunghezza del giardino
3.2 stampare la larghezza del giardino
3.4 stampare il totale
```

É importante che si comprenda il lavoro che abbiamo fatto, non c'è niente di superfluo: è così che si procede per realizzare un programma!

2.2.2 Scrivere il programma

Il programma ha bisogno di utilizzare tre numeri - la lunghezza e la larghezza del giardino e la lunghezza totale della recinzione. Quale tipo di numeri dovrebbero essere? Interi o decimali? Decidi quale a quale tipo dovranno appartenere (`integer` o `real`) queste variabili e trascrivilo sul tuo quaderno ricopiando la tabella 3. Esercizio

Apri il tuo editor Pascal e scrivi il codice del programma, puoi aiutarti con l'esempio che abbiamo visto in precedenza. Salvalo col nome `fencing.pas`. Compila ed esegui il programma.

2.2.3 Verifica del funzionamento (meglio *testing*)

Adesso che il tuo programma funziona[7], Esercizio

- completa una tabella simile a quella dell'esercizio precedente (vedi 2) scegliendo dei numeri opportuni per testare il programma
- scrivi una breve relazione su quello che hai fatto

2.2.4 Valutazione

Sempre basandoti su quanto fatto nell'esercizio precedente, scrivi una relazione che valuti il programma fatto. Esercizio

2.3 Esempio (darts.pas)

Il gestore di un bar richiede un programma che calcoli il punteggio di un giocatore dopo che ha tirato tre freccette[8]. Il programma dovrebbe mostrare i tre punteggi e i tre totali progressivi.

2.3.1 Trovare una soluzione

Scrivi un procedimento che risolva il problema. Mostralo al tuo insegnante

2.3.2 Scrivere il programma

Scrivi il codice del programma. Salva il programma col nome **darts.pas**. Compila ed esegui il programma

2.3.3 Verifica del funzionamento (meglio *testing*)

Come negli esercizi precedenti prepara una tabelle con degli opportuni dati. Scrivi una breve relazione sui risulti.

2.3.4 Valutazione

Scrivi una relazione del tuo programma. Mostrala al tuo insegnante.

2.4 Esempio (average1.pas)

Un insegnante vuole un programma che calcoli la media dei tre test fatti dai suoi studenti. Ogni test è composto da 60 quesiti. Il programma dovrebbe calcolare la media per i tre test e per tutti i quesiti.

2.4.1 Trovare una soluzione

Scrivi un procedimento che risolva il problema. Mostralo al tuo insegnante

2.4.2 Scrivere il programma

Scrivi il codice del programma. Salva il programma col nome **average1.pas**. Compila ed esegui il programma.

2.4.3 Verifica del funzionamento (meglio *testing*)

Come negli esercizi precedenti prepara una tabelle con degli opportuni dati. Scrivi una breve relazione sui risulti.

2.4.4 Valutazione

Scrivi una relazione del tuo programma. Mostrala al tuo insegnante.

[7] Ovviamente questo richiederà un po di tempo...

[8] Stiamo parlando del tiro delle freccette', un gioco che nei paesi anglosassoni è diffuso come le bocce' lo sono da noi.

2.5 Esempio (soccer.pas)

UL'editore di un giornale sportivo vorrebbe un programma che calcoli il punteggio di una squadra di calcio sapendo il numero delle partite vinte, pareggiate e perse ha disputato.

2.5.1 Trovare una soluzione

Scrivi un procedimento che risolva il problema. Mostralo al tuo insegnante.

2.5.2 Scrivere il programma

Scrivi il codice del programma. Salva il programma col nome **soccer.pas**. Compila ed esegui il programma

2.5.3 Verifica del funzionamento (meglio *testing*)

Come negli esercizi precedenti prepara una tabelle con degli opportuni dati. Scrivi una breve relazione sui risulti.

2.5.4 Valutazione

Scrivi una relazione del tuo programma. Mostrala al tuo insegnante.

3 Variabili Testo

Tutti i programmi che abbiamo scritto fino ad ora hanno utilizzato variabili *numeriche*[9]. Ogni variabile ha un suo nome ed appartiene a un certo tipo. Le variabili possono essere usate nel calcolo come per esempio:

```
average := total/3;
```

In questo caso, `average` viene calcolata in funzione di una'altra variabile `total` dividendola per 3. Ci sono altri tipi di variabili...

Le variabili per rappresentare del testo sono spesso chiamate *stringhe*. Un esempio potrebbe essere una variabile contenente un nome.

3.1 Esempio (age.pas)

Scrivi il seguente programma:

```
PROGRAM text; {using text variables}
VAR {by A Programmer}
  name : STRING[10];
  age  : INTEGER;
BEGIN
  name := 'A Programmer'; {use your name here}
  age := 15;
  WRITELN('my name is ', name);
  WRITELN('my age is ', age)
END.
```

La variabile chiamata `name` è di tipo stringa:

```
name : STRING[10];
```

[9] cioè numeri

Il numero 10 tra parentesi quadrate indica che la stringa può contenere fino a 10 caratteri. Questo programma non fa niente di particolare: stampa sempre gli stessi valori ogni volta che viene eseguito. Allora modifichiamolo in modo che diventi:

```
PROGRAM text; {using text variables}
VAR
  name : STRING[10];
  age  : INTEGER;
  days : INTEGER;
BEGIN
  WRITE('enter your name : ');
  READLN(name);
  WRITE('enter your age  : ');
  READLN(age);
  days := 365 * age;
  WRITELN('hello ', name);
  WRITELN('you are ', age);
  WRITELN('you must be at least ', days, ' days old!')
END.
```

Salva il programma col nome **age.pas**, compila ed esegui.

Con le modifiche apportate il programma adesso stampa valori diversi ogni volta che viene eseguito perché chiede delle informazioni all'utente, e in base a quelle costruisce l'output.

Esercizio

Notate che le programma che avete salvato vengono usate le istruzioni `WRITE` e `WRITELN`, ricordate la loro differenza? Ritornate ad editare il programma e utilizzate `WRITELN` al posto di `WRITE`. Avete capito? Scrivete sul vostro quaderno una breve descrizione sulle differenze fra le due istruzioni, spiegate quale secondo voi lavora meglio. Cosa succede se inserite dei nomi più lunghi di 10 caratteri?

4 Punteggiatura

Avrete certamente notate che alcune righe del codice Pascal che abbiamo scritto terminano con un punto e virgola e altre no. Ti sarà certamente capitato di avere degli errori nella compilazione dei programmi perché avevi dimenticato di trascrivere un punto e virgola, è un tipico errore che si fa.

Ma qual è la regola? Quando i punti e virgola sono richiesti e quando non lo sono?

I punti e virgola servono a *separare* fra loro le istruzioni del Pascal , così il compilatore capisce dove finisce una istruzione e ne inizia un'altra. Ecco un esempio tratto da uno dei programmi che abbiamo già scritto:

```
  WRITELN('you are ', age);
  WRITELN('you must be at least ', days, ' days old!')
END.
```

Il punto e virgola separa le due istruzioni `WRITELN`, ma non c'è la necessità di mettere nessun segno alla fine della seconda riga perché `END` è una parola chiave e non una istruzione.

4.1 Esempio (percent.pas)

Questo programma chiede all'utente di inserire il suo nome, il numero di risposte corrette che ha dato in un test e il numero di risposte corrette totali. Come output il programma stamperà la percentuale del risultato. Per esempio, Stefano Squalo ha risposto correttamente a 15 domande su 20, perciò la percentuale è 75%.

4.1.1 Trovare una soluzione

Scrivere un analisi completa del programma. Mostrarla all'insegnante.

4.1.2 Scrivere il Programma

Scrivere il codice Pascal . Salvare il file con il nome **percent.pas**, compilare ed eseguire il programma.

4.1.3 Verifica del funzionamento

Come al solito, completare una tabella con opportuni dati in input per testare il programma.

Scrivere una breve relazione sui risultati ottenuti.

4.1.4 Valutazione

Scrivi una valutazione sul programma. Mostra il tuo lavoro all'insegnante

4.2 Esercizio chips1.pas

In un centro giovanile il Venerdì, il Sabato e la Domenica vengono venduti sacchetti di patatine di vari gusti a 65c l'uno. Scrivere un programma che richiede il gusto delle patatine, il numero di sacchetti venduti per ogni singolo giorno e restituisce il ricavo per ogni singolo giorno e il totale dei tre giorni.

4.2.1 Trovare una soluzione

1. ottenere le informazioni dall'utente
2. calcolare i soldi ricavati
3. stampare i risultati

Perfezionando ogni singolo passo abbiamo:

1. ottenere le informazioni dall'utente
   (a) chiedere all'utente di inserire il gusto delle patatine
   (b) leggere il gusto
   (c) chiedere all'utente di inserire il numero di sacchetti venduti il Venerdì
   (d) leggere il numero di sacchetti venduti il Venerdì
   (e) chiedere all'utente di inserire il numero di sacchetti venduti il Sabato
   (f) leggere il numero di sacchetti venduti il Sabato
   (g) chiedere all'utente di inserire il numero di sacchetti venduti la Domenica
   (h) leggere il numero di sacchetti venduti il Domenica
2. calcolare i soldi ricavati
   (a) calcola il totale di sacchetti venduti
   (b) calcola il totale ricavato
3. stampare i risultati
   (a) scrivere il nome del gusto
   (b) scrivere il numero dei sacchetti venduti
   (c) scrivere il totale ricavato

4.2.2 Scrivere il Programma

- In base all'analisi del programma e ai valori che vengono elaborati dal programma decidere quante e quali variabili utilizzare.
- Quali sono i tipi ai quali assegnare le variabili?
- Come conviene chiamare le variabili?
- Se non sei certo delle tue scelte, prima di continuare chiedi al tuo insegnante.
- Utilizzando l'analisi precedente, scrivi il codice del programma.
- Salva il programma col nome **chips1.pas**.
- Compila ed esegui il programma.

4.2.3 Verifica del funzionamento (meglio *testing*)

- Come al solito, compila una tabella con opportuni dati in input per testare il programma.
- Scrivi una breve relazione sui risultati ottenuti.

4.2.4 Valutazione

Scrivi una valutazione sul programma e preveda dei possibili miglioramenti

> *D'ora in avanti non chiederò più espressamente di completare il terzo e quarto passo (Testing e Valutazione). Questa operazione deve essere fatta sempre e non perché è l'insegnante che te lo richiede. Ecco perché quando avrai dei problemi in questi due passaggi devi chiedere al tuo insegnante.* ***Sul tuo quaderno gli esercizi devono essere svolti sempre secondo lo schema che abbiamo seguito fino ad ora.***

4.3 Esempio (soprts.pas)

Il tuo insegnante di Educazione Fisica vorrebbe un programma che calcoli il tempo medio con quale gli studenti corrono i cento metri. Il programma deve richiedere il nome dello studente, la sua classe e i migliori tre tempi con i quali ha corso i cento metri. Il risultato sarà il nome dello studente, la sua classe e la media dei tempi.

4.3.1 Trovare una soluzione

Utilizza il precedente esercizio per aiutarti a scrivere l'analisi del problema. Mstralo al tuo insegnante.

4.3.2 Scrivere il programma

- Decidi di quali variabili ha bisogno il programma.
- Come in precedenza, se non sei sicuro, prima di continuare chiede al tuo insegnante un consiglio.
- Usa la tua analisi per scrivere il programma.
- Salva il programma col nome **sports.pas**.
- Compila ed esegui il programma.

5 Decisioni

I programmi che abbiamo scritto fino ad ora erano composti da sequenza di istruzioni una di seguito all'altra. Certe volte, però, c'è la necessità di scegliere fra due o più opzioni.

Il pascal utilizza le parole chiavi `IF`, `THEN` e `ELSE` per descrivere una scelta.

5.1 Esempio (voting.pas)

Questo programma legge l'età di una persona e scrive un messaggio. Copia il programma seguente e salvalo col nome **voting.pas**.

```
PROGRAM voting; {to decide if you can vote or not}
VAR
  age  : INTEGER;
  name : STRING[20];
BEGIN
  WRITE('enter your name ');
  READLN(name);
  WRITE('enter your age ');
  READLN(age);
  WRITELN('hello ', name);
  IF age < 18
  THEN
    WRITELN('you are too young to vote')
  ELSE
    WRITELN('vote for the purple party!')
END.
```

Esegui questo programma e controlla il funzionamento a seconda dei diversi input. Nota che solo uno dei due messaggi che appaiono nel codice viene visualizzato e ciò dipende dall'età della persona.

5.2 Esempio (passfail.pas)

Ecco un altro esempio. Esso legge il risultato di un test (il numero delle risposte corrette) e scrive se l'utente ha superato o no l'esame. Scrivi il programma e salvalo con il nome **passfail.pas**.

```
PROGRAM exam; {pass or fail an exam}
VAR
  name : STRING[20];
  mark : INTEGER;
BEGIN
  WRITE('enter your name ');
  READLN(name);
  WRITE('enter your mark ');
  READLN(mark);
  IF mark > 50
  THEN
    BEGIN
      WRITELN('hi ', name);
      WRITELN('you scored ', mark);
      WRITELN('well done, you passed')
    END
```

```
  ELSE
    BEGIN
      WRITELN('sorry ', name);
      WRITELN('you scored ', mark);
      WRITELN('you failed, better luck next time')
    END
END.
```

Compila ed esegui il programma. Provalo inserendo dati diversi.

5.3 Esempio (cities1.pas)

Questo è un gioco per due giocatori. Il programma chiede a un giocatore il nome di uno stato e la sua capitale, a questo punto il video viene cancellato e al secondo giocatore dato il nome dello stato inserito in precedenza viene chiesto il nome della sua capitale, se questa coincide con quella che il primo giocatore ha inserito allora l'esito del gioco sarà positivo altrimenti no.

5.3.1 Trovare una soluzione

Il primo passo dovrebbe essere del tipo:

1. ottieni le informazioni dal primo giocatore
2. ottieni la risposta dal secondo giocatore
3. stampa i risultati appropriati

Come al solito raffinando il procedimento otteniamo:

1. ottieni le informazioni dal primo giocatore
   (a) Chiedi al giocatore i il paese
   (b) lesse il paese
   (c) chiedi al giocatore 1 la capitale
   (d) leggi la capitale
   (e) cancella lo schermo
2. ottieni la risposta dal secondo giocatore
   (a) chiedi al giocatore 2 la risposta
   (b) leggi la risposta
3. stampa i risultati appropriati
   (a) se la risposta è corretta
   (b) allora
   (c) stampa il messaggio 'correct'
   (d) altrimenti
   (e) stampa il messaggio 'incorrect'

Riassumendo otteniamo la seguente pseudo codifica:

```
1.1 prompt player 1 for country
1.2 read in country
1.3 prompt player 1 for capital
1.4 read in capital
1.5 clear the screen

2.1 prompt player 2 for answer
2.2 read in answer

3.1 if answer is correct
3.2 then
3.3 print 'correct' message
3.4 else
3.5 print 'incorrect' message
```

5.3.2 Scrivere il programma

Scrivi il seguente programma. L'istruzione `CLRSCR`, quella che cancella lo schermo, potrebbe essere diversa a seconda del compilatore che stai utilizzando, chiedi al tuo insegnante per chiarimenti.

```
PROGRAM quiz; {a quiz game}
VAR
  country : STRING[8];
  capital : STRING[8];
  answer  : STRING[8];
BEGIN
  WRITE('enter name of country ');
  READLN(country);
  WRITE('enter name of capital ');
  READLN(capital);
  CLRSCR;
  WRITE('what is the capital of ', country, '?');
  READLN(answer);
  IF answer = capital
  THEN
    BEGIN
      WRITELN('the capital of ', country, ' is ', capital);
      WRITELN('well done, you got it right')
    END
  ELSE
    BEGIN
      WRITELN('sorry, you got it wrong');
      WRITELN('the correct answer is ', capital)
    END
END.
```

Salva il programma col nome **cities1.pas**, compilalo ed eseguilo.

5.3.3 Testing

Copia e completa la tabella 4 con i valori di input specificati.

Il sommario dei risultati potrebbe essere:

> *Il programma fornisce risultati corretti quando la risposta coincide, anche nella lettere maiuscole e minuscole, con la capitale esatta. In altre*

Paese	Capitale	Risposta	Risultato
Francia	Parigi	Parigi	correct
Francia	Parigi	parigi	
Francia	Parigi	77	
Venezuela	Caracas	Caracas	
USA	Washington	Washington	
Italia	Roma	Roma	

Tabella 4: tabella citta

parole il programma è case-sensitive. *Per esempio, Parigi non è la stessa cosa che parigi. Se l'utente inserisce un numero al posto di qualche lettera la risposta è errata, ma non si verifica nessun* run-time-error.

Se il testo è più lungo di otto caratteri allora la parola viene troncata, vengono considerate le prime otto lettere.

5.3.4 Valutazione

La valutazione del programma potrebbe essere:

Il programma è conforme alla analisi fatta e produce i risultati corrette durante la fase di test. Il programma potrebbe essere migliorato, facendo in modo che non faccia differenza fra lettere maiuscole e minuscole. Il programma potrebbe prevedere anche più di una domanda e dare un voto in base al numero di risposte corrette.

5.4 Esempio (scores.pas)

Un grande tabellone elettronico in uno stadio sportivo mostra i punteggi delle squadre durante la partita. Alla fine del gioco vogliamo mostrare il messaggio contratulations' a coloro che hanno vinto. Viene richiesto un programma che chiede all'utente i nomi delle due squadre e i rispettivi punteggi finali, successivamente verrà mostrato il messaggio appropriato.

5.4.1 Trovare una soluzione

Come al solito fate una analisi del problema. Mostrala al tuo insegnante.

5.4.2 Scrivere il programma

Scrivi il programma Pascal , salvalo il programma col nome **scores.pas**. Compila ed esegui il programma.

5.4.3 Testing

Verifica il funzionamento del programma con dei dati opportuni. Stila una opportuna tabella e commenta.

Attenzione che c'è una evidente mancanza nella descrizione del problema. Hai capito qual è? Se non sei certo chiedi al tuo insegnante.

5.4.4 Valutazione

Descrivi il problema trovato nell'analisi e suggerisci una soluzione. Scrivere una valutazione sul programma e prevedi dei possibili miglioramenti.

5.5 Esempio (taxes.pas)

Chi ha delle entrate che superano un certo valore devono pagare delle tasse. Per fare un esempio, se Tizio guadagna meno di $5000 non paga tasse, altrimenti deve versare al fisco il 25% dell'eccedenza, se guadagna $5500 dovrà al fisco il 25% di $500, cioè 125%.

Se non è ti è chiaro quello che abbiamo espresso, chiedi al tuo insegnante prima di continuare.

Vogliamo scrivere un programma che chieda all'utente di inserire l'imponibile e che visualizzi l'eventuale tassa da versare.

5.5.1 Trovare una soluzione

Come al solito fate una analisi del problema. Mostralo al tuo insegnante

5.5.2 Scrivere il programma

Scrivi il programma Pascal . Salva il programma col nome **taxes.pas**, compila ed esegui.

6 Ripetizioni

Un ciclo è una parte di programma che viene ripetuta un certo numero di volte. Il primo programma Pascal che abbiamo scritto ne aveva uno al suo interno. Un ciclo può essere 'condizionato', cioè ripetuto finché succede qualcosa - una *condizione* appunto - oppure può essere 'incondizionato', cioè viene ripetuto per un certo numero di volte.

Il primo programma che vevamo scritto conteneva un ciclo 'incondizionato'. Il ciclo iniziava con:

```
FOR index := 1 TO 10 DO
```

e la istruzione (o le istruzioni) che venivano ripetute erano racchiuse fra le istruzioni `BEGIN` e `END`:

```
BEGIN
  WRITELN(index)
END
```

In questo caso la variabile chiamata `index` è un *contatore* che controlla il ciclo, esso inizia da 1 e *corre* fino a 10.

6.1 Cicli incondizionati

Riprendiamo l'Esempio 2.4 dove un insegnante voleva un programma che leggesse l'esito di tre test fornisse la media dei risultati. Ora, supponiamo che invece di tre esiti, ce ne siano 10 o 20 oppure 200. I cicli servono proprio a questo, se sappiamo quante volte dobbiamo eseguire una certa operazione.

Ecco come potremmo rifare l'analisi del programma.

1. prendi le informazioni dall'utente
2. calcola la media
3. stampa i risultati

Raffinando otteniamo:

1. prendi le informazioni dall'utente
   (a) assegna al totale il valore zero
   (b) ripeti per ogni studente
   (c) | chiedi all'utente di inserire l'esito
   (d) | leggi il valore
   (e) aggiorna il totale
   (f) fine delle operazioni da ripetere
2. calcola la media
   (a) media = totale / numero di punteggi inseriti
3. stampa i risultati
   (a) stampa il totale dei punteggi inseriti
   (b) stampa la media dei punteggi

Ecco l'analisi completa:

```
1.1 set total to zero
1.2 loop for each student
1.3     prompt user to enter mark
1.4     read in the mark
1.5     update the total
1.6 end loop

2.1 average = total / number of marks

3.1 print out the total marks
3.2 print out the average mark
```

Scrivi il seguente programma e salvalo con il nome **average2.pas**.

```
PROGRAM unconditional;
VAR
  mark    : INTEGER;
  total   : INTEGER;
  average : REAL;
BEGIN
  total := 0;
  FOR counter = 1 TO 10 DO
  BEGIN
    WRITE ('type in the exam mark: ');
    READLN (mark);
    total := total + mark
  END;
  average := total / 10;
  WRITELN('the total is: ');
  WRITELN('the average is: ', average)
END.
```

- Attenzione perché ci sono due errori nel programma che ti impediranno di compilarlo, riesci a trovarli?

- Trovati gli errori di compilazione, ci sono ancora due errori che troverai durante l'esecuzione, li puoi correggere?

Riportiamo una analisi migliore della precedente, che chiede all'utente quanti sono gli studenti.

```
1.1 set total to zero
    1.2 prompt user to enter number of students
    1.3 read in number of students
    1.4 loop for each student
    1.5     prompt user to enter mark
    1.6     read in the mark
    1.7     update the total
    1.8 end loop

    2.1 average = total / number of students

    3.1 print out the total marks
    3.2 print out the average mark
```

Fai i dovuti cambiamenti al programma in modo che rispecchi questa ultima descrizione. **Se hai dei dubbi chiedi al tuo insegnante.**

6.2 Cicli Condizionati

Questi sono cicli dei quali non si conosce a priori il numero di volte che varranno eseguiti. Vi sono due modi di rappresentare questi cicli:

1. vengono ripetute delle operazioni *finché* non fi verifica una certa condizione,
2. *mentre* una certa condizione viene verificata vengono eseguite certe operazioni.

Scrivi il seguente programma:

```
PROGRAM conditional;                {a conditional loop}
VAR
  number : INTEGER;
  guess  : INTEGER;
BEGIN
  number := 5;
  REPEAT
    WRITE('enter a number between 0 and 10: ');
    READLN(guess);
  UNTIL guess = number;
  WRITELN('what took you so long?')
END.
```

Compila ed esegui.
Come puoi vedere, il ciclo viene ripetuto finché non viene inserito il numero 5.

6.3 Esempio (guess1.pas)

Estendiamo il codice precedente in modo da creare un giochino, in cui il giocatore deve indovinare un numero.

6.3.1 Trovare una soluzione

Ecco come si potrebbe affrontare il problema

1. il promo giocatore inserisce un numero
2. il secondo giocatore inserisce dei numeri finché non lo indovina
3. stampa un messaggio

Perfezionando...

1. il primo giocatore inserisce un numero
   (a) chiedi al primo giocatore di inserire un numero
   (b) leggi il numero
   (c) cancella lo schermo
2. il secondo giocatore inserisce dei numeri finché non lo indovina
   (a) ripeti
   (b) | chiedi al secondo giocatore di inserire un numero
   (c) | leggi il numero
   (d) finché il tentativo non è corretto
3. stampa un messaggio

L'analisi completa del programma allora sarà:

```
1.1   prompt player 1 for number
  1.2 read in number
  1.3 clear the screen

  2.1 repeat
  2.2     prompt player 2 for guess
  2.3     read in guess
  2.4 until guess is correct

  3   print message
```

6.3.2 Scrivere il programma

Scrivi il programma Pascal . Salva il programma col nome **guess1.pas** e compila ed esegui.

6.4 Esempio (guess2.pas)

Estendiamo il funzionamento del gioco precedente aggiungendo alcune funzionalità. Sarebbe bello se il programma desse dei suggerimenti al giocatore che deve indovinare il numero, dicendo se il tentativo è più grande o più piccolo del numero da indovinare. Sarebbe altresì utile che il programma contasse il numero di tentativi fatti e li visualizzasse alla fine del gioco. L'analisi iniziale del programma è quella che abbiamo scritto in precedenza, pero la perfezioneremo in modo differente:

6.4.1 Trovare una soluzione

1. il primo giocatore inserisce un numero
   (a) chiedi al primo giocatore di inserire un numero
   (b) leggi il numero
   (c) cancella lo schermo
2. il secondo giocatore inserisce dei numeri finché non lo indovina
   (a) ripeti
   (b) | chiedi al secondo giocatore di inserire un numero
   (c) | leggi il numero
   (d) | visualizza il suggerimento
      i. se il tentativo è maggiore del numero
      ii. allora stampa tentativo troppo grande
      iii. se il tentativo è minore del numero
      iv. allora stampa tentativo troppo piccolo
   (e) | aggiorna il numero di tentativi
   (f) finché il tentativo non è corretto
3. stampa un messaggio
   (a) stampa un messaggio
   (b) stampa il numero di tentativi

Riassumento:

```
1.1 prompt player 1 for number
    1.2 read in number
    1.3 clear the screen

    2.1 repeat
    2.2     prompt player 2 for guess
    2.3     read in guess
    2.4.1       if guess > number
    2.4.2       then
    2.4.3           print "guess is too big"
    2.4.4       if guess < number
    2.4.5       then
    2.4.6           print "guess is too small"
    2.5     update number of guesses
    2.6 until guess is correct

    3.1 print message
    3.2 print number of guesses
```

6.4.2 Scrivere il programma

Ricordate che l'analisi di un problema viene fatta per riuscire a scrivere un programma semplice da scrivere e da leggere. Verificate che la soluzione che abbiamo scritto in precedenza corrisponde al programma che riportiamo di seguito.

```
PROGRAM guessing;                    {a guessing game}

VAR
  number : INTEGER;
  guess  : INTEGER;
  total  : INTEGER;

BEGIN
  WRITE('enter the number to guess ');
  READLN(number);
  CLRSCR;

  REPEAT
    WRITE('enter your guess ');
    READLN(guess);
    IF guess > number
    THEN
      WRITELN('guess is too big');
    IF guess < number
    THEN
      WRITELN('guess is too small');
    total := total + 1;
  UNTIL guess = number;

  WRITELN('well done');
  WRITELN('you took ', total, ' guesses')
END.
```

Salva il programma con il nome **guess2.pas**, compilalo ed eseguilo.

Con questo ultimo esempio abbiamo completato l'introduzione al Pascal . Adesso, se avete seguito attentamente, tutte le lezioni avete i rudimenti per progettare e scrivere dei semplici[10] programmi. Per avere una visione più ampia sulla programmazione strutturata si vedano i miei appunti [1], il classico libro [3], inoltre su internet[11] si trova parecchio materiale.

Nelle sezioni che seguiranno affronteremo degli argomenti più specifici. Continuate la lettura solamente se quello che avete letto vi risulta chiaro, se riuscite autonomamente a scrivere del programmi, anche semplici, in Pascal .

7 Procedure

I programmi che abbiamo scritto fino ad ora erano piuttosto semplici, quando i problemi da risolvere diventano più complicati e complessi, il codice deve essere necessariamente più lungo. Per tale ragione e anche perché è buona norma farlo, molti programmi sono divisi in sezioni più piccole chiamate *procedure*. Una procedura quindi è una porzione di codice che svolge un compito particolare. Questa parte delle dispense tratta in modo piuttosto semplificato l'utilizzo delle procedure nel Pascal . Vi accorgerete che l'argomento è piuttosto vasto, ma non ci interessa entrare troppo nel dettagli, almeno per ora!

[10] Questo dipende anche dalla esperienza che avrete maturato. Se l'argomento vi interessa ci sono parecchi approfondimenti che potrete affrontare.

[11] Comunque sul mio sito personale trovate già del materiale catalogato (*http://www.matematiche.org*)

7.1 Esempio chips2.pas

Per illustrare l'utilizzo delle procedure, useremo l'Esempio 4.2, la sua analisi era:

1. ottenere le informazioni dall'utente
2. calcolare i soldi ricavati
3. stampare i risultati

che portava alla una pseudo codifica:

1. ottenere le informazioni dall'utente
   (a) chiedere all'utente di inserire il gusto delle patatine
   (b) leggere il gusto
   (c) chiedere all'utente di inserire il numero di sacchetti venduti il Venerdì
   (d) leggere il numero di sacchetti venduti il Venerdì
   (e) chiedere all'utente di inserire il numero di sacchetti venduti il Sabato
   (f) leggere il numero di sacchetti venduti il Sabato
   (g) chiedere all'utente di inserire il numero di sacchetti venduti la Domenica
   (h) leggere il numero di sacchetti venduti il Domenica
2. calcolare i soldi ricavati
   (a) calcola il totale di sacchetti venduti
   (b) calcola il totale ricavato
3. stampare i risultati
   (a) scrivere il nome del gusto
   (b) scrivere il numero dei sacchetti venduti
   (c) scrivere il totale ricavato

Si noti che, come in molti altri Esempi, si inizia spesso con uno schema standard, composto da tre passi che successivamente vengono scomposti in istruzioni più dettagliate.

Vorremmo utilizzare questa suddivisione iniziale per suddividere il programma in tre sezioni o procedure, ognuna delle quali svilupperà le istruzioni che abbiamo scritto nella pseudo codifica. Diamo uno sguardo a queste tre sezioni per decidere il loro nome come abbiamo sempre fatto per le variabili.

1. ottenere le informazioni dall'utente — **get_info** (input dati)
2. calcolare i soldi ricavati — **calc_sales** (calcola il ricavato)
3. stampare i risultati — **results** (stampa i risultati)

Il programma è composto da tre procedure: **get_info**, **calc_sales** e **results**. Scrivi il codice seguente:

```
PROGRAM chips; {to calculate chip sales}
VAR
  flavor   : STRING[10];
  friday   : INTEGER;
  saturday : INTEGER;
  sunday   : INTEGER;
  bags     : INTEGER;
  sales    : REAL;

BEGIN {main program starts here}
  get_info;
  calc_sales;
  results
END.
```

Compila il programma. Saranno segnalati un bel po' di errori, ciò non sorprende visto che all'interno del programma non vengono definite per nulla le tre procedure. Dobbiamo descrivere il funzionamento altrimenti non concludiamo nulla. Prendiamo la procedura **get_info** e ricordiamo i dettagli della pseudo codifica che abbiamo precedentemente scritto.

1. chiedere all'utente di inserire il gusto delle patatine
2. leggere il gusto
3. chiedere all'utente di inserire il numero di sacchetti venduti il Venerdì
4. leggere il numero di sacchetti venduti il Venerdì
5. chiedere all'utente di inserire il numero di sacchetti venduti il Sabato
6. leggere il numero di sacchetti venduti il Sabato
7. chiedere all'utente di inserire il numero di sacchetti venduti la Domenica
8. leggere il numero di sacchetti venduti il Domenica

La nostra procedura farà esattamente quello che è descritto nella sua descrizione:

```
PROCEDURE get_info; {to get details from user}
BEGIN
  WRITE('enter flavor of chips ');
  READLN(flavor);
  WRITE('enter number sold on Friday ');
  READLN(friday);
  WRITE('enter number sold on Saturday ');
  READLN(saturday);
  WRITE('enter number sold on Sunday ');
  READLN(sunday)
END;
```

Si noti che abbiamo seguito passo passo la pseudo codifica. Nel programma il codice che descrive la procedura va messo prima dell'inizio del programma principale, sicché abbiamo:

```
PROGRAM chips; {to calculate chip sales}
VAR
  flavor  : STRING[10];
  friday   : INTEGER;
  saturday : INTEGER;
  sunday   : INTEGER;
  bags     : INTEGER;
  sales    : REAL;

PROCEDURE get_info; {to get details from user}
BEGIN
  WRITE('enter flavor of chips ');
  READLN(flavor);
  WRITE('enter number sold on Friday ');
  READLN(friday);
  WRITE('enter number sold on Saturday ');
  READLN(saturday);
  WRITE('enter number sold on Sunday ');
  READLN(sunday)
END;

BEGIN {main program starts here}
  get_info;
  calc_sales;
  results
END.
```

Modifica il codice che hai scritto in precedenza e salvalo con il nome **chips2.pas**. Compilando il programma noterai che ci sono ancora degli errori, ma abbiamo capito che ci sono perché mancano le descrizioni delle altre due procedure. Riportiamo anche la seconda procedura, mentre la terza la devi scrivere tu!

```
PROGRAM chips; {to calculate chip sales}
VAR
  flavor   : STRING[10];
  friday   : INTEGER;
  saturday : INTEGER;
  sunday   : INTEGER;
  bags     : INTEGER;
  sales    : REAL;

PROCEDURE get_info; {to get details from user}
BEGIN
  WRITE('enter flavor of chips ');
  READLN(flavor);
  WRITE('enter number sold on Friday ');
  READLN(friday);
  WRITE('enter number sold on Saturday ');
  READLN(saturday);
  WRITE('enter number sold on Sunday ');
  READLN(sunday)
END;

PROCEDURE calc_sales; {to calculate the sales}
```

```
BEGIN
  bags  := friday + saturday + sunday;
  sales := 0.65 * bags
END;

{ATTENZIONE MANCA LA PROCEDURA RESULTS,
 QUESTA AL DEVI SCRIVERE TU!!!}

BEGIN {main program starts here}
  get_info;
  calc_sales;
  results
END.
```

Salva il programma completo, compila ed esegui. Mostra i risultati al tuo insegnante[12].

7.2 Esempio (cities2.pas)

Riprendiamo l'Esercizio 5.3, utilizziamo la pseudo codifica ed aggiungiamo i nomi delle procedure:

1. ottieni le informazioni dal primo giocatore **get_info** (input informazioni)
2. ottieni la risposta dal secondo giocatore **get_answer** (input risposte)
3. stampo i risultati appropriati **message** (messaggi finali)

Così il programma avrà questa codifica:

```
PROGRAM quiz; {a quiz game}
VAR
  country : STRING[8];
  capital : STRING[8];
  answer  : STRING[8];

BEGIN {main program starts here}
  get_info;
  get_answer;
  message
END.
```

Come in precedenza, se provi a compilare il programma otterrai degli errori perché le tre procedure che abbiamo inserito non hanno nessuna codifica. Trascriviamo le prime due e lasciamo il compito a te di scrivere la terza.

```
PROGRAM quiz; {a quiz game}
VAR
  country : STRING[8];
  capital : STRING[8];
  answer  : STRING[8];

PROCEDURE get_info; {to get information from player 1}
BEGIN
```

[12] Congratulazioni hai scritto la tua prima procedura!

```
  WRITE('enter name of country ');
  READLN(country);
  WRITE('enter name of capital ');
  READLN(capital);
  CLRSCR
END;

PROCEDURE get_answer; {to get answer from player 2}
BEGIN
  WRITE('what is the capital of ', country, '? ');
  READLN(answer)
END;

{ATTENZIONE MANCA LA PROCEDURA MESSAGE!!}
{the other procedure goes in here}

BEGIN {main program starts here}
  get_info;
  get_answer;
  message
END.
```

Completa il codice, salvalo con il nome **cities2.pas**. Di seguito riportiamo la procedura message, ma attenzione che le linee non sono in ordine, come esercizio prova a rimaneggiare la sequenza delle righe in modo da ottenere la procedura corretta.

```
PROCEDURE message;          {to print an appropriate message}
ELSE
WRITELN('wrong - the answer is ', capital)
IF answer = capital
WRITELN('correct - well done!')
THEN
END;
BEGIN
```

7.3 Esempio (invoice.pas)

Vediamo adesso un Esempio completo per verificare se abbiamo compreso tutto e bene.

Una ditta prende ordini dai suoi clienti e applica uno sconto del 10% sugli ordini che superano i $200. Sul totale scontato viene appicata una tassa pari al 5%. Un tipico conto potrebbe essere:

Se il valore è inferiore a $200 allora lo sconto deve essere nullo, ma la tassa deve essere aggiunta lo stesso. Se non è chiaro il problema, chiedi al tuo insegnante di rileggerlo con te.

value	=	220.00
discount	=	22,00
subtotal	=	198.00
sales tax	=	9,90
total	=	207.00

Tabella 5: tabella invoice

7.3.1 Trovare una soluzione

1. ottieni i dati dell'ordine
2. calcola il totale
3. stampa la ricevuta

Noterai che anche in questo caso il programma è composto da tre parti, che possono essere le procedure che cerchi.

Un ulteriore raffinamento porta alla seguente pseudo codifica:

1. ottieni i dati dell'ordine
   (a) Chiedi all'utente il valore dell'ordine
   (b) leggi il valore
2. calcola il totale
   (a) Se il valore è maggiore di $200
   (b) allora
   (c) | lo sconto è il 10% del valore
   (d) altrimenti
   (e) | lo sconto è nullo
   (f) il sottototale è il valore meno lo sconto
   (g) il totale è il sottototale più la tassa
3. stampa la ricevuta
   (a) stampa il valore
   (b) stampa lo sconto
   (c) stampa il sottototale
   (d) stampa la tassa
   (e) stampa il totale

Usiamo **get_value**, **calc_total** e **invoice** come nomi delle tre procedure. Il programma senza le procedure sarà:

```
PROGRAM orders; {to print out an invoice}
VAR
  value     : REAL;
  discount  : REAL;
  subtotal  : REAL;
  sales_tax : REAL;
  total     : REAL;

{procedure get_value goes here}

{procedure calc_total goes here}

{procedure invoice goes here}

BEGIN {main program starts here}
  get_value;
  calc_total;
  invoice
END.
```

Salva il programma col come **invoice.pas**. Aggiungi le tre procedure facendo attenzione a non commettere errori di battitura. Salva e compila il programma.

8 Conclusioni

Spero che tu abbia trovato questa dispensa chiara e utile. Se è così allora credo che tu abbia fatto bene e io pure. Non pensare che sia finita qua, vi sono parecchie cose da approfondire: comincia dalla bibliografia e continua su internet e ...

Certo è che abbiamo rotto il ghiaccio e questa è solitamente la parte più complicata da fare.

Una cosa non l'abbiamo detta: *quale compilatore devi usare?* Solitamente nell'ambito scolastico viene utilizzato Turbo Pascal della Borland (vedi [6], ma io ho apprezzato molto FreePascal (vedi [7]. Chiedi al tuo insegnante o cerca su internet ai riferimenti che troverai nella bibliografia per averli.

Riferimenti bibliografici

[1] Enrico Centenaro. *Appunti di Programmazione Strutturata*. Scrittura privata, 1997-2000.

[2] Vari. *Appunti sul Pascal*. Dispense, 1987.

[3] Niklaus Wirth. *Algoritmi + Strutture Dati = Programmi*. Casa Editrice.... anno....

[4] D. E. Knuth *The TeXbook*. Addison-Wesley. 1984, ISBN 0-201-13447-0.

[5] Leslie Lamport, *LaTeX: A Document Preparation System*. Addison-Wesley. 1994.

[6] Borland, *http://www.borland.com*.

[7] Open Source, *http://www.freepascal.org*.