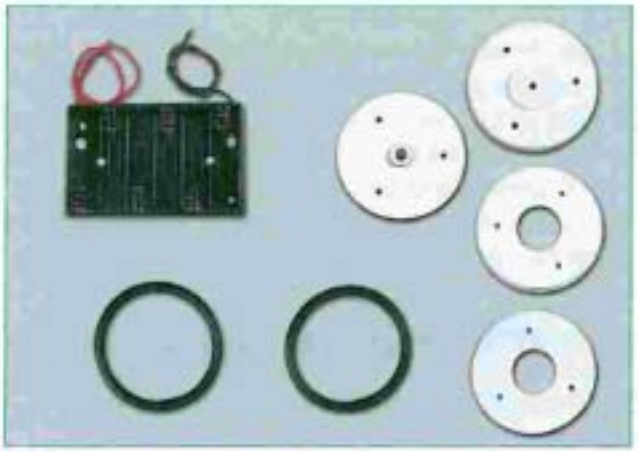
norme di buona programmazione

Pseudo codice e non

Come preannunciato, proseguono gli allegati che costituiranno il nuovo set di locomozione e alimentazione del robot. Rimandandone la trattazione al prossimo fascicolo, torniamo invece al 'problema labirinto'. Una volta ottenuto un algoritmo ben strutturato (tramite il diagramma di flusso) siamo quasi pronti per tradurlo in codice di programmazione. Prima, però, ci soffermiamo su una fase intermedia: **trasformeremo cioè l'algoritmo nel cosiddetto 'pseudo codice', a metà strada tra il linguaggio naturale (nel nostro caso l'italiano) e il linguaggio di programmazione che utilizzeremo (cioè il PBASIC)**, come illustrato qui sotto.

A destra. Gli allegati a questo fascicolo sono i componenti delle ruote e il nuovo portabatterie.

Sotto. Parte del diagramma di pag. 193, modificato (freccia rossa). Numerato in base allo pseudo codice (l'elenco al centro), è poi tradotto in codice (a destra).



A differenza dell'algoritmo che, essendo 'generale', può essere implementato tramite diversi linguaggi di programmazione, il codice è 'specifico' di ogni linguaggio: per questo è utile passare per la stesura di uno pseudo codice che, iniziando a specificare, definire e numerare i passaggi visualizzati nel diagramma di flusso, faciliti l'identificazione dei legami tra tali passaggi e il linguaggio che si utilizzerà per il codice finale.

PSEUDO CODICE	CODICE
1. Avanza	- ETICHETTA1: Avanza
2. Esegui Rileva_IRFrontali	- Rileva_IRFrontali
3. Se *statoIR*=0 vai a 1.	- IF *statoIR*=0 THEN ETICHETTA1
4. Ruota_90	- Ruota_90
5. Esegui RilevaIR	- RilevaIR
6. Testa *statoIR*:	- BRANCH *statoIR*, [ETICHETTA2,ETICHETTA3, ETICHETTA2,ETICHETTA4]
6.1. se *statoIR*=0 vai a 7.	
6.2. se *statoIR*=1 vai a 9.	
6.3. se *statoIR*=2 vai a 7.	
6.4. se *statoIR*=3 vai a 11.	
7. 'casella 2 Dir OPPURE casella 3 Dir; direzione ammissibile = dir_pred' inverti direzione predefinita	- ETICHETTA2: *dir_pred=dir_pred*+1
8. Vai a 1.	- GOTO ETICHETTA1
9. 'casella 2 Dir; direzione ammissibile ≠ dir_pred' inverti direzione di marcia	- ETICHETTA3: *direzione=direzione*+1
10. Vai a 1.	- GOTO ETICHETTA1
11. 'casella 1 Dir' attiva lo stato *Back* (=1)	- ETICHETTA4: *Back*=1
12. Ruota_90	- Ruota_90
13. Vai a 1.	- GOTO ETICHETTA1

Vediamo dunque quali sono i passaggi necessari per tradurre un diagramma di flusso in pseudo codice. Per semplicità, analizziamo solo la porzione dell'algoritmo riportata a pag. 195: noterai che si tratta di uno dei due macro-stati di cui si è detto a pag. 194, cioè quello in cui il robot avanza senza occuparsi delle strade laterali se non in presenza di un ostacolo. Ci si limita all'impostazione della variabile *Back*, quindi, senza però testarla, motivo per cui non si passa mai al secondo macro-stato. Ciò che sappiamo già è di avere a disposizione: **tre variabili di stato legate al moto** (*Back*, *direzione* e *dir_pred*); **due routine di moto** (vale a dire Avanza e Ruota_90, dipendenti rispettivamente da *direzione* e *dir_pred*); **una variabile di stato da 4 bit relativa ai quattro sensori IR** (*statoIR*) e **due routine che**, implementate in funzione della variabile *direzione*, **interagiscono con i sensori IR settando il valore di *statoIR*** (Rileva_IRFrontali e RilevaIR). In più, Rileva_IRFrontali setta *statoIR*=0 in assenza di ostacoli e *statoIR*=1 in presenza di un ostacolo. RilevaIR, invece, setta *statoIR*=0 in assenza di ostacoli; *statoIR*=1 con ostacolo rilevato frontalmente; *statoIR*=2 con ostacolo rilevato posteriormente; e *statoIR*=3 con ostacolo rilevato da entrambe le coppie di IR.

Le norme di programmazione

Arrivati (o tornati) al codice, siamo ora in grado di scrivere un programma sintatticamente corretto. Perché, dunque, non scriverlo anche bene, per ciò che riguarda la forma e la struttura? Di fatto, è già capitato che parlassimo di 'norme di buona scrittura'; ora vale la pena riprenderle e trattarle più approfonditamente.

Alla 'buona programmazione' fanno capo regole e norme legate alla scrittura e alla strutturazione del codice, che permettono di sviluppare programmi (preventivamente corretti) dotati di particolari proprietà formali. A differenza delle regole sintattiche, imposte dal particolare linguaggio usato (per noi il PBASIC) e obbligatorie perché un programma sia corretto e funzionante, le regole di buona programmazione sono indicazioni opzionali, consigli, che permettono di rendere più funzionale e ottimale il codice scritto. Come sai, una proprietà molto importante per un programma è la sua modificabilità, ossia la sua 'disponibilità' a subire modifiche anche importanti all'interno del codice, senza per questo doverne riscrivere intere porzioni o, nel peggiore dei casi, riprogettare l'intera struttura. Vedremo tra poco che, spesso, l'utilizzo oculato di costanti, variabili e subroutine consente di ottimizzare il codice in questo senso. Un'altra proprietà molto importante è la leggibilità del codice: una caratteristica che interessa sia il programmatore che scrive sia un eventuale lettore che potrà anche voler apportare modifiche. La leggibilità riguarda soprattutto la forma 'estetica' del codice, intesa proprio come ordine e 'pulizia' di scrittura: così come la calligrafia (ossia la bella scrittura) favorisce la comprensione di un testo manoscritto, così la stesura 'ragionata' e chiara del codice di un programma ne agevola sia la lettura sia la comprensione. Certo la leggibilità del codice non è una proprietà necessaria: un programma scritto in modo disordinato e poco 'amichevole' alla vista (ad esempio, privo di spazi di tabulazione e con tutto il codice scritto su una stessa riga), se è corretto sintatticamente, funziona allo stesso modo e offre le stesse prestazioni di un programma più leggibile. Tuttavia, se a distanza di tempo il programma venisse ripreso da un'altra persona o, come capita spesso, dal programmatore stesso, la differenza si farebbe sentire, eccome. Vediamo ora come tradurre in fatti quanto detto.

La struttura del codice - Un'importante norma di buona scrittura riguarda la distinzione della struttura e la separazione delle varie parti del codice: in generale, è opportuno separare, sempre e anche visivamente, la dichiarazione di variabili e quella di costanti, le fasi di inizializzazione, il corpo centrale del programma e le subroutine. Nei programmi proposti finora, ad esempio, si è adottata spesso questa norma, separando le porzioni di codice con commenti e segni grafici che ne delineassero la successione. Come risultato si ha il riconoscimento immediato della struttura del programma, con conseguente facilitazione della sua analisi. Il listato a sinistra riassume quanto detto e può fungere da modello per ogni programma che andrai a costruire. Trascrivilo nell'area di editing e salva il file in una directory diversa da quella in cui hai memorizzato i tuoi programmi; in futuro potrai caricare questo file, salvarlo (scegliendo 'Save As' dal menu File) con il nome del programma che vorrai costruire (in questo modo duplicherai il modello, senza cancellarlo) e completarlo, sostituendo le scritte racchiuse tra i segni < > con il codice PBASIC. Manterrai così, fin dall'inizio, una buona struttura.

```
'{$STAMP BS2}
'**************************************************************
'
'        <DESCRIZIONE DEL PROGRAMMA>
'
'**************************************************************
'-------------------- Dichiarazione Variabili --------------------
        <DICHIARA QUI LE VARIABILI>
'-------------------- Dichiarazione Costanti ---------------------
        <DICHIARA QUI LE COSTANTI>
'-------------------- Inizializzazione ---------------------------
        <FASE DI INIZIALIZZAZIONE DELLE VARIABILI/PORTE>
'-------------------- Programma Principale -----------------------
        <PROGRAMMA PRINCIPALE>
'-------------------- Routine ------------------------------------
        <ROUTINE UTILIZZATE NEL PROGRAMMA PRINCIPALE>
'-------------------- Subroutine ---------------------------------
        <DICHIARA QUI LE SUBROUTINE UTILIZZATE NEL PROGRAMMA>
```

Nome routine	Dipendenze
Avanza	direzione
Ruota_90	dir_pred
Rileva_IRFrontali	direzione
RilevaIR	direzione

Sopra. Aggiornamento della tabella di pag. 187, utile per controllare i dati certi.

Partendo da questi dati, possiamo tradurre l'algoritmo in pseudo codice, come mostrato a pag. 195 (dal diagramma, a sinistra, all'elenco numerato, al centro), quindi, in codice vero e proprio (la lista non numerata, a destra). Per farlo, considera che:

(1) Al posto della parola '**se**' dello pseudo codice, si sostituisce (in base al numero di alternative possibili) **if... then** oppure **branch**.
(2) A ogni occorrenza della parola '**vai**' dello pseudo codice, si sostituisce l'istruzione **goto**.
(3) Ai **numeri** cui rimandano i salti (i 'vai', trasformati in goto) si sostituisce un'**etichetta**.
(4) Tutti i **commenti** dello pseudo codice (racchiusi tra apici) si possono mantenere come tali (solo, **preceduti da un unico apice**).

Questo codice è di fatto il cuore della navigazione del robot nel labirinto, rispetto a uno dei due macro-stati. Partendo da qui non dovrebbe esserti difficile ricavare il codice relativo all'intero algoritmo e, meglio ancora, allenarti a modificarne a tuo piacere la struttura. Quella proposta, infatti, non è che una delle possibili soluzioni, non certo l'unica né per forza la migliore. Questi sono i 'ferri del mestiere'. Ma il bello è usarli.

I commenti - Tra le componenti basilari di un codice ben scritto ci sono i commenti, in quanto contribuiscono in maniera determinante alla comprensione del testo, anche a distanza di molto tempo dalla sua scrittura. Di fatto, permettono di integrare il linguaggio di programmazione con il linguaggio naturale di più immediata lettura. Il ruolo di un commento, infatti, è sostanzialmente quello di un 'promemoria'. Se volessi distribuire un tuo programma a un amico, ad esempio, un codice ben commentato sarà senz'altro gradito. Inoltre, tornerà utile anche a te, qualora a distanza di tempo volessi riutilizzare o modificare quel programma. Come ricorderai, i commenti inseriti nel codice (preceduti da un apice ') vengono del tutto ignorati quando il programma viene caricato nella memoria del robot: che tu ne abbia scritto cento o nessuno, cioè, non avrai comunque intaccato in alcun modo le prestazioni del robot. Meglio allora inserire quanti più commenti possibile (cioè tutti quelli utili). In generale, è buona prassi avere all'inizio del programma una breve descrizione di quanto seguirà, riassumendone le finalità e le risorse che verranno utilizzate. Considera ad esempio il frammento di codice a destra: il lungo commento iniziale riporta lo scopo del programma e alcune informazioni sui componenti utilizzati. Per rendere più ordinato il codice, puoi utilizzare alcuni segni grafici (come il carattere * dell'esempio) che separino commenti successivi: alla descrizione iniziale, può seguire un promemoria con l'elenco delle porte logiche disponibili sul microcontrollore e il loro utilizzo nel programma. Per avere sotto controllo la configurazione delle porte richieste dal programma, soprattutto in casi in cui, come qui, sono utilizzati molti componenti, è consigliabile inserire una legenda come quella dell'esempio. Supponiamo, infatti, di voler integrare una nuova funzionalità all'interno di questo programma, mediante un circuito costruito sulla breadboard (cosicché, ad esempio, un led lampeggi durante i movimenti della pinza). Grazie alla lista delle porte inserita nel commento iniziale sapremo subito, e senza sforzo, quali porte sono ancora libere (P9 o P10). Un'altra serie di informazioni che è bene avere a disposizione è, ad esempio, la legenda dei movimenti dei motori a spazzola, per non doverli ricordare a memoria o procedere per tentativi. Un semplice schema come quello proposto permette l'immediata associazione di ogni movimento alla rispettiva porta. Nel frammento di codice, infine, c'è un commento relativo all'uso del telecomando, con le associazioni codice-movimento. A prescindere dall'esempio, comunque, è un'ottima abitudine riportare all'inizio di un programma tutte quelle informazioni che potranno risparmiarci fatica inutile. È quindi consigliabile commentare variabili e costanti all'atto della dichiarazione, così da annotarne l'uso che ne verrà fatto;

così come è buona norma commentare i vari passaggi all'interno del codice, aggiungendo una breve annotazione per le istruzioni più complesse o per i blocchi di istruzioni logicamente collegati tra loro. Allo stesso modo è opportuno inserire brevi commenti all'inizio di ogni routine o subroutine, per ricordarne la funzione.

Le costanti - Al pari dei commenti, le costanti non occupano spazio in memoria: prima di caricare il programma nella EEPROM, infatti, l'editor provvede a sostituire tutti i nomi delle costanti con il relativo valore dichiarato, senza perciò intaccare preziose risorse di memoria. L'utilizzo delle costanti è consigliabile per almeno due motivi. In primo luogo, una costante è costituita da un nome, tramite il quale è possibile, attraverso il linguaggio naturale, associare un particolare significato semantico al valore numerico cui la costante si riferisce. Consideriamo ad esempio il frammento di codice in basso, relativo al programma di prima. Le tre dichiarazioni ALTO con 1, BASSO con 0 e PORTA_RC con 11 associano i rispettivi valori numerici a costanti dal nome significativo (ALTO e BASSO richiamano direttamente i due valori logici possibili; PORTA_RC, invece, ricorda che si tratta di una porta e ne richiama la funzione). Il nome in linguaggio naturale, chiarisce perciò il significato, immediatamente, con ovvi vantaggi nella lettura e comprensione del codice. Per questo è bene scegliere con cura il nome delle costanti in modo che possano ricondurre facilmente un valore numerico al suo significato. In secondo luogo, le costanti permettono di ottenere una maggiore modificabilità del codice. Nell'esempio precedente, la porta 11 è collegata al circuito RC costruito sulla breadboard: se per qualche motivo si volesse collegare il circuito a un'altra porta e non si fosse dichiarata la costante PORTA_RC, occorrerebbe cercare tutte le istruzioni che fanno riferimento alla vecchia porta e sostituirne il valore con quello nuovo. Utilizzando la costante (PORTA_RC, invece del valore numerico 11), basterà invece sostituirne il valore una sola volta, nella dichiarazione, aggiornando così l'intero programma in un colpo solo. Infine, un'altra buona norma è distinguere a livello grafico le costanti dalle variabili, scegliendo di usare, ad esempio (ma non esistono regole assolute), caratteri maiuscoli per le costanti e minuscoli per le variabili. In questo modo, la diversità dei due elementi salterà all'occhio, permettendo di riconoscerli a prima vista.

Le variabili - Alcune delle considerazioni fatte per le costanti valgono ovviamente anche per le variabili. Utilizzare nomi significativi, appartenenti al linguaggio naturale, è comunque e sempre una buona soluzione. Per favorire una rapida scrittura del codice, però, è preferibile associare le variabili (ma anche le costanti) a nomi non troppo lunghi: occorre cioè trovare un compromesso tra espressività e concisione di un nome. Spesso i termini inglesi consentono una 'espressività concisa', motivo per cui sono spesso prediletti. Al contrario delle costanti, però, le variabili occupano spazio in memoria RAM e, dunque, è necessario utilizzarle al meglio, soprattutto quando si intende sviluppare programmi complessi che potrebbero generare problemi nella gestione della RAM. All'atto della dichiarazione, dunque, è opportuno calcolare e dimensionare correttamente il tipo (bit, byte, nibble o word) di variabili necessarie, cercando di prevedere quale sarà il valore massimo (e quindi la dimensione massima) che, nel corso dell'intero programma, si dovrà memorizzare all'interno di ogni variabile. In questo modo si evitano inutili sprechi di memoria che potrebbero anche compromettere la realizzabilità del programma stesso. In base a tali considerazioni, inoltre, è consigliabile riutilizzare il più possibile le stesse variabili: quella usata come contatore all'interno di un ciclo for, per esempio, può essere riutilizzata (tenendo ben presente le caratteristiche del programma) in più cicli diversi. Anche nella dichiarazione delle variabili, infine, è consigliabile mantenere un certo ordine: le si può ad esempio raggruppare in blocchi, in base al tipo o alla funzione.

Le subroutine - Come ricorderai, le subroutine favoriscono la modificabilità del codice e la sua organizzazione concettuale. È buona norma raggruppare tutte le subroutine di un programma, separandole dalle routine principali. Occorre però fare molta attenzione a dove le si posiziona: vanno infatti collocate in un punto del codice (tipicamente in fondo) non direttamente raggiungibile dalla normale esecuzione del programma. Infine, l'utilizzo di subroutine permette di risparmiare spazio nella memoria EEPROM.

```
'------------------------ Dichiarazione Variabili ------------------------

BaffoDx          var     in8        ' Alias per la porta del baffo Destro
BaffoSx          var     in6        ' Alias per la porta del baffo Sinistro

FC_aperto        var     bit        ' Variabile di stato per Fine corsa apertura pinza
FC_chiuso        var     bit        ' Variabile di stato per Fine corsa chiusura pinza
FC_alzato        var     bit        ' Variabile di stato per Fine corsa salita carrello
FC_abbassato     var     bit        ' Variabile di stato per Fine corsa discesa carrello

contatore        var     nib        ' Variabile ausiliaria

codicePulsante   var     byte       ' Contiene il valore del codice pulsante premuto
loop             var     byte       ' Variabile contatore per servomotori

movDx            var     word       ' Variabili per il movimento dei servo
movSx            var     word

impulso          var     word       ' Variabili per la decodifica del segnale IR del telecomando
inputIR0         var     word
inputIR1         var     word
inputIR2         var     word
inputIR3         var     word
inputIR4         var     word

misuraRC         var     word       ' Variabile per la misura della costante di tempo RC
sogliaBracci     var     word       ' Variabile di soglia Fine corsa per bracci
sogliaCarrello   var     word       ' Variabile di soglia Fine corsa per carrello

'------------------------ Dichiarazione Costanti ------------------------

M1A              con     14         ' Costanti relative alle porte dei motori DC
M1B              con     15         ' M1 = Motore DC su I2
M2A              con     3          ' M2 = Motore DC su I3
M2B              con     5

IR_DX            con     6          ' Porta IR detect per segnale da telecomando
ALTO             con     1          ' Valore logico alto
BASSO            con     0          ' Valore logico basso
PORTA_RC         con     11         ' Porta a cui è connesso il circuito RC per Fine corsa

SERVO_DX         con     13         ' Porte relative ai servomotori
SERVO_SX         con     12

ANTIORARIO_DX    con     1088       ' Valori per il movimento delle ruote: sostituisci a questi valo
ANTIORARIO_SX    con     1088       ' i valori tipici per i tuoi servo
ORARIO_DX        con     520
ORARIO_SX        con     520

'------------------------ Programma Principale ------------------------

low   M1A   low   M1B
low   M2A   low   M2B                 ' Tutti i motori all'inizio sono fermi

low SERVO_DX       low SERVO_SX

goto valori_Fine_corsa                ' Salta alla routine per campionare i valori di
                                      ' riferimento di RC per i Fine corsa

main:

      gosub decodifica                ' salta alla subroutine di decodifica del segnale

      branch codicePulsante, [AvantiSx, Avanti, AvantiDx, IndietroSx, Indietro, IndietroDx, Apri, Alza, Ch

goto main

'--------------- Routine di assegnazione con motori Servo ---------------

AvantiSx:
```