

Aprendendo a programar com o BASIC Step M8

Neste artigo nós vamos falar sobre as estruturas da linguagem BASIC.

Antigamente, o único modo de alterar a seqüência de execução era com um salto, utilizando o comando GOTO. Os programas ainda podem ser escritos utilizando este comando e funcionam mas quando os programas ficam maiores, maior que uma página, começa a ficar complicado entender o que o programa está fazendo. Geralmente, para escrever programas grandes, é melhor evitar o uso do comando GOTO.

O compilador do BASIC Step M8 possui diversos comandos que pode ser usados ao invés do GOTO.

IF ... THEN ... ELSE ... END IF

Nos programas anteriores nós já usamos a estrutura do comando IF, mas será bom entender melhor como ele funciona.

O comando inicia com a palavra IF. A palavra IF deve ser seguida de uma comparação de algum tipo. Por exemplo, se quiser testar se a variável é 0, faremos

```
IF x = 0 THEN
```

Veja que um espaço é necessário em ambos os lados do sinal de igual. Você deve saber também que a palavra THEN é opcional.

Se o teste for verdadeiro, o comando imediatamente seguinte ao IF será executado. Se o teste for falso, a execução do programa irá continuar com o comando seguinte ao ELSE (se existir) ou o END IF. Seria mais ou menos assim:

```
IF x = 0 THEN
    ` os comandos serão executados aqui se X for 0
ELSE
    ` os comandos serão executados aqui se x não for 0
END IF
```

Existem 4 tipos de comparação permitidos:

=	igual
	não igual (diferente)
>	maior que
<	menor que

O compilador não aceita mais de um teste por comando. Você não pode fazer testes múltiplos utilizando operadores lógicos. Também não pode colocar expressões no teste,

apenas variáveis e constantes.

É altamente recomendável deslocar as linhas de código dentro da estrutura IF. Você deve deslocar também as outras estruturas IF dentro do IF. Por exemplo se você quiser setar o bit B,2 somente se x for 0 e y% for maior que z% você deve escrever.

```
IF x = 0 THEN
  IF y% > z% THEN
    SETBIT B,2
  ELSE
    CLRBIT B,2
  END IF
ELSE
  CLRBIT B,2
END IF
```

Quando você desloca as linhas dentro do IF fica realmente muito fácil ver quem controla o que quando você for ler o código depois de algum tempo.

GOSUB and RETURN

É freqüente encontrar algumas partes do código que devem ser executadas em diversos lugares do programa. Isto pode ser feito facilmente com o comando GOSUB. A idéia desta estrutura é facilmente entendida com um exemplo. Nós queremos ligar o bit B,2 desligar o bit B,1 e mudar o bit B,3. Nós iremos escrever uma sub-rotina (chamada DOIT2B). Esta sub-rotina irá executar tudo o que queremos. Ela irá começar com o label DOIT2B: e terminar com o comando RETURN.

Esta rotina pode ser colocada quase em qualquer lugar do programa, mas geralmente é mais conveniente colocar no final do programa, depois do código principal. Agora suponha que queiramos executar a rotina se a variável x for 1, 3 ou 6. Nós poderíamos escrever:

```
IF x = 1 THEN
  GOSUB DOIT2B
END IF
IF x = 3 THEN
  GOSUB DOIT2B
END IF
IF x = 6 THEN
  GOSUB DOIT2B
END IF
END

DOIT2B:   SETBIT B,2
          CLRBIT B,1
          TOGGLE B,3

RETURN
```

Você pode ver que o uso do comando GOSUB reduz o tamanho de nosso programa pois não precisamos repetir a rotina diversas vezes.

O Comando CASE

O comando CASE é um modo excelente de tomar decisões baseadas em um valor. No exemplo a seguir nós podemos fazer exatamente o que fizemos no exemplo anterior usando diversos comandos IF. Foi adicionada uma nova rotina para desfazer o que DOIT2B faz se nenhuma das condições for satisfeita.

```
BEGIN CASE x
  CASE 1, 3
    GOSUB DOIT2B
  CASE 6
    GOSUB DOIT2B
  CASE ELSE
    GOSUB UNDOB
END CASE

DOIT2B:   SETBIT B,2
          CLRBIT B,1
          TOGGLE B,3
RETURN

UNDOB:   CLRBIT B,2
          SETBIT B,1
          TOGGLE B,3
RETURN
```

Veja que o CASE 6 e as linhas seguintes não serão executadas se você acrescentar um “,6” no final do primeiro CASE. Este código foi escrito apenas para demonstrar que você pode usar múltiplos CASE dentro de uma estrutura BEGIN CASE.

PROCEDURES

O único problema com os comandos GOSUB/RETURN é que a sub-rotina pode alterar os valores usados no programa principal. Isto nos leva a um problema de lembrar qual rotina altera qual variável. Isto pode ser contornado usando procedures.

Numa procedure, os valores de determinadas variáveis são passadas pelo programa principal. A procedure só pode modificar os valores destas variáveis. Qualquer outra variável utilizada pela procedure é local a ela e não é vista pelo programa principal.

Tudo isto é feito pelo comando CALL para chamar a procedure e os comandos SUB e END SUB para identificar e procedure.

No programa a seguir iremos calcular:

$$Z\% = Y\% / X\%$$

E também

$$P\% = Q\% / R\%$$

E a seguir vamos converter P% de binário para o formato BCD.

```

CALL IDIVI (y%, x%) (z%)
CALL IDIVI (q%, r%) (p%)
CALL INT2BCD (p%) (p%)

'====INTEIRO PARA BCD PROCEDURE=====
'= CONVERTE a% para n% no formato BCD =
'=====
SUB INT2BCD (a%) (n%)
  n% = 0
  WHILE a% > 999
    a% = a% - 1000
    n% = n% + &H1000
  WEND
  WHILE a% > 99
    a% = a% - 100
    n% = n% + &H100
  WEND
  WHILE a% > 9
    a% = a% - 10
    n% = n% + &H10
  WEND
  n% = n% + a%
END SUB
'====FIM DE INT2BCD=====

'====DIVISÃO DE INTEIRO POR INTEIRO=====
'= Faz uma divisão inteira sem sinal =

```

Conhecendo o BASIC Step M8 – parte 5

```
'= answer% = top% \ bot%
'= NOTA: MSB de top% deve ser zero =
'=====
SUB IDIVI(top%,bot%)(answer%)
  IF bot% = 0
    answer% = &HFFFF
    EXIT SUB
  END IF
  answer% = 0
  ctr~ = 0
  WHILE bot% < top%
    IF bot% < &H8000
      INCR ctr~
      SHIFT bot%,1,LEFT
    ELSE
      EXIT WHILE
    END IF
  WEND
  INCR ctr~
  WHILE ctr~ | 0
    SHIFT answer%,1,LEFT
    IF top% < bot%
      ELSE
        top% = top% - bot%
        INCR answer%
      END IF
    SHIFT bot%,1,RIGHT
    DECR ctr~
  WEND
END SUB
'====FIM DE IDIVI=====
```