LOGO • MAT

Programa para resolver o problema da travessia do deserto

Segue-se a listagem do programa anunciado no artigo «Travessia Discreta do Deserto» (ver noutro local desta revista):

```
to lista.6 :m
op lista1.6 1 :m
end

to lista1.6 :p :m
if equalp :p :m+1 [op [ ] ]
op fput 6 lista1.6 :p+1 :m
end

to soma :lista
if emptyp :lista [op 0]
op sum first :lista soma bf :lista
end

to dia.deserto :lista
if emptyp :lista [op [ ] ]
if equalp 0 first :lista [op fput 0 dia.deserto bf :lista]
op fput difference first :lista 1 dia.deserto bf :lista
end

to ajustar :lista
op ajustar1 soma :lista :lista
end

to ajustar1 :n :lista
if emptyp :lista [op [ ] ]
if equalp 0 :n [op fput 0 ajustar1 :n bf :lista]
if or :n<6 :n=6 [op fput :n ajustar1 0 bf :lista]
op fput 6 ajustar1 :n–6 bf :lista
end

to lista.final? :lista
if equalp count :lista 1 [op equalp 6 first :lista]
op and equalp 6 first :lista or equalp 1 first bf :lista
equalp 0 first bf :lista
end

to distancia :n.homens
pr distancia 1 0  lista.6 :n.homens
end

to distancia1 :n :lista
if lista.final? : lista [op :n+6]
distancia 1 :n+1 ajustar dia.deserto :lista
end

to distancias :n1 :n2
if equalp :n1 :n2 [stop]
distancia :n1 distancias :n1+1 :n2
end
```

Notas sobre os procedimentos:

lista.6 :m — este procedimento constrói uma lista de **:m** dígitos, todos iguais a 6; serve-se da recursão **lista1.6 :p :m**. A lista construída será depois transformada dia a dia, representando no início de cada dia a quantidade de comida que cada um dos **:m** homens transporta.

soma :lista — este procedimento auxiliar dá como resultado a soma dos elementos (supostos números) de uma lista.

dia.deserto :lista — subtrai 1 a cada elemento da lista (significa o consumo diário em cada dia de marcha).

ajustar :lista — este procedimento corresponde às trocas de comida que os carregadores fazem ao fim de cada dia, na estratégia utilizada na resolução do problema; através da recursão auxiliar **ajustar1 :n :lista**, transforma por exemplo a lista **[5 5 5]** na lista **[3 6 6]**.

lista.final? :lista — este procedimento verifica se, a partir daqui, o mensageiro tem que contar apenas consigo; isto dá-se nas seguintes condições: ou o mensageiro está sozinho e tem 6 dias de comida, ou está acompanhado por um carregador que tem quando muito um dia de comida.

distancia :n.homens — este é o procedimento principal. A partir da lista inicial (produzida por **lista.6**), vai-a transformando até chegar à lista final, e conta o número de dias necessário para isso; depois, basta somar 6, isto é, o número de dias que o mensageiro pode andar sozinho se partiu, como é o caso, com seis dias de comida.

Quanto a saber se o programa funciona bem, não há como experimentar. Até agora não descobri nenhum bug e os resultados coincidem com os de Cláudia e Ana e dão soluções inteiras, como é próprio de um problema da Matemática Discreta (forçando um pouco, podia-se ainda dizer que o LOGO não corta os carregadores às décimas e centésimas, como o BASIC). Quanto à razão porque funciona, é pensar um bocadinho...

Eduardo Veloso

Um procedimento de cada vez

Thing, esse desconhecido!

Thing é um procedimento que existe em todos os LOGOS... até prova em contrário... Mas quando alguém fala nele, é certo e sabido que se levantam algumas caras espantadas e se ouve uma exclamação do tipo: «nunca ouvi falar, isso existe?!»

Bom, não é preciso mais para compreender que é possível viver muito tempo a programar em LOGO sem nunca recorrer ao **thing**. Mas chega um dia em que é preciso, e possivelmente o que tem acontecido é que muitos projectos interessantes já foram abandonados por desconhecimento do **thing**.

Para compreender o funcionamento de **thing**, criemos uma variável com o procedimento **make**:

make "polígono "pentágono

Então, se teclarmos

pr thing "polígono

o computador responde

pentágono

Isto é, o procedimento **thing** é uma operação que aplicada ao nome de uma variável dá como resultado o seu valor. Como sabemos, obteríamos a mesma resposta do computador se tivéssemos teclado

pr :polígono

Pode assim dizer-se que o prefixo : é uma abreviatura de **thing**. Porque não usar então sempre o prefixo : em vez de **thing**? Para o compreendermos, criemos outra variável com o nome "pentágono

make "pentágono "penta1

Assim, a variável de nome "polígono tem como valor a palavra "pentágono, que é por sua vez o nome de uma variável de valor "penta1. Se quisermos saber directamente *qual é o valor da variável cujo nome é o valor da variável de nome "polígono* (leia outra vez mais devagar...), não poderemos teclar **pr ::polígono** pois obteremos uma mensagem de erro. Mas podemos teclar **pr thing :polígono** e o computador responde **penta1**.

Então afinal para que serve o procedimento **thing**? Para podermos usar variáveis que tenham como valores nomes de variáveis. E por sua vez para que pode ser isso útil? Por exemplo, para fazermos um programa que calcule a derivada de uma função (derivadas com o LOGO!!!?). Um exemplo pessoal e mais comezinho: sem o **thing** o LOGO GEOMETRIA teria sido muito difícil, senão impossível, de programar...

Eduardo Veloso

Materiais para a aula de Matemática

Todas as máquinas de calcular, mesmo as mais simples, têm uma tecla M, a tecla de memória.

A função da memória é conhecida de todos, ela permite guardar números para posterior utilização. Mas será esta a sua única potencialidade? Como os guarda ela?

Exploremos a memória da nossa máquina de calcular!

Teclas

MRC	→	Traz ao visor o número guardado na memória.
M⁻	→	Subtrai o número indicado no visor ao número guardado na memória
M⁺	→	Adiciona o número indicado no visor ao número guardado na memória.

Visor

M	→	Indica que um número está guardado na memória. Quando não assinalado, a memória encontra-se a zero.

Estas são funções fundamentais das teclas de memória que interessa conhecer em cada máquina porque a sua apresentação varia de modelo para modelo.

O pior é, quase sempre, descobrir como «apagar» a memória, isto é, como pô-la de novo a zero. Nalguns modelos há uma tecla própria para essa função mas noutros não há. Que fazer neste caso?

Não ... desligar a máquina não «apaga» a memória!

Num modelo em que apareçam as 3 teclas de memória aqui indicadas (MRC, M-, M+) é um pequeno problema descobrir uma forma de «apagar» a memória, sem recorrer às instruções da máquina, é claro!

Agora, que já explorámos um pouco da memória da nossa máquina, utilizemo-la.

O cálculo de somatórios ou a estimação de valores de somatórios com um número infinito de termos pode ser um campo com várias possibilidades.

Estimemos $1+1/2+1/4+1/8+ \ldots$

Utilizando a memória podemos ir acumulando os termos, à medida que são calculados, com a possibilidade de, em qualquer momento, conhecer o valor do somatório.

Será que não poderíamos ter simplesmente somado $1+1/2+1/4$ utilizando a tecla da adição?

$1+1/2+1/4+ \ldots\ldots +1/1024 = 1{,}9990233$

$1+1/2+1/4+ \ldots\ldots +1/16384 = 1{,}9999386$

Destes cálculos podemos inferir que

$1+1/2+ \ldots +1/2^{n-1}+ \ldots = 2.$

Este tipo de inferências, facilitadas pela máquina de calcular, permitem resolver exercícios e problemas interessantes.

Mas uma pergunta se apresenta. Será que $1+1/2+ \ldots + 1/2^{n-1}+ \ldots$ é mesmo igual a 2? Não ultrapassará 2?

Há níveis em que esta dúvida permanecerá, há níveis em que poderão ser dúvidas destas que conduzam à necessidade de demonstrações e generalizações. E quantas vezes não há maneiras de esclarecer algumas destas dúvidas sem utilizar todo o formalismo matemático dos anos terminais do ensino secundário.

Mas mesmo quando a dúvida tem de permanecer será razão para banir situações deste tipo? Não serão elas suficientemente ricas dos pontos de vista matemático e pedagógico para surgirem bastante cedo?

Cristina Loureiro