CEFET-PR **Centro Federal de Educação Tecnológica do Paraná**
Departamento de Engenharia Industrial Elétrica ênfase Eletrônica.
UFPR **Universidade Federal do Paraná**

CURSO DE PROGRAMAÇÃO HP48G/GX

Anderson Juarez Moreira
Luciano da Silva Ribas

Maiores informações:

Luciano Ribas
R. Maurício Nunes Garcia, n280 apt103
Jardim Botânico – Curitiba-PR
CEP:80210-150
BRAZIL

Ou e-mail: *ribas@nupes.cefetpr.br*
Home-page: http://nupes.cefetpr.br/~ribas

Dedico este trabalho ao meu Pai e a minha Mãe:

Hamilton Ribas Filho
Matilde Pereira de Silva Ribas

que me deram todo o apoio e incentivo impressindíveis ao sucesso deste curso. Não poderia deixar de esquecer de agrader a paciência do meu irmão Cristiano e da minha querida irmãzinha Adriane.

Luciano Ribas

“Pai, inclina-nos a pensar sentindo, para que não guardemos gelo no cérebro, e induze-nos a sentir pensando para que não tenhamos fogo no coração.”

Emmanuel

CURSO DE PROGRAMAÇÃO HP48G/GX

Introdução ao Curso

A idéia de um curso de programação em HP48 surgiu devido à falta de uma literatura específica para a calculadora, os manuais até então encontrados tratavam o assunto de maneira superficial e eram geralmente publicados em inglês.

Para o usuário é extremamente frustrante possuir uma ferramenta poderosa e não saber como usá-la. O curso, nesse sentido, é uma alternativa àqueles que pretendem solucionar os seus problemas de forma adequada. Inicialmente o curso englobava operação e programação básica, no entanto, a experiência não foi gratificante, pois havia degraus muito grandes entre os conhecimentos de cada aluno em relação à calculadora. Após um estudo sobre o conteúdo do curso, o mesmo foi separado em dois módulos: - Operação - Programação.

Estes por sua vez são independentes entre si, pois a estrutura da calculadora permite que isto seja feito.

Não há a pretensão de que todos se tornem bons programadores após terminarem o curso. Mas, deve ficar claro que os métodos e ferramentas, fornecidos durante o curso, serão suficientes para que cada um possa aprimorar suas noções de programação até tornar-se um bom programador.

Distribuição da memória

A HP48 tem dois tipos de memória:

- ROM (Read-only memory - memória somente de leitura): é a parte da memória que não pode ser alterada, pois guarda a programação interna da calculadora (conj. de comandos). A HP48G/GX tem 512kbytes de ROM contra os 256kbytes da HP48S/SX e os 64kb de um AT286. Isto significa que a HP48 tem mais instruções internas que a BIOS de um computador!

- RAM (Random-access memory - memória de acesso aleatório): é a parte da memória que podemos modificar, gravar dados e apagar o seu conteúdo. Também é conhecida como memória do usuário, pois é nela que ficam armazenados os programas e demais objetos criados pelo usuário.

escrita de palavras.

- [α] + [←] ativa os caracteres minúsculos, e alguns caracteres especiais para o teclado numérico.
- [α] + [→] ativa os caracteres especiais em todo teclado alfanumérico.

Modos de exibição

Ao se realizar uma operação matemática sobre um determinado número, freqüentemente deseja-se obtê-lo sob um tipo de notação. A HP48 oferece os seguintes modos de exibição de números:

- [STD] notação padrão (todas as 12 casas decimais são mostradas).
- [FIX] notação com n casas decimais.
- [ENG] notação no formato de engenharia com n casas decimais.
- [SCI] notação no formato científico com n casas decimais.

Existem duas maneiras para se alterar o modo de exibição:

1. *Processo Interativo*:
   - pressione [→] [MODES]
   - selecione o campo NUMBER FORMAT usando as setas de movimentação do cursor.
   - pressione CHOOS para escolher o formato desejado e pressione OK
   - se o formato desejado for o FIX, SCI, ou ENG aparecerá um campo ao lado indicando o número de casas decimais desejadas.
   - finalmente pressione OK para confirmar o modo de exibição do display.
2. *Processo Manual:*
   - pressione [←] [MODES]
   - digite o número de casas decimais caso queira utilizar os modos FIX, ENG ou SCI.
   - pressione a tecla do menu correspondente ao formato desejado.

Apesar do primeiro processo ser muito mais intuitivo e fácil de ser executado iremos durante o decorrer do curso utilizar o segundo processo para a maioria dos casos, pois será neste formato (argumento + função) que iremos introduzir a programação. Além disto, veremos mais adiante que estes menus (FIX, SCI, etc.) são na verdade comandos, desta forma é bom começarmos a tomar contato com eles.

Seguindo a mesma idéia das calculadoras científicas normais existem os modos das funções trigonométricas: [DEG], [RAD], [GRAD], que podem ser alteradas por processos semelhantes aos mencionados acima. Existem, ainda os modos de exibição de vetores: [CYLI], [SPHER], [RE]; de exibição de números binários: [BIN], [DEC], [OCT], [HEX] e o sinal de “ponto” [FM,].

Teclas Especiais:

Dentre as teclas que possuem funções especiais, veremos inicialmente as que merecem maior atenção, pois serão imprescindíveis no decorrer do curso:

[EDIT] - edita um número real, complexo ou binário; ou ainda um programa, uma string, etc. Ou seja, edita o objeto que estiver na primeira linha do stack (pilha) da calculadora.
[CMD] - apresenta na tela os quatro últimos valores, programas, strings digitados.
[ARG] - retorna para o stack os argumentos utilizados pela última função executada.
[CLEAR] ou [DEL] - limpa todas as linhas do stack.

[DROP] ou [⇐] - apaga a primeira linha do stack.
[‘] - entra no modo de entrada de um objeto algébrico.
[“”] - entra no modo de entrada de uma string.

Reset - auto-test - clear memory - off-clock:

Além das combinações normais das teclas existem outras combinações importantes:

[ON]+[C] - Reseta a calculadora: ocorre um processo semelhante ao do computador quando se dá um BOOT: o stack e a tela gráfica (pict) são reinicializados, bibliotecas são recarregadas na memória ou instaladas, o path {HOME } é selecionado, ocorrem processos a nível de reorganização de memória e de registradores da CPU. É indicado em casos em que a calculadora travou, e não responde às teclas normais de interrupção.

[ON]+[D] - Executa um programa interno da ROM da HP que verificará o perfeito funcionamento da calculadora, testando o display (os pixels e a tensão), a saída serial e infra-vermelha, o clock, a RAM, a ROM, os cartões de memória, etc.

[ON]+[A]+[F] - Apaga a calculadora!! Limpa a RAM (apenas a localização dos programas na RAM, tornando-os inacessíveis)

[ON]+[SPC] - Desliga a calculadora e o circuito de clock. (o relógio da calculadora para de funcionar)

Troca da bateria - travamento acidental

Quando for necessária a troca das pilhas da calculadora deve-se observar o seguinte:

- nunca misture pilhas fortes e pilhas fracas, nem de marcas diferentes pois podem ocorrer vazamentos, danificando a sua calculadora.
- desligue a calculadora antes de trocar as pilhas e não aperte a tecla [ON] durante a troca das pilha, pois a calculadora tentará se ligar, descarregando um capacitor interno que supre o circuito (refresh) que mantém a memória RAM de sua calculadora.
- você tem cerca de três minutos para trocar as pilhas.

Se a calculadora por algum motivo qualquer travou e você já tentou pressionar [ON]+[C] e não ocorre mais nada, retire o apoio de borracha que fica na parte superior esquerda da calculadora e usando um palito de madeira reset a calculadora, caso isto não funcione não resta outra solução a não ser retirar as pilhas com a calculadora ligada e pressionar [ON] (este procedimento, na pior hipótese, apagará todos os programas que estavam na calculadora), a seguir deve-se colocá-las de volta na calculadora, e ligar a calculadora novamente. Se isto não funcionou, então será preciso apelar para medidas mais drásticas: descarrecar o capacitor interno do circuito de refresh que mantém a RAM. Para isso é necessário ligar a calculadora com as pilhas invertidas, o circuito de proteção contra a inversão acidental das pilha se encarregará de descarregar o capacitor, eliminando qualquer problema de software que tenha travado a calculadora.

Estrutura dos menus:

A calculadora HP48G/GX torna acessível a maioria de seus comandos através de vários menus. Estes menus foram organizados de forma a agilizar as operações básicas de cálculo e de programação, uma vez que estes comandos foram agrupados de acordo com as suas aplicações.

A estrutura dos menus da HP48, é muito semelhante à estrutura dos diretórios de um computador, pois apresentam a forma de uma árvore. Veremos mais a fundo os diretórios em um tópico à parte.

Além dos menus que contém as funções básicas de programação e de cálculo, existem outros menus que incorporam funções de edição de equações [EQUATION], de resolução de equações [SOLVE], de plotagem de gráficos [PLOT], de manipulação de objetos algébricos [SYMBOLIC], de controle de alarmes e do clock [TIME], de funções estatísticas [STAT], de controle e conversão de unidades [UNIT], de comunicação de dados [I/O], de controle sobre as bibliotecas e cartões de expansão [LIBRARY], de resolução múltiplas equações e biblioteca de equações [EQ LIB], de seleção dos flags internos da HP48 [MODES], de controle da memória e dos diretórios [MEMORY], de exibição da tabela dos caracteres gráficos [CHARS], etc.

Veremos nesta seção apenas uma introdução aos menus elementares e indispensáveis ao nosso curso de programação, sendo os restantes analisados oportunamente durante o decorrer do curso.

Gravando variáveis:

Podemos armazenar programas, números, strings e outros objetos. Para isso é necessário um comando chamado STO (store - armazenar) que é utilizado da seguinte maneira:

- Coloque o objeto a ser armazenado na primeira linha do stack.
- Digite entre os delimitadores ‘ ‘ nome da variável que armazenará o objeto.
- Pressione a tecla [STO] para gravar o objeto na variável.

Ou pelo por um método mais direto e rápido: (para uma variável já existente)

- Coloque o objeto a ser armazenado na primeira linha do stack.
- pressione a tecla [←┐] e em seguida a tecla de menu correspondente à variável que guardará o objeto.

recuperando variáveis:

Para recuperar um objeto armazenado em uma variável podemos proceder de duas formas diferentes:

1) simplesmente pressionando [┌→] e em seguida a tecla correspondente à variável do menu VAR que contém o objeto.

2) dentro do modo algébrico digitar o nome da variável e em seguida pressionando a tecla [RCL] (Recall - chamar).

Pilha

- Definição de Pilha
- Os comandos: EDIT, CLEAR, DROP, DUP,SWAP, ROT,ARG, UNDO, CMD
- Usando a Pilha em cálculos
- Exercícios de cálculos
- Pilha Interativa
- Usando a Pilha Interativa em cálculos
- Construindo equações com a Pilha

(Exercícios, equações, cálculos)

Definição de Pilha ou Stack:

O conceito de pilha é simples: imagine uma pilha de pratos, cada prato corresponde a um determinado número, quanto mais pratos colocarmos mais crescerá a pilha, porém só podemos tirar da pilha o último prato que foi colocado. Dessa forma funciona a pilha (ou Stack) da calculadora HP: ao digitarmos um número e depois outro, e mais outro, etc. Estamos na verdade empilhando números, porém só temos acesso direto ao último número colocado, ou seja, a primeira linha do Stack. No Stack podemos empilhar números, strings, programas, matrizes, enfim, todos os tipos de objetos da calculadora.

Felizmente existem certos comandos que nos permitem acessar outras linhas do Stack. Estes comando são geralmente desconhecidos pela maioria dos usuários da HP, porém são importantes ferramentas dentro da programação, pois economizam tempo e tornam os programas mais fáceis de entender.

O Stack da HP48 não tem um limite máximo de linhas, porem está sujeito a existência de memória disponível para guardar os dados que ficam no Stack.

Comandos: EDIT, CLEAR, DROP, DUP, SWAP, ROT, ARG, UNDO, CMD

Estes comandos são considerados os mais úteis durante a manipulação de objetos na pilha. Temos a seguir um resumo dobre cada um deles:

[EDIT] edita o objeto que está na primeira linha do Stack.
[CLEAR] limpa todo o Stack (o mesmo que [DEL])
[DROP] apaga a primeira linha do Stack.
[DUP] duplica a primeira linha do Stack.
[SWAP] troca o conteúdo da primeira linha com o conteúdo da segunda linha.
[ROT] move o objeto da terceira linha para a primeira linha.
[ARG] devolve ao Stack os argumentos da última função executada.
[UNDO] retorna o Stack anterior à última operação.
[CMD] mostra na tela um menu mostrando as quatro últimas entradas via teclado.

Usando a Pilha em Cálculos:

Agora iremos desenvolver o "Raciocínio do Stack", ou seja, vamos reaprender a álgebra e a programação de outra forma, não mais limitada a uma única linha ou expressão, mas agora, ilimitada como as linhas do Stack. Vamos aprender a montar equações e a resolver problemas de uma maneira muito mais simples e direta do que a maneira a qual estamos habituados.

Após dominar as operações matemáticas e as funções do teclado, temos condições de aprender a manipular os objetos na pilha, resolvendo problemas e calculando expressões.

Exercícios de Cálculos:

Utilizando as funções [DUP] ,[SWAP] e [ROT] encontre as respostas ao lado:

d*igite os objetos e encontre:*

{ HOME }

4:
3:
2:
1:

'x'
32
22
10

a) 10^22

b) 10^22-32

c) (10^22-32)

ap*ague o stack, redigite e encontre:*

a) 10*22+22/10

b) (10*22+22/10)-x

c) ((10*22+22/10)-x) + ((10*22+22/10)-x)/32

Ainda para os mesmos objetos do Stack, sem utilizar a calculadora, mostre a situação final do Stack para cada seqüência de comandos:

a) SWAP + DUP - + SWAP /
b) ROT SWAP ROT + DUP ROT - / SWAP *
c) SWAP ROT DROP DUP ROT + * /
d) SIN + ROT COS - /

Pilha interativa:

Existe um sistema especial de edição e manipulação dos objetos armazenados no Stack, esta função chama-se Pilha Interativa, e está disponível através do menu [←] [STACK] .

Para usar a Pilha Interativa proceda da seguinte forma:

1. pressione [←] [STACK].
2. use as setas do cursor para selecionar o objeto na linha desejada.
3. pressione [ENTER] ou [CANCEL] para sair da Pilha Interativa
4. para sair sem efetuar as mudanças feitas no Stack pressione [→] [UNDO]

A seguir temos as opções de comando disponíveis dentro da Pilha Interativa:

tecla: Descrição:
[ECHO] - Copia o conteúdo da linha corrente para a posição do cursor na linha de comando.
[VIEW] - Mostra ou edita o objeto na linha corrente usando o modo de apresentação mais adequado.
[→] [VIEW] - Mostra ou edita o objeto especificado pelo nome na linha corrente usando o formato correto.
[PICK] - Copia o conteúdo do nível corrente para o nível 1.
[ROLL] - Move o conteúdo da linha corrente para o nível 1.
[ROLD] - Move o conteúdo do nível 1 para a linha corrente.
[→ LIST] - Cria um objeto lista contendo todos os objetos desde o nível 1 até o nível corrente.
[DUPN] - Duplica uma porção do stack que vai desde o nível 1 a até a linha corrente para a parte superior a linha corrente, deslocando para cima o que já existia nesta porção do stack.
[DROPN] - Apaga uma porção do stack que vai desde o nível 1 a até o nível corrente.
[⇐] - Apaga o objeto no nível corrente

Objetos

- Definição de Objetos
- Lista dos Objetos
- Criação de Objetos

Definição de Objetos:

Os elementos básicos de Informação que a calculadora HP48 utiliza se chamam objetos. A calculadora HP48 pode armazenar e manipular diversos tipos de objetos. Os objetos são estruturas de dados internas da HP48, por ex: número real, complexo, matriz, list, etc.

Lista dos objetos:

Nome do Objeto	*tipo:*	*Nome do Objeto*	*tipo*
número real	0	diretório	15
número complexo	1	library	16
string	2	backup objeto	17
matriz real	3	inicio funções	18
matriz complexa	4	inicio comandos	19
listas	5	**Objetos de sistema:**	
nome global	6	sistema binário	20
nome local	7	extend real	21
programa	8	extend complex	22
objeto algébrico	9	matriz linkada	23
inteiro binário	10	character	24
objetos gráficos	11	objeto CODE	25
targged objeto	12	Library data	26
objeto unidade	13	External objeto	26-31
XLIB name	14		

Criação dos objetos:

Muitas das operações da calculadora HP48 são as mesmas para todos os tipos de objetos; por exemplo: usa-se o mesmo procedimento para armazenar um número real, uma matriz ou um programa. Algumas operações se aplicam somente a um tipo particular de objeto - por exemplo, não se pode extrair a raiz quadrada de um programa.

Agora iremos fazer um breve resumo dos objetos mais importantes da calculadora, sendo que os veremos em detalhes mais tarde.

- **Números Reais:** ($-9.99999999999 \times 10^{499}$ até $9.99999999999 \times 10^{499}$)
- **Números Complexos:** (forma retangular (x,y) e forma polar (r,))
- **Nomes:** são utilizados para identificar as variáveis. Para introduzir um nome é necessário pressionar a tecla [‘]
- **Objetos Algébricos:** os objetos algébricos, como os nomes, são delimitados por marcas (‘). Os objetos algébricos representam expressões matemáticas na forma extensa. O aplicativo Equation Writer ajuda a introduzir e a manipular os objetos algébricos, mostrando-os como se fossem impressos num livro.
- **Programas:** os programas são seqüências de comandos e outros objetos encerrados pelos delimitadores << e >>.
- **Cadeias de Caracteres ou Strings:** são seqüências de caracteres, utilizadas normalmente para representar textos em um programa. São delimitadas por “. Exemplo: “isto é um exemplo de uma string”

- **Listas:** as listas são seqüências de objetos agrupados, delimitados por chaves, por exemplo: { “eu sou uma string” 32 << DROP >> }. As listas permitem que se agrupem objetos para que possamos manipulá-los como se fossem um só.
- **Objetos Gráficos:** armazenam os gráficos que podem ser obtidos pelo traçado de equações ou pelo desenho artístico do usuário. Os objetos gráficos também podem ser armazenados em variáveis e manipulados na pilha, onde aparece da seguinte forma: Graphic n x m (onde n é o número de colunas e m o número de linhas que compõem o objeto gráfico)
- **Objetos de Unidades:** consta de um número real combinado com uma unidade ou uma expressão de unidades, por exemplo: 3_m, 27,2_m*kg/s^2
- **Objetos de Diretórios:** a calculadora HP48 utiliza objetos para especificar estruturas hierárquicas de diretórios armazenados. Os objetos de diretórios serão vistos em um tópico à parte.
- **Objetos Adicionais:**

objetos de segurança: permitem armazenar toda a memória da HP

objetos biblioteca: uma biblioteca é um diretório de comandos e operações que não estão incorporados na calculadora.

objetos XLIB: são objetos que fazem referência a uma biblioteca.

objetos EXTERNAL: são endereços que apontam instruções internas.

objetos CODE: são programas em linguagem avançada (de máquina).

Diretórios

- Estrutura em Árvore de um diretório
- Os comandos: STO, RCL, EDIT, PURGE, VIEW

Estrutura em Árvore de um diretório

Os objetos criados pelo usuário devem ser armazenados, e para isso são usadas as variáveis. Os comandos relacionados ao armazenamento e a recuperação de objetos, como já foram vistos, são: STO (armazena o obj. na variável), RCL (recupera o conteúdo da variável para o stack), [EDIT] (permite a visualização e a edição do objeto), PURGE (apaga a variável indicada), [VIEW] (ativa o modo de edição mais apropriado para o objeto)

Depois de armazenados em variáveis, é natural que exista uma desorganização na ordem das variáveis, ou seja, programas de cálculos, de gráficos, equações, jogos, aparecem todos misturados no mesmo menu. Para solucionar este problema utilizamos um menu diferente para cada programa, através de diretórios diferentes, ou seja, um jogo, ou um utilitário possuem diretórios diferentes.

O diretório principal da HP48 é o { HOME }, no entanto podemos criar outros diretórios no interior deste, formando uma estrutura conhecida como árvore de um diretório.

{ HOME }

{ CALC } {GRAPH} {SOUND} {JOGOS}

[PRG1] [PRG2] [A] [B] [WAV1] [WAV2] [TETRIS] [ANTS]

Memória

- Comandos do menu MEMORY
- Comandos dos sub-menus DIR e ARITH

Comandos do menu MEMORY

As funções do Menu [→ MEMORY] servem para manipular objetos armazenados em variáveis, através dele podemos copiar, mover, criar e editar variáveis.

A estrutura do menu MEMORY é a seguinte:

[MEMORY]

[EDIT] [CHOOSE] [VEHK] [NEW] [COPY] [MOVE] [RCL] [PURG]

[EDIT] - usado para modificar o conteúdo de uma variável total ou parcialmente.
[NEW] - utilizado para criar novas variáveis ou diretórios. (para criar um novo diretório o campo object deve estar vazio)
[COPY] - copia variáveis de um diretório para outro ou com outro nome no mesmo diretório.
[MOVE] - move variáveis de um diretório para outro.

Os subdiretórios EDIT, NEW, COPY e MOVE possuem as seguintes funções internas:

[EDIT] - edita o objeto presente no campo selecionado
[CANCL] - cancela a operação
[OK] - executa a operação
[RESET] Delete Value - limpa o campo selecionado
Reset All - restaura todos os campos às seus valores default

Comandos do Menu [← MEMORY]

[MEM] - retorna a memória livre disponível ao usuário.
[BYTES] - retorna o número de bytes que ocupa na memória o objeto colocado no nível 1 do stack. (retorna ainda o checksum do objeto)
[NEWOB] - cria outra cópia do objeto na memória.

[DIR]

[PATH] [CRDIR] [PGDIR] [VARS] [TVARS] [ORDER]

[PATH] - retorna o caminho dos diretórios desde {HOME} até o diretório corrente.
[CRDIR] - cria um diretório a partir de um nome para o diretório
[PGDIR] - apaga um diretório e todo o seu conteúdo
[VARS] - retorna uma lista contendo o nome de todas as variáveis do diretório
[TVARS] - retorna uma lista contendo o nome de todas as variáveis com um determinado tipo de conteúdo.
[ORDER] - ordena as variáveis de acordo com uma lista que contém os nomes das variáveis em suas novas posições.

Comandos do sub-menu ARITH

As funções do sub-menu ARITH podem otimizar em muito um programa, pois reduzem o número de operações ao se trabalhar com variáveis. Abaixo temos um resumo de suas funções:

- STO+ : soma, quando possível, o objeto do nível 2 ao o conteúdo da variável indicada no nível 1, armazenando o resultado na própria variável.
- STO- : subtrai, quando possível, o objeto do nível 2 ao o conteúdo da variável indicada no nível 1, armazenando resultado na própria variável.
- STO* e STO/ : são análogas as anteriores, e executam a multiplicação e a divisão.
- INCR : incrementa em 1 o conteúdo da variável indicada, retornando o novo valor da variável para a pilha.
- DECR : Decrementa em 1 o conteúdo da variável indicada, retornando o novo valor da variável para a pilha.
- SINV : calcula a recíproca (1/x) do conteúdo da variável indicada, armazenando na própria variável o novo valor.
- SNEG : inverte o sinal do conteúdo da variável indicada.
- SCON : calcula o conjugado do conteúdo da variável indicada, caso seja complexa, armazenando na própria variável o novo valor.

Sistema de Unidades

- Menus de Unidades
- Operações com Unidades
- Conversão de Unidades
- Os comandos: UBASE, UFACT, UVAL

A seguir temos um resumos das principais operações de gerenciamento de sistemas de unidades na calculadora HP:

[← UNITS] - apresenta vários diretórios que contêm o catálogo das unidades utilizadas em grandezas físicas como a massa, a velocidade, a luminosidade, etc. Permite que sejam criados objetos com unidades, e que seja feita conversões entre unidades consistentes.
[→ UNITS] - contém comandos para a conversão de objetos com unidades.

O menu [← UNITS] :

- Para criar um objeto com unidade, digite o valor e em seguida pressione a tecla correspondente à unidade desejada, o objeto pode também ser criado através da linha de comando usando a tecla "_".
- Podemos incluir prefixos quando estamos criando um objeto com unidade, por exemplo: 1_uA que eqüivale a 0.000001_A
- Para utilizar a conversão rápida utilize o procedimento:
  - coloque no nível 1 do stack o objeto que se deseja converter
  - encontre nos menus de unidades a unidade destino da conversão
  - e pressione [←] [unidade destino]
- Para executar a conversão rápida entre sistemas de unidades que não se encontram no menu do catálogo de unidades a solução é ou usar o comando [CONV] ou criar com um menu CST contendo um banco de novas unidades definidas pelo usuário (para um caso no qual se precisa repetir a conversão para diversos valores).

O menu [→ UNITS]:

[CONV] - converte o objeto com a unidade do nível 2 para a unidade indicada no nível 1, desprezando o valor numérico do objeto do nível 1.
[UBASE] - converte o objeto de unidades para outro objeto que possui apenas as unidades base do SI. (muito interessante)
[UVAL] - retorna apenas o valor numérico do objeto de unidades. (útil para conversões)
[UFACT] - fatora a unidade do objeto do nível 2 envolvendo a unidade do obj. do nível 1.
[→UNIT] - combina o valor presente nível 2 com a unidade do nível 1, desprezando a o valor do nível 1.

SOLVER BÁSICO

Para resolver uma equação usando métodos manuais seguimos o procedimento:
- Escrevemos a equação.
- Se possível isolamos a variável desconhecida.
- Substituimos os valores conhecidos.
- Calculamos o valor da variável desconhecida.

Quando utilizamos o Solve, executamos um processo semelhante, porém não é necessário isolar a variável desconhecida na equação.

SOLVE

1) SOLVE EQUATION - é utilizado para calcular uma variável desconhecida em uma equação ou expressão qualquer, quando usamos uma expressão o solve calcula o zero da mesma.

EQ: recebe a equação ou expressão que será utilizada.

EDIT: edita o objeto do campo selecionado.

CHOOSE: Seleciona uma equação ou expressão que esteja armazenada em alguma variável do diretório corrente. Se pressionado novamente, seleciona um novo diretório para procurar as equações ou expressões.

VARS: retorna uma lista com as variáveis da equação corrente

EXPR: retorna ao stack o valor da expressão para um valor pré-determinado de variável.

INFO: se pressionado após a calculadora ter efetuado algum cálculo no solve, retorna o status da variável e seu valor.

MENSAGENS:

a) Sign Reversal: a calculadora isolou 2 pontos que resultam sinais opostos quando aplicados a expressão ou a equação, mas não encontrou um ponto, entre os dois, que possa zerar a função, isto geralmente ocorre devido a falta de precisão ou a descontinuidade da função.

b) Extremum: a expressão ou a equação tende a zero para valores muito altos ou muito baixos, desta forma o zero (raiz) é obtido pela falta de precisão e portanto pode ou não ser um zero verdadeiro.

SOLVE POLINOMIAL

Utilizado para resolução de polinômios de tipo: $A_oX^n + A_1X^{n-1} + \ldots + A_{n-1}X^1 + A_nX^o = 0$

Coeficientes $[A_n \ A_{n-1} \ \ldots \ A_1 \ A_o]$*:* campo utilizado para a entrada (ou resolução) dos coeficientes do polinômio.

Roots: campo utilizado para a resolução (ou entrada) das raízes do polinômio.

Symb: se o campo selecionado for “coeficientes” ao se pressionar [Symb], retorna para o stack o polinômio. Se selecionarmos “roots” teremos no stack o produto das raízes que resulta no polinômio em questão.

Solve: encontra os coeficientes à partir das raízes do polinômio e vice-versa, de acordo com o campo selecionado.

SOLVE LINEAR SYSTEM

Utilizado para resolver sistemas de equaçoes lineares da forma: **A.X = B**

$$\begin{aligned} a_1 x + a_2 y + a_3 z &= b_1 \\ a_4 x + a_5 y + a_6 z &= b_2 \\ a_7 x + a_8 y + a_9 z &= b_3 \end{aligned}$$

$$\begin{matrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{matrix} \begin{matrix} x \\ y \\ z \end{matrix} = \begin{matrix} b_1 \\ b_2 \\ b_3 \end{matrix}$$

A: campo utilizado para a entrada da matriz que possui os coeficientes das variáveis.
B: campo utilizado para a entrada (ou cálculo) dos coeficientes lineares do sistema.
X: campo utilizado para o cálculo (ou entrada) do valor das variáveis.

SOLVER AVANÇADO

(EQ LIB MES)

MULTI-EQUATION SOLVER

Utilizada para calcular variáveis à partir de um conjunto de equações especificadas por uma lista:

{ ‘EQ_1’ ‘EQ_2’ ‘EQ_3’ ‘EQ_4’ . . . ‘EQ_n’ }

- esta lista deve ser armazanada em ‘EQ’ (variável que armazena as equações que a calculadora deverá processar)

- em seguida devemos inicializar o programa M.E.S. apertando [MINIT}.

- para calcular pressionamos [MSOLV] e obtemos um menu da variáveis.

- agora devemos colocar os valores nas variáveis que já conhecemos, digitamos o valor e pressionamos a tecla correspondente à variavel.

- podemos calcular o valor de uma variável pressionando + (variável desejada).

* a tecla [ALL] limpa o conteúdo de todas as variáveis e a combinação [ALL], faz com que a calculadora encontre o valor de todas as variáveis desconhecidas.

PROGRAMAÇÃO

Fundamentos de Programação:

- Como introduzir e executar um programa.
- Como editar um programa.
- Como utilizar variáveis locais.
- Programas que manipulam dados na pilha.
- Como utilizar sub-rotinas.
- Execução de um programa passo-a-passo.

Um programa é um objeto definido pelos delimitadores << e >>, e é composto por comandos e outros objetos. No exemplo seguinte calcularemos o volume de uma esfera, primeiro utilizando a calculadora normalmente e depois utilizando um programa.

Exemplo: Cálculo do Volume de uma Esfera.

O volume de uma esfera é dado através da seguinte fórmula: V = 4/3*pi*r^3

Para calcularmos o volume utilizando apenas as funções normais da calculadora, assumindo que já entramos com o raio na pilha, procedemos do seguinte modo:

3 [y] [π] [*] 4 [*] 3 [-] [NUM]

Se desejarmos calcular o volume de muitas esferas, podemos criar um programa. O programa seguinte também assume que o raio já está na pilha:

<< 3 ^ π * 4 * 3 / NUM >>

Como o programa é um objeto, é possível colocá-lo na pilha e salvá-lo em uma variável. Para jogar o programa na pilha depois de digitá-lo pressione [ENTER]. Para armazená-lo em uma variável, por exemplo: 'VOL' , pressione ['], escreva o nome da variável: VOL, e finalmente pressione a tecla [STO] para gravar o programa. Agora calcule o volume de qualquer esfera simplesmente executando [VOL] (selecione o menu VAR e pressione [VOL]). Pode-se executar VOL quantas vezes quisermos, pois agora ele tem o mesmo efeito que um comando incorporado.

VOL é um programa do tipo mais simples que existe; uma série de objetos e comandos, escritos na mesma ordem em que se escreveria normalmente para calcular o volume de uma esfera.

Como Escrever um Programa

Para definirmos o começo de um programa pressionamos ← [<< >>]. E aparecerá o indicador PRG, que indica o modo de entrada de programa. Neste modo, pressionando as teclas de comando escreve-se o nome do comando pressionado. (Também podemos escrever os comandos com caracteres alfabéticos).

O programa seguinte (chamado de SPH) calcula o volume de um segmento esférico de raio r e altura h, utilizando a seguinte fórmula:

$$V = 1/3*pi*h^2*(3*r-h)$$

A partir de agora utilizaremos para fins didáticos de um diagrama dos argumentos que são utilizados pelo programa, que é apropriado para mostrar como deve estar a pilha antes que se execute o programa e que resultados este programa retorna para a pilha. Temos a seguir o diagrama da pilha para o programa SPH:

Argumentos:	Resultados:
2:	2:
1:	1: volume

Este diagrama indica que SPH não usa argumentos da pilha e devolve o valor do setor esférico ao nível 1. (SPH assume que já se tenha armazenado o valor do raio na variável R e a altura na variável H).

A listagem abaixo mostra o programa na coluna da esquerda, os comentários na coluna da direita e como escrever o programa na coluna do meio. (Lembre-se: para digitarmos um comando podemos pressionar a tecla correspondente ou escrever o nome do comando)

Programa:	*Teclas:*	*Comentários:*
<<	← [<< >>]	Começa o programa.
'1/3	[‘] 1 [/] 3	Começa a expressão algébrica para calcular o volume.
*pi*H^2	[*] pi [*] H [Y^x] 2	Multiplica por h .
*(3*R-H)'	[*] ← [( )] 3 [*] R [-] H [→] [→]	Multiplica por 3r-h, completando o cálculo e finalizando a expressão.
NUM	[→NUM]	Converte a expressão para um número.
>>		Termina o programa.
	[ENTER]	Coloca o programa na pilha.
	[‘] SPH [STO]	Armazena o programa na variável SPH.

Como Executar um Programa:

Há várias maneiras de executar o programa SPH:
- Escreva SPH na linha de comando e pressione [ENTER].
- Selecione o menu VAR e pressione SPH.
- Se o programa ou o nome do programa estão no nível 1, então pressione [EVAL].

Como exemplo, utilize SPH para calcular o volume de um setor esférico de raio r=10 com uma altura de h=3.

Armazene os dados nas variáveis apropriadas (10 [‘] H [STO] 3 [‘] R [STO]). Depois selecione o menu VAR e ao executar o programa deveremos encontrar no nível 1 a resposta: 254.46004942

Como Editar um Programa:

Vamos modificar o programa SPH de maneira que ele armazene o conteúdo do nível 1 na variável H e o conteúdo da variável do nível 2 na variável R.

Pressione ↱[SPH] no menu VAR para jogar o programa na pilha, em seguida pressione [EDIT] para editar. Desloque o cursor após o primeiro delimitador de programa (<<) e insira os novos passos ao programa: 'H' STO 'R' STO. O programa fica da seguinte forma:

```
<<
 'H' STO
 'R' STO
 '1/3*pi*H^2*(3*R-H)'
   NUM
>>
```

Salve esta nova versão do programa na variável 'SPH2', pressionando [ENTER] para sair do modo de edição de programas e digite a sequência: 'SPH2' STO para gravar.

obs: Para abortar a edição de um programa pressione [CANCEL] e para confirmar o fim da edição de um programa pressione [ENTER].

Como Utilizar Variáveis Locais:

O programa SPH utiliza variáveis globais para armazenar e utilizar os dados. As desvantagens de se utilizar variáveis globais são as seguintes:

- Depois da execução do programa é necessário apagar as variáveis globais inúteis para liberar o menu VAR e a memória do usuário.
- Deve-se armazenar explicitamente os dados em variáveis globais antes de se executar o programa, ou fazer com que o próprio programa execute STO.

Veremos como as variáveis locais corrigem as desvantagens das variáveis globais. As variáveis locais são variáveis temporárias criadas por um programa. Existem somente enquanto se está executando o programa e não podem ser utilizadas fora do programa que as criou. Nunca aparecem no menu VAR.

Para criar variáveis locais, devemos utilizar a seguinte sequência de comandos e objetos, chamados de *estruturas de variável local:*

1. O comando → (pressione ↱[0])
2. Um ou mais nomes de variáveis.
3. Uma expressão algébrica ou um programa que utilizará as variáveis locais.

Esta estrutura tem a seguinte aparência:

<< → nome1 nome2 . . . $nome_n$ << programa >> >>

ou

<< → nome1 nome2 . . . $nome_n$ 'expressão algébrica' >>

Quando se executa o comando → em um programa, tomam-se n valores da pilha que são assumidos pelas variáveis nome1 nome2 . . . $nome_n$. Por exemplo, considere a seguinte pilha:

```
                    4:
3:                                  10
2:                                   6
1:                                  20
```

onde:
a) → a cria a variável local a=20.
b) → a b cria as variáveis locais a=6 e b=20.
c) → a b c cria as variáveis locais a=10, b=6 e c=20.

Agora vamos calcular o volume de um setor esférico utilizando variáveis locais, abaixo temos o diagrama dos argumentos do programa:

```
Argumentos:        Resultados:
2:      r          2:
1:      h          1:      volume
```

Programa:	*Comentários:*
<<	
→ r h	Cria as variáveis locais r e h para guardar o raio e a altura da esfera.
<<	
'1/3* *h^2*(3*r-h)'	Procedimento para a estrutura de variáveis locais no qual as variáveis locais são válidas.
→NUM	Converte a expressão para um número.
>>	
>>	
[ENTER]	Coloca o programa na pilha.
['] SPHVL [STO]	Armazena o programa na variável SPHVL.

Para executar o programa entre com os dados na pilha e chame o programa, por exemplo: raio=10 e altura=3. Entre com os dados da seguinte forma: 10 [ENTER] 3 [ENTER] , execute o programa: VAR SPHVL

Programas que Manipulam Dados na Pilha

Os programas anteriores SPH e SPHVL utilizam variáveis para armazenar e recuperar dados. Um método de programação alternativa manipula números na pilha, sem armazaná-los em variáveis. Este método geralmente tem um tempo de execução mais rápido. Porém o método de manipulação da pilha tem várias desvantegens:

- Quando se escreve um programa, deve-se localizar a posição dos dados na pilha. Por exemplo os argumentos devem ser duplicados se forem utilizados por mais de um comando.

- Um programa que manipula dados da pilha é geralmente mais difícil de ler e entender que um programa que utiliza variáveis.

O programa SPHSTACK utiliza o método de manipulação da pilha para calcular o volume de um setor esférico.

Argumentos:		Resultados:	
2:	r	2:	
1:	h	1:	volume

Programa:	*Comentários:*
<<	
DUP	Duplica o número do nível 1 da pilha.
ROT	Move o número que se encontra no nível 3 para o nível 1.
3 *	Multiplica o raio por 3.
SWAP -	Troca os números do nível 1 (altura) pelo 2 (raio) e calcula 3r-h.
SWAP SQ *	Troca a cópia da altura pelo nível 1, acha a raiz quadrada da altura e multiplica por 3r-h.
pi * 3 /	Multiplica por pi e divide por 3, completando o cálculo.
→NUM	Converte a expressão para um número.
>>	
[ENTER]	Coloca o programa na pilha.
['] SPHSTACK [STO]	Armazena o programa na variável SPHSTACK.

Execução de um Programa Passo-a-Passo

É mais fácil compreender como funciona um programa se o rodarmos passo-a-passo, observando o efeito de cada comando. Este procedimento geralmente é utilizado para corrigir erros de programação dentro de um programa e também para ajudar a enteder como programas escritos por outras pessoas funcionam, por isso este procedimento é chamado de DEBUG.

As operações para executar o DEBUG estão contidas no menu PRG RUN.

Comandos do DEBUG:

DBUG: toma como argumento o nome, ou o programa a ser executado passo-a-passo, começando a execução do programa e depois suspendendo como se fosse executado o comando HALT.
SST: executa o próximo comando do programa suspenso.
SST↓: igual a SST, porém executa subrotinas do programa principal passo-a-passo também.
NEXT: mostra no display o próximo comando a ser executado.
HALT: suspende a execução de um programa na posição do comando HALT.
CONT: retorna a execução de um programa suspenso.
KILL: cancela a execução e o processamento passo-a-passo de todos os programas suspensos.

Exemplo de execução passo-a-passo:

1. Coloque o programa ou o nome do programa no nível 1.
2. Pressione [PRG] [NEXT] RUN DBUG.
3. opcional: pressione NEXT para mostrar o próximo comando.
4. Pressione SST para executar cada passo do programa.

obs:
- para abandonar a execução do programa pressione KILL
- para continuar a executar até o fim do programa normalmente pressione CONT.

Se for necessário executar passo-a-passo um programa a partir de um determinado ponto no seu interior, basta colocarmos um comando de parada: HALT. Executamos o programa normalmente, e ele irá parar quando chegar no comando HALT, passando o controle ao DEBUG. A partir daí, podemos executar passo-a-passo o programa seguindo os ítens 2, 3 e 4 descritos acima.

ESTRUTURAS CONDICIONAIS E TESTES:

Veremos comandos e estruturas que permitem aos programas fazer perguntas e tomar decisões:

- estruturas de comparação e funções lógicas: que permitem que um programa faça um teste para verificar se existe uma condicão específica.

- estruturas de programas chamadas de estruturas lógicas: usam os resultados de um teste para tomar decisões.

Exemplo: Estruturas Condicionais e Testes

O programa abaixo usa um teste dentro de uma estrutura condicional para executar a seguinte tarefa:

“Se os dois números da pilha tem o mesmo valor, elimine um dos dois e armazene o outro na variável V1. Caso contrário, os dois números são diferentes, armazena o número do nível 1 em V1 e o número do nível 2 em V2”

Programa:	*Comentários:*
<<	
DUP2	Duplica os números dos níveis 1 e 2 da pilha.
IF	Começa a estrutura de teste condicional.
SAME	Verifica se os dois números são iguais.
THEN	Executa somente se o teste retornou verdadeiro.
DROP	Apague um e armazena o outro
‘V1’ STO	na variável V1
ELSE	Executa somente se o teste retornou falso.
‘V1’ STO	Armazene o número do nível 1 em V1 e
‘V2’ STO	o número do nível 2 em V2.
END	Termina a estrutura condicional.
>>	
[ENTER]	Termina a digitação do programa.
[‘] TST [STO]	Grava o programa da pilha em TST.

Comandos de Testes

Um teste é uma seqüência de comandos que devolvem um resultado verdadeiro (1) ou falso (0). Por exemplo, digite 10 5 >, você irá obter 1 significando que a condição do teste é verdadeira, ou seja que 10 é maior do que 5; da mesma forma podemos digitar 5 10 >, neste caso obteremos 0 indicado que a nossa suposição é falsa.

Os comandos utilizados nos testes podem se classificar em:
- *funções de comparação.*
- *funções lógicas.*
- *comandos de teste dos flags*

Funções de Comparação:

<	menor que
>	maior que
<	menor ou igual
>	maior ou igual
==	igual
=	diferente
SAME	igual (+ rápido que ==)

Funções Lógicas:

AND	retorna 1 se os dois argumentos são V.
OR	retorna 1 se um ou os dois argumentos são V.
XOR	retorna 1 se um ou outro é V mas não os dois.
NOT	retorna 1 se o argumento for F.

ESTRUTURAS CONDICIONAIS:

As estruturas condicionais permitem a calculadora tomar uma decisão baseada em um resultado de um ou mais testes. As estruturas condicionais são:

- IF ... THEN ... END.
- IF ... THEN ... ELSE ... END.
- CASE ... END.

A Estrutura IF ... THEN ... END

IF ... THEN ... END executa uma sequência de comandos (cláusula verdadeira) se a condição for verdadeira (cláusula de teste). Sua sintaxe é a seguinte:

IF *cláusula de teste* THEN *cláusula verdadeira* END

A cláusula de teste (condição de teste) pode ser uma sequência de comandos (por exemplo: A B >) ou um algébrico (por exemplo: 'A>B'). No caso de ser um algébrico a expressão é avaliada automaticamente para um número, não necessitando do EVAL ou de NUM.

Exemplo 1:

Os dois programas seguintes fazem um teste do valor no nível 1. Se o valor é positivo tranforma-se em negativo. O primeiro programa usa uma sequência de comandos como cláusula de teste:

```
<<  DUP  IF  0 >  THEN  NEG  END  >>
```

O valor na pilha deve ser duplicado porque o comando > extrai dois argumentos da pilha (a cópia do valor é feita pelo DUP)

A seguinte versão usa um algébrico como cláusula de teste:

```
<<  x  <<  IF  'x>0'  THEN  x  NEG  END  >>  >>
```

Exemplo 2:

Este programa multiplica dois números se ambos são diferentes de zero:

Programa:	*Comentários:*
<<	
→ x y	Cria variáveis locais x e y que contém os dois números da pilha.
<<	
IF	Começa a estrutura de teste condicional.
'x=0'	Verifica se o número é diferente de zero, deixando o resultado do teste na pilha.
'y=0'	Verifica se o número é diferente de zero, deixando o resultado do teste na pilha.
AND	Verifica se os dois testes são verdadeiros.
THEN	Executa somente se o teste retornou verdadeiro.
x y *	Se AND devolver verdade, multiplica os dois números.
END	Termina a estrutura condicional.
>>	
>>	

O seguinte programa tem o mesmo efeito que o anterior:

```
<<  DUP2  IF  AND  THEN  *  ELSE  DROP2  END  >>
```

Como funciona a estrutura IF...THEN...END: IF começa a cláusula do teste, deixando o resultado do teste na pilha. THEN extrai o resultado do teste da pilha. Se o valor é diferente de zero, executa-se a cláusula verdadeira. Caso contrário, a execução do programa continua após o comando END.

A Estrutura IF...THEN...ELSE...END

IF...THEN...ELSE...END executa uma seqüência de comandos (cláusula verdadeira) se o teste é verdadeiro e outra (cláusula falsa) se o teste é falso. Sua sintaxe é a seguinte:

IF *cláusula de teste* THEN *cláusula verdadeira* ELSE *cláusula falsa* END

Se a cláusula de teste for um algébrico (por exemplo: 'A>B'), a expressão é avaliada automaticamente para um número, não necessitando do EVAL ou de NUM.

Exemplo 1:

O seguinte programa toma um valor da pilha e calcula sin(x)/x. Porém em x=0 a divisão estará errada, por isso o programa devolve o valor limíte 1 neste caso:

```
<< → x  <<  IF  'x=0'  THEN  x SIN x /  ELSE  1  END >>
```

Exemplo 2:

Este programa, como o exemplo 2 para IF...THEN...END, multiplica dois números se ambos são diferentes de zero. Porém o programa devolve a cadeia "ZERO" se algum dos dois é 0.

Programa:	*Comentários:*
<<	
→ n1 n2	Armazena nas variáveis locais os níveis 1 e 2 da pilha.
<<	
IF	Começa a estrutura de teste condicional.
'n1=0 AND n2=0'	Verifica se ambos os números são diferentes de zero, deixando o resultado do teste na pilha.
THEN	Executa somente se o teste retornou verdadeiro.
n1 n2 *	Se AND devolver verdade, multiplica os dois números.
ELSE	Executa somente se o teste retornou falso.
"ZERO"	Devolve a cadeia "ZERO"
END	Termina a estrutura condicional.
>>	
>>	

O seguinte programa tem o mesmo efeito:

```
<<  DUP2  AND  IF  THEN  *  ELSE  DROP2  "ZERO"  END  >>
```

Como funciona a estrutura IF...THEN...ELSE...END: IF começa a cláusula do teste, deixando o resultado do teste na pilha. THEN extrai o resultado do teste da pilha. Se o valor é diferente de zero, executa-se a cláusula verdadeira. Caso contrário, se executa a cláusula falsa. Depois da execução da cláusula apropriada, a execução do programa continua após o comando END.

A Estrutura CASE...END

A estrutura CASE...END permite executar uma série de *casos* (testes). O primeiro teste que tem um resultado verdadeiro causa a execução da correspondente cláusula verdadeira, finalizando a estrutura CASE...END. Opcionalmente, pode-se incluir após o último teste uma cláusula de default que se executará se todos os testes forem falsos.

A estrutura CASE...END tem a seguinte sintaxe:

CASE
cláusula de teste1 **THEN** ***cláusula verdadeira1*** **END**
cláusula de teste2 **THEN** ***cláusula verdadeira2*** **END**
. . .
***cláusula de teste*n** **THEN** ***cláusula verdadeira*n** **END**
cláusula default *(opcional)*
END

Exemplo 1:

O seguinte programa armazena o argumento do nível 1 em uma variável chamada STR se o argumento for uma cadeia; numa variável chamada LIST se o argumento for uma lista e em PRG se o argumento for um programa.

Programa:	*Comentários:*
`<<`	
`→ y`	Armazena o argumento na variáveis local y.
`<<`	
`CASE`	Começa a estrutura do case.
`y TYPE 2 SAME`	Case 1: se o argumento é do tipo cadeia,
`THEN y 'STR'`	armazene-o na variável 'STR'
`STO END`	

```
    y  TYPE   5  SAME          Case 2: se o argumento é do tipo lista,
    THEN  y  'LIST'           armazene-o na variável 'LIST'
          STO  END

    y  TYPE   8  SAME          Case 3: se o argumento é do tipo programa,
    THEN  y  'PRG'            armazene-o na variável 'PRG'
          STO  END

   END                        Termina a estrutura case.
  >>
>>
```

Como funciona a estrutura CASE...END: Ao executar CASE, se calcula a cláusula teste1. Se o teste é verdadeiro, se executa a cláusula do test1, e a execução salta para END. Se a cláusula teste é falsa, a execução passa para a próxima cláusula teste. A execução na estrutura CASE continua até que todas as cláusulas teste tenham sido avaliadas como falsas. Opcionalmente, pode-se incluir uma cláusula de default, que é executada caso todas as outras cláusulas tenham sido avaliadas como falsas.

O Comando IFT (if-Then-End)

O comando IFT toma dois argumentos: o resultado de um teste no nível 2 e um objeto no nível 1 (a "cláusula verdadeira"). O objeto do nível 1 é executado se o resultado do teste é verdadeiro.

Exemplo: O programa abaixo extrai um número da pilha e retorna a cadeia "POSITIVO" se o número for positivo:

```
<<  0  >  "POSITIVO"  IFT  >>
```

A função IFTE

A função IFTE toma três argumentos: o resultado de um teste no nível 3, e os objetos dos níveis 2 e 1. O objeto do nível 2 (correspondente a cláusula verdadeira) é executado se o resultado do teste é verdadeiro. Caso contrário, o objeto do nível 1 (a cláusula falsa) é executado.

Exemplo1: O programa abaixo extrai um número da pilha e retorna a cadeia "POSITIVO" se o número for positivo e "NEGATIVO" caso o número seja negativo:

```
<<  0  >  "POSITIVO"  "NEGATIVO"  IFTE  >>
```

Exemplo2: Podemos utilizar a função IFTE dentro de um algébrico: O programa abaixo calcula sin(x)/x se x é diferente de zero. Se x é zero, o programa devolve 1:

<< → x 'IFTE(x=0 , S N(x)/x , 1)' >>

ESTRUTURAS ITERATIVAS:

As estruturas iterativas executam uma parte de um programa repetidamente. Existem dois tipois básicos de laços:

- *Para um laço definido,* o programa especifica previamente quantas vezes será executada a cláusula do laço.
- *Em um laço indefinido,* o programa utiliza um teste para determinar se deve executar novamente a cláusula do laço.

Estruturas Iterativas Definidas

Fazem parte das estruturas interativas definidas as seguintes variações:

- **START...NEXT e START...STEP.**
- **FOR...NEXT e FOR...STEP.**

A Estrutura START...NEXT

START...NEXT executa uma parte do programa um determinado número de vezes. Sua sintaxe é:

início fim **START** ***cláusula do laço*** **NEXT**

Exemplo: O programa seguinte cria uma lista que contém dez cópias da cadeia "ABC":

<< 1 10 START "ABC" NEXT 10 LIST >>

Como funciona START...NEXT: START toma dois números da pilha (início e fim) e os armazena como valores inicial e final para o contador do laço. Depois, executa a *cláusula do laço*. NEXT incrementa o contador em 1 e verifica se este valor é menor ou igual ao fim. Se é, executa novamente a *cláusula do laço.*

A Estrutura START...STEP

START...STEP funciona exatamente da mesma forma que o START...NEXT, exceto que permite especificar um incremento diferente de 1. Sua sintaxe é:

***início fim* START *cláusula do laço incremento* STEP**

Exemplo: O programa seguinte toma um número x da pilha e calcula o quadrado deste número x/3 vezes:

```
<< DUP x << x 1 START x SQ -3 STEP >>
```

Como funciona START...STEP: START toma dois números da pilha (início e fim) e os armazena como valores inicial e final para o contador do laço. Depois, executa a *cláusula do laço*. STEP toma o *incremento da pilha* e incrementa o contador com este valor. O incremento pode ser positivo ou negativo. Se é positivo, executa novamente a *cláusula do laço* quanto o contador é menor ou igual ao fim. Se o incremento é negativo, executa o laço quando o contador é maior ou igual ao fim.

A Estrutura FOR...NEXT

Um laço FOR...NEXT executa uma parte de um programa um número especificado de vezes, utilizando uma variável local como contador das iterações. Pode-se utilizar esta variável dentro do laço. Sua sintaxe é:

***início fim* FOR *contador cláusula do laço* NEXT**

Exemplo1: O programa seguinte coloca na pilha os quadrados dos número inteiros de 1 a 10:

```
<< 1 10 FOR j j SQ NEXT >>
```

Exemplo2: O programa seguinte calcula o fatorial de um número da pilha:

```
<< 1 1 ROT FOR j j * NEXT >>
```

Como funciona FOR...NEXT: FOR toma dois números da pilha (início e fim) e os armazena como valores inicial e final para o contador de iterações, depois cria uma variável local *contador* como contador de iterações. Depois, se executa a cláusula do laço. NEXT incrementa o *contador* em 1 e verifica se este valor é menor ou igual ao *fim*. Se é, executa novamente a *cláusula do laço*. Ao sair do laço o contador é apagado, ou seja ele só existe dentro da cláusula do laço.

A Estrutura FOR...STEP

FOR...STEP funciona exatamente da mesma forma que FOR...NEXT, exceto que permite especificar um incremento diferente de 1 ao contador. Sua sintaxe é:

***início fim* FOR *contador cláusula do laço incremento* STEP**

Exemplo: O programa seguinte calcula os quadrados dos inteiros ímpares de 1 a 15:

```
<< 1 21 FOR j j SQ 2 STEP >>
```

Como funciona FOR...STEP: FOR toma dois números da pilha (início e fim) e os armazena como valores inicial e final para o contador de iterações, depois cria uma variável local *contador* como contador de iterações. Depois, executa a cláusula do laço. STEP toma o *incremento* da pilha e incrementa o *contador* com este valor, e verifica se o valor do *contador* é menor ou igual ao *fim*. Se é, executa novamente a *cláusula do laço*. Ao sair do laço o contador é apagado, ou seja ele só existe dentro da cláusula do laço.

Estruturas Iterativas Indefinidas

Fazem parte das estruturas interativas definidas as seguintes variações:

- **DO...UNTIL...END.**
- **WHILE...REPEAT...END.**

A Estrutura DO...UNTIL...END

DO...UNTIL...END executa repetidamente um laço enquanto a cláusula de teste retornar um valor falso. Como se executa primeiramente a cláusula do laço e depois a cláusula do teste, executa-se ao menos uma vez o laço. Sua sintaxe é a seguinte:

DO *cláusula do laço* UNTIL *cláusula de teste* END

Exemplo: O programa seguinte calcula n + 2n + 3n + ... para um valor de n. O programa para quando a soma exceder 10000, e devolve a soma e o coeficiente de n:

Programa:	*Comentários:*
<<	
DUP 1 → n s c	Duplica n e armazena o valor em n e s; inicializa o contador c com 1.
<<	
DO	Começa a estrutura do.
'c' INCR	Incrementa o contador em 1 e devolve para a pilha o novo valor de c.
n * 's' STO+	Calcula c*n, e soma o produto a s.
UNTIL	Começa a estrutura de teste.
s 10000 >	Repete o laço até que s>10000.
END	Termina a cláusula de teste.
s c	Coloca na pilha s e c.
>>	
>>	

Como funciona DO...UNTIL...END: DO começa a cláusula do laço. UNTIL finaliza a cláusula do laço e começa a cláusula de teste. A cláusula de teste deixa o resultado do teste na pilha. END extrai o resultado deste teste da pilha. Se o valor é zero, executa novamente a cláusula do laço; caso contrário, a execução do programa continua após o END.

A Estrutura WHILE...REPEAT...END

WHILE...REPEAT...END avalia repetidamente um teste e executa a cláusula do laço se o teste é verdadeiro. Como a cláusula do teste ocorre antes da cláusula do laço, nunca se executa um laço sem antes verificar se a cláusula do teste é verdadeira. Sua sintaxe é a seguinte:

WHILE *cláusula de teste* REPEAT *cláusula do laço* END

Exemplo: O programa seguinte realiza uma divisão por dois sobre o número que está na pilha repedidamente sempre que o resultado da divisão seja divisível por um número par.

<< WHILE DUP 2 MOD 0 == REPEAT 2 / DUP END DROP >>

Como funciona WHILE...REPEAT...END: WHILE executa-se a cláusula do teste e devolve o resultado do teste para a pilha. REPEAT toma os valores da pilha. Se o valor é diferente de zero, continua a execução do laço; caso contrário, a execução do programa continua após o END.

PROGRAMAS INTERATIVOS:

Para escrevermos programas interativos necessitamos de comandos específicos que permitam a comunicação do programa com o usuário. Estes comandos possuem funções específicas, e são basicamente de dois tipos:

- de entrada: **inform, choose**
- de saída: **disp, msgbox**

COMANDOS DE SAÍDA :

DISP

Mostra em uma determinada linha da tela um objeto especificado. Por exemplo:

```
3:  “Isto é um teste”          mostrará na primeira linha do display a
2:                  1          string “Isto é um teste”
1:              DISP
```

```
<<   1  100  FOR  n                mostrará na segunda linha do display
                n  2  DISP         uma contagem de 1 até 100.
             NEXT
>>
```

```
<<   DUP
     SIZE  1  SWAP
              FOR  n                    mostrará na tela da calculadora uma
                   DUP  n  n  SUB       letra por vez de uma string
                   1 DISP
              NEXT
>>
```

CLLCD

Limpa a tela da calculadora. Exemplo:

```
<<  CLLCD   "EU SEI PROGRAMAR"  1  DISP  >>      mostra um messagem na tela.


<< 1  1000  FOR  n                              calcula e mostra o seno
                CLLCD  n  SIN  1  DISP          dos números de 1 a 1000.
             NEXT
>>
```

BEEP

Emite um beep com uma frequência e uma duração especificadas. Exemplo:

```
<<    600  .7  BEEP        emite um beep de 600Hz durante .7s
     1300  .2  BEEP        emite um beep de 1300Hz durante .2s
>>

<<  CLLCD  1  20000
       FOR  f
          f   1  DISP          exibe na tela o valor da freq. do beep
          f  .3  BEEP          emite um beep com uma freq. f  durante .2s
       20 STEP
>>
```

FREEZE

Congela uma determinada área do display:

```
1 FREEZE  -  área de status
2 FREEZE  -  área da pilha
4 FREEZE  -  área do menu
```

```
<<  "ABC
     DEF
     GHI"  CLLCD  1  DISP
     3  FREEZE                        congela a área do status+pilha  (3=1+2)
>>
```

WAIT

Provoca uma pausa na execução do programa em n segundos. (se n=-1 espera até que uma tecla qualquer seja pressionada, retornando a posição da tecla pressionada)

```
<<  700  .4  BEEP
    800  .2  BEEP
     .4   WAIT
    900  .5  BEEP
>>
```

```
<<  "Pressione qualquer tecla..."
     1  DISP  3  FREEZE  -1  WAIT
    "A tecla pressiona é:"  SWAP  +
     1  DISP
>>
```

MSGBOX

Utilizado para mostrar mensagens curtas ao usuário: erros, procedimentos, etc.

```
<<  "Cálculo terminado e verificado"  MSGBOX  >>
```

COMANDOS DE ENTRADA:

KEY

Retorna o resultado de um teste para verificar se alguma tecla está sendo pressionada, em caso afirmativo retorna também a posição da tecla pressionada (linha, coluna). Exemplo:

```
<<  WHILE
      KEY  NOT
    REPEAT
      "Nenhuma tecla pressionada"  1  DISP
    END
    "A tecla pressionada foi:"  SWAP  +  1  DISP
     3  FREEZE
>>
```

INFORM

Cria uma estrutura de entrada de dados no padrão dos menus da HP. Utilizando este comando podemos inserir dados através de menus e janelas, utilizando comandos à eles associados (EDIT, CANCEL, OK, RESET, CALC, TYPES). Os argumentos para a utilização do comando são os seguintes:

stack:

5: “conhecendo o inform” - pequena string que será exibida no topo da caixa.
4: { “A=” “B=” “C=” } - lista de strings contendo a identificação dos campos.
3: { 1 5 } - lista contendo o número de campos por linha e a distância entre a identificação e o campo.
2: { 0 0 0 } - valores assumidos pelos campos quando se dá um RESET
1: { 1 2 3 } - valores defaut (iniciais)

Este comando retorna o valor 0 se a entrada foi cancelada e 1 se a entrada foi confirmada, neste caso o comando retorna uma lista, na 2 linha do stack, contendo os novos conteúdos dos campos apresentados.

CHOOSE

O comando CHOOSE é utilizado quando necessitamos que o usuário faça uma escolha dentro de um programa. O comando gera um menu de barras, contedo as opcões, ou objetos desejados. O comando é utilizado com os seguintes argumentos:

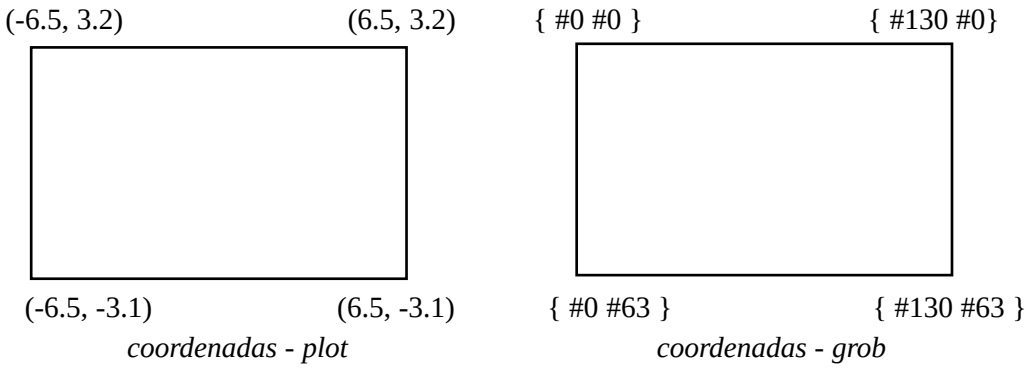
3: “Escolha a opção:” - pequena string que será exibida no topo da caixa.
2: { “LER” “GRAVAR” << 1 >> ‘C’ } - lista contendo as opções (objetos).
1: 1 - posição inicial da barra sobre o menu.

A barra deve ser deslocada até o objeto desejado e em seguida pressionada a tecla ENTER ou OK para efetuarmos a escolha, ou CANCEL para cancelarmos. Se a função foi cancelada teremos como resposta o valor 0 no stack. Se a função foi confirmada temos o valor 1 como resposta, e na segunda linha do stack, teremos a opção (objeto) selecionada.

COMANDOS GRÁFICOS:

TELA GRÁFICA:

A maneira de acessar um determinado pixel na tela gráfica da HP é através de um par de coordenadas:



coordenadas - plot *coordenadas - grob*

Observe que as coordenadas do pixel são dadas em forma de uma lista contendo dois binários inteiros, o primeiro idicando a coluna e o segundo indicando a linha. Estas coordenadas são a forma mais coveniente de se trabalhar com objetos gráficos. Já as coodenadas do plot são definidas através da variável PPAR e indicam pontos na tela de uma função matemática traçada através do plot, por isso seu uso não é muito útil pois freqüêntemente alteradas.

COMANDOS GRÁFICOS:

PICT

Variável na qual está armazenado o objeto gráfico corrente.

PDIM

Dimensiona o tamanho da tela gráfica de acordo com a coordenada do canto inferior direito da nova tela, colocadas nos níveis 2 e 1 do stack (altura largura).

LINE

Desenha uma linha na tela gráfica corrente entre dois pontos colocados no stack.

TLINE

Desenha uma linha na tela gráfica corrente entre dois pontos colocados no stack, trocando o estado de cada ponto encontrado no caminho (on-off xor).

BOX

Desenha um retângulo na tela gráfica usando duas coordenadas no stack que indicam a diagonal do retângulo.

ARC

Desenha um arco na tela gráfica centrado na coordenada presente no nível 4 do stack, com um raio especificado no nível 3, com o ângulo inicial e final indicados nos níveis 2 e 1 respectivamente.

PIXON

Liga o pixel da tela gráfica ndicado no nível um do stack.

PIXOFF

Desliga o pixel da tela gráfica ndicado no nível um do stack.

PIX?

Retorna 1 se o pixel da tela gráfica indicado no nível um do stack estiver ligado, e zero se estiver desligado.

PVIEW

Mostra a tela gráfica corrente a partir de um ponto especificado (geralmente {#0#0}).

PX→C

Converte as coordenadas de pixel para coordenadas do tipo plot.

C→PX

Converte as coordenadas do tipo plot para coordenadas de pixel.

→GROB

Converte o objeto (nível 2) para o objeto gráfico usando o número real n (de 0 a 3 no nível 1 do stack) para especificar o tamanho do caracter. O objeto gráfico resultante é uma string cujo tamanho do caracter é pequeno (n=1), médio (n=2) ou grande (n=3). Para n=0 o tamanho do caracter é o mesmo que para n=3, exceto para objetos algébricos e unidades, o objeto gráfico resultante neste caso será uma tela do Equation Writer.

BLANK

Cria um objeto gráfico em branco a partir da largura e da altura especificadas nos níveis 2 e 1 respectivamente.

GOR

Sobrepõe o objeto gráfico presente no nível 1 do stack ao objeto gráfico presente no nível 3 do stack, a partir da coordenada especificada no nível 2 do stack.

GXOR

Sobrepõe, usando um XOR lógico, o objeto gráfico presente no nível 1 do stack ao objeto gráfico presente no nível 3 do stack, a partir da coordenada especificada no nível 2 do stack.

SUB

Extrai uma região do objeto gráfico presente no nível três definida por duas coordenadas presentes nos níveis 2 e 1 do stack que definem a diagonal do retângulo extraído.

REPL

Insere o objeto gráfico presente no nível 1 do stack ao objeto gráfico presente no nível 3 do stack, a partir da coordenada especificada no nível 2 do stack.

→LCD

Mostra o objeto gráfico do nível 1 no display-stack.

LCD→

Retorna para o nível 1 o objeto gráfico que está sendo mostrado na tela.

SIZE

Para um objeto gráfico presente no nível 1, retorna a largura (nível 2) e a altura (nível 1) em pixels.

ANIMATE

Toma a partir do nível 2 até n+1 uma sequência de grobs e do nível 1:

- o número de grobs (n)
- uma lista contendo 4 ítens:

a) o número de grobs.

b) uma lista contendo as coordenadas de pixel (#nx #ny) do canto superior esquerdo da região onde a animação será feita.

c) o delay (em segundos) entre cada quadro da animação.

d) o número de vezes que a seqüência de animação será repetida (0 para repetir indefinidamente, até que uma tecla seja apertada).

ANALISE DE TÉCNICAS DE PROGRAMAÇÃO

O objetivo deste módulo é fornecer idéias para a construção de programas corretos, inteligíveis e de fácil manutenção. Para isto o primeiro passo estudado será a decomposição do problema e criação de um algorítimo. A partir destes quesitos busca-se um método que forneça:

- Programas corretos.
- Programas bem estruturados.
- Programas manuteníveis.
- Controle do programa desde o projeto..

Para programas de grande porte seriam necessários ainda documentação, comentários e controle da qualidade do programa. No entanto, os programas desenvolvidos para a HP48 são de pequeno porte, médio porte ocasionalmente para a HP48GX, portanto os ítens acima são suficientes para a obtenção de um bom programa.

são utilizados procedimentos básicos de programação na confecção de pequenos programas, tais como:

- Visualização do problema.
- Criação do algorítimo.
- Criação do fluxograma.
- Criação do programa.

Para programas maiores e mais complexos existem vários métodos de programação, mas usaremos o método de modularização devido a sua alta eficiência e ao fato de trabalharmos com programação estruturada. O método de modularização segue os seguintes passos:

- Visualização do problema.
- Criação do algorítimo.
- Divisão do problema em módulos.
- Criação de um fluxograma modular.
- Implementação e teste de cada módulo.
- Implementação do programa principal.

A modularização está englobada no que chamamos de programação estruturada, que é associada a vários conceitos:

- Programação sem o uso de “GO TO”.
- Uso exclusivo de concatenação (seqüência), selação (“if then else”), repetição (“while do”).
- Desenvolvimento de programas do tipo “TOP-DOWN” (refinamento sucessivo).

No entanto, nenhum dos ítens acima , e nem o conjunto, fornecem a idéia exata do que vem a ser programação estruturada. Então vamos definí-la:

- Programação estruturada é aquela na qual são satisfeitas as acertivas pré-determinadas, construindo o programa com um fluxo lógico que seja consistente dentro de cada trecho do programa (módulos).

Vamos analisar agora alguns programas interessantes, buscando fixar e desenvolver o uso dos comandos de programação. Será proveitoso digitá-los e executá-los no modo DEGUG (passo-a-passo). Os programas abaixo foram transferidos da HP para o computador pelo kermit utilizando translate code igual a 3.

1-Calculo da Determinante de uma Matriz qualquer.

```
%%HP: T(3)A(D)F(.);
\<<
  IFERR
    WHILE
“CALCULO DO DET” {
“MATRIZ” } { 1 2 }
{ } { } { } INFORM
    REPEAT EVAL DUP
SIZE EVAL SWAP 1 -
\-> m c l
      \<< 0 1 c
        FOR x 1 0 l
          FOR y m y
1 + x y +
            IF DUP
4 / DUP 1 >
            THEN IP
c * -
            ELSE
DROP
            END 2
\->LIST GET *
          NEXT +
        NEXT 0 1 c
        FOR x 1 0 l
          FOR y m y
1 + x y -
            IF DUP
1 <
            THEN
DUP c / IP 1 + c *
+
            END 2
\->LIST GET *
          NEXT +
        NEXT -
“DET:” SWAP +
MSGBOX m
      \>>
    END
  THEN CLEAR ERRN
DOERR
  END
\>>
```

2-Agenda de Telefones e Aniversários.

```
%%HP: T(3)A(D)F(.);

DIR

  AGEN
    \<< CLLCD BIB
"      AGENDA A.J.M"
{ "    PROCURAR"
"    INCLUIR"
"    EXCLUIR" } 1
CHOOSE 0 ==
      IF
      THEN UPDIR
KILL
      END DUP
"    PROCURAR" ==
      IF
      THEN CLEAR S
KILL
      END
"    INCLUIR" ==
      IF
      THEN CLEAR E
KILL
      END CLEAR D
    \>>

  BIB
    DIR


      D
       \<< CLLCD
FONE
"       EXCLUSÃO"
VARS DUP SIZE 0 ==
         IF
         THEN
CLEAR
"NÃO HÁ
CADASTRO"
MSGBOX UPDIR UPDIR
KILL
         END 1
         \<< "" +
         \>> DOSUBS
1 CHOOSE 0 ==
```

```
        IF
        THEN
UPDIR UPDIR KILL
        END UPDIR
OBJ\-> DUP FONE PURGE
UPDIR ANIV PURGE
UPDIR UPDIR
      \>>
```

S

```
      \<< CLLCD
"  PROCURA (\Ga + INICIAL)"
FONE VARS DUP SIZE
0 ==
        IF
        THEN
UPDIR UPDIR
"N\195O HÁ
CADASTRO."
MSGBOX CLEAR KILL
        END 1
        \<< "" +
        \>> DOSUBS
1 CHOOSE 0 ==
        IF
        THEN
UPDIR UPDIR KILL
        END DUP
DUP "  " SWAP +
":
 F: " + SWAP
OBJ\-> EVAL DUP NOVAL
\=/
        IF
        THEN +
        ELSE DROP
        END
"
 A: " + ANIV SWAP
OBJ\-> DUP NOVAL \=/
        IF
        THEN +
        ELSE DROP
        END
MSGBOX UPDIR UPDIR
      \>>
```

```
    E
      \<< ANIV
“INCLUS\195O (FONE,ANIV\->STRING)”
{ “NOME:” “FONE:”
“ANIV:” } { 1 3 } {
} { } INFORM 0 ==
        IF
        THEN
UPDIR UPDIR KILL
        END OBJ\->
DROP ROT DUP “” +
DUP “E” == SWAP DUP
```

```
“S” == SWAP DUP “D”
== SWAP DUP “ANIV”
== SWAP “FONE” ==
OR OR OR OR 1 ==
        IF
        THEN
CLLCD CLEAR
“   VARIÁVEL
  RESERVADA.”
UPDIR UPDIR MSGBOX
KILL
        END DUP
EVAL TYPE 6 \=/
        IF
        THEN
CLEAR UPDIR UPDIR
“ NOME JÁ USADO
 OU INVÁLIDO.
 (USE VARIÁVEL)”
MSGBOX KILL
        END DUP
ROT SWAP STO VARS
SORT ORDER UPDIR
FONE STO VARS CLLCD
“       A.J.M.” 3
DISP
“      ORDENANDO...”
5 DISP SORT ORDER
UPDIR UPDIR
       \>>
```

FONE
DIR

CEFET
2247735

LUCIANO
“2626133R33”

END

ANIV
DIR

CEFET
NOVAL

```
      LUCIANO
        NOVAL

      END

   END
END
```

COMUNICAÇÃO SERIAL

PARÂMETROS

Antes de iniciarmos a transferência de dados é preciso ajustar os parâmetros de comunicação serial da calculadora:

PORT: seleciona o modo de comunicação (*Wire para HP-PC* ou *Infrared para HP-HP*).
TYPE: seleciona o protocolo de comunicação (*Kermit* ou *XModem*).
FMT: seleciona o formato dos dados (*ASCII* ou *Binary*).
XLAT: seleciona o tipo de tradução de caracteres usado no formato ASCII.
CHK: seleciona o metodo de verificação de erros (checksum) usado na transferência.
BAUD: seleciona a velocidade de transferência de dados (1200 2400 4800 ou 9600).
PARITY: seleciona o tipo de método para gerar a paridade durante a comunicação.
OVRW: habilita ou não a gavação (overwrite) de variáveis já existentes.

COMUNICAÇÃO HP-HP

Podemos transferir qualquer tipo de objeto entre duas calculadoras HP através da comunicação por infravermelho. Para a estabelecer a transferência de programas entre duas calculadoras seguimos os seguintes passos:

1- Entre no menu de comunicação I/O (pressione [→] [I/O])
2- Selecione [Transfer...]
3- Configure os parâmetros do mesmo modo nas duas calculadoras:
(imprescindível: port=IR)
(recomendado: type=Kermit fmt=Bin Xlat=None chk=1)
4- Pressione [NEXT] [OK]
5- Entre novamente no menu de comunicação I/O (pressione [→] [I/O])
6- Selecione [Send to HP 48...] na calculadora que irá mandar os programas.
7- Pressione [CHOOS] e selecione os programas que serão enviados pressionando a tecla [_chk], caso o programa esteja em outro diretório pressione novamente [CHOOS] e selecione o novo diretório.
8- Antes de mandar o programa pressionando [SEND] é necessário entrar em [Get from HP 48] na calculadora que irá receber.
9- Finalmente podemos mandar o programa pressionando [SEND]

COMUNICAÇÃO HP-PC

A transferência entre a calculadora e o computador é praticamente igual ao processo descrito acima, com algumas modificações:

1- Entre no menu de comunicação I/O (pressione [→] [I/O])
2- Selecione [Transfer...]
3- Configure os parâmetros do mesmo modo nas duas calculadoras:
(imprescindível port=Wire)
(recomendado type=Kermit fmt=Bin Xlat=None chk=1)
(recomendado baud=9600 parity=None)
4- Pressione [NEXT] [OK]
5- Entre novamente no menu de comunicação I/O (pressione [→] [I/O])
6- Selecione [Send to HP 48...] no caso da calculadora ir mandar os programas.
7- Pressione [CHOOS] e selecione os programas que serão enviados pressionando a tecla [_chk], caso o programa esteja em outro diretório pressione novamente [CHOOS] e selecione o novo diretório.
8- Antes de mandar o programa pressionando [SEND] é necessário preparar o computador para receber.
9- Finalmente podemos mandar o programa pressionando [SEND]
10- No caso em que a calculadora deve receber o programa, deve-se entrar em [Get from HP 48] e através do computador mandar o programa.

KERMIT

Para utilizar a transferência de programas do computador para a calculadora é necessário conhecer os seguintes comandos do Kermit:

help ou h - mostra o help do programa.

ta porta1 - roda um programa de inicialização automática da porta 1
ta porta2 - roda um programa de inicialização automática da porta 2

set por 1 - utiliza a porta serial 1 para a transferência.
set bau 9600 - configura a velocidade de transmissão.

send [programa] - manda o [programa] para a calculadora.
recv - prepara o computador para receber um programa.

* Para configurar a serial:
ta porta1

* Para mandar:
send [nome do programa]

* Para receber:
recv

COMANDOS DE TRANSFÊRENCIA DE DADOS:

Para escrevermos programas que realizem uma comunicação serial (com o protocolo Kermit ou Xmode) precisamos conhecer os seguintes comandos:

SEND: transfere os objetos para serial cujos nomes estão no stack em forma de lista.
RECV: recebe dados via comunicação serial.
SERVER: entra no modo de servidor.
KGET: recebe um objeto cujo nome é enviado em forma de string.
FINISH: envia uma mensagem de término de transferência de dados ou do modo server.
RECN: recebe um objeto mudando o seu nome para outro que esta indicado no stack.
PKT: envia um comando via serial em forma de string.
KERRM: obtem o código do último erro ocorrido durante a transmissão de dados.

COMANDOS DE COMUNICAÇÃO SERIAL:

Algumas vezes precisamos receber informações de um circuito eletrônico ou de algum outro tipo de dispositivo que não permite a comunicação bidirecional, ou seja a calculadora ou só transmite dados ou só recebe dados. Por isso devemos conhecer os seguinte comandos de comunição serial que nos fornecem um controle maior sobre a serial da calculadora.

OPENIO: “abre” a porta serial, ou seja habilita a recepção de dados e a sua armazenagem num buffer de no máximo 255 caracteres.
CLOSEIO: “fecha” a porta serial , ou seja desabilita a recepção de dados limpando o buffer.
XMIT: envia para a serial da calculadora uma string. (transmissão de dados)
SRECV: obtém uma string de um tamanho especificado do buffer. (recepção de dados)
STIME: especifica o tempo de intervalo entre as tentativas de transferência de dados.
SBRK: envia para a serial um sinal de Break (parada)
BUFLEN: obtém a quantidade de caracteres recebidos e guardados no buffer de comunicação serial e um numero que indica se ocorreu algum tipo de erro durante a recepção dos dados. (0 se ocorreu e 1 se nada ocorreu de errado).

Legendas

legenda	*Descrição*	*legenda*	*Descrição*
[array]	Vetor ou matriz real ou complexa.	PICT	Objeto gráfico atual.
date	Data na forma MM.DDAAAA ou DD.MMAAAA.	point	Ponto {#n #m} de tela ou (x,y) de plot eq.
{dim}	Lista de dimensão da formação.	lc.p	linha, coluna e plano de tecla.
'nome'	Nome global.	(r,∅)	Número complexo na forma polar.
grob	Objeto gráfico.	"string"	Cadeia de caracteres.
hms	Formato horas minutos segundos (H.MMSSs).	'symb'	Expressão, equação ou nome tratado como algebrico.
index	Número real especificando um elemento de um obj.	V/F	Resultado de um teste (verdadeiro ou falso).
LID	Número de biblioteca.	time	Tempo na forma HH.MMSSs.
{list}	Lista de objetos.	[vector]	Vetor real ou complexo.
m n	Número real ou inteiro positivo.	x y	Número real.
[matriz]	Matriz real ou complexa.	x_unit	Objeto de unidade.
#n	Inteiro binário.	(x,y)	Número complexo na forma retangular.
obj	Qualquer objeto.	z	Número real ou complexo.

Guia de Referência dos Comandos

Comando	*Descrição*	*Entrada*		*Saída*
ABS	Valor Absoluto.	z	⇨	|z|
ACK	Reconhecimento de alarme vencido no display.			
ACKALL	Reconhecimento de todos os alarmes passados no display.			
ACOS	Arco-cosseno.	z	⇨	acos(z)
ACOSH	Arco-cosseno hiperbólico.	z	⇨	acosh(z)
ALOG	Antilogarítimo comum (base 10).	z	⇨	10^z
AND	AND lógico ou binário.	#n1 #n2	⇨	#n1 AND #n2
APPLY sem avaliar.	Devolve expressões avaliadas como argumentos a nomes locais 'nome' {symb ...}	⇨		
ARC raio, e ângulo de início e fim.	Desenha um arco em PICT com um ponto central especificado,	point xr x1 x2	⇨	
ARCHIVE	Faz uma cópia de segurança do diretório HOME	:n: name	⇨	
ARG	Devolve o ângulo polar O.	z	⇨	x∅
ARRY→	Separa uma formação	[array]	⇨	z1 ... zn {dim}
→ARRY	Cria uma formação	z1 ... zn {dim}	⇨	[array]
ASIN	Arcosseno.	z	⇨	asin(z)
ASINH	Arcosseno Hiperbólico.	z	⇨	asinh(z)
ASN	Liga uma objeto a uma tecla do modo de usuário.	obj lc.p	⇨	
ASR	Rotação aritmética para direita de 1 bit.	#n1	⇨	#n2
ATAN	Arcotangente.	z	⇨	atan(z)

ATANH	Arcotangente hiperbólica.	z	⇨	atanh(z)
ATTACH	Liga uma biblioteca ao diretório atual	LID	⇨	
AUTO	Auto-escala o eixo y			
AXES	Fixa as coordenadas de intersecção dos eixos.	point	⇨	
BAR	Seleciona o traçado BAR.	point	⇨	
BARPLOT	Plota o diagrama de barra dos dados de ΣDAT.			
BAUD	Fixa a taxa de transferência.	nbaud	⇨	
BEEP	Emite um beep	nfreq nduração	⇨	
BESTFIT	Seleciona o modo de traçado que produz o maior valor absoluto do coeficiente de correlação LR.			
BIN	Fixa a base binária.			
BINS	Ordena os elementos na coluna de variáveis independentes de ΣDAT em n+2 casas.	xmin xcolunas nbins	⇨	[[b1]...[bn]][b<b>]
BLANK	Cria um objeto gráfico vazio.	#ncolunas #nlinhas	⇨	grob
BOX	Desenha um quadrado a partir de dois cantos opostos.	point1 point2	⇨	
BUFLEN	Devolve o número de caracteres no buffer da serial.		⇨	n
BYTES	Devolve o checksum e o tamanho em bytes de um objeto.	obj	⇨	#nchecksum xtam
B→R	Converte binário para real.	#n	⇨	n
CASE	Começa a estrutura CASE... THEN ... END ... END			
CEIL	Devolve o número inteiro seguinte.	x	⇨	n
CENTR	Fixa o centro da tela do plot no ponto.	point	⇨	
CF	Desativa um flag específico.	m	⇨	
%CH	Calcula o variação percentual.	xinicial yfinal	⇨	xΔ%
CHR	Converte o código de um caracter em uma cadeia de um caracter.	n	⇨	“string”
CKSM	Fixa o tipo de checksum de E/S.	nchecksum	⇨	
CLEAR	Apaga todos os dados no stack.	obj1...objn	⇨	
CLKADJ	Ajusta o clock da calculadora em ticks. (1/8192 seg)	nmarcas	⇨	
CLLCD	Limpa a tela.			
CLOSEIO	Fecha a porta serial.			
CLUSR	Igual a CLVAR			
CLVAR	Apaga todas as variáveis de usuário no diretório atual.			
CLΣ	Apaga ΣDAT.			
CNRM	Calcula a norma da coluna de uma matriz.	[array]	⇨	xnormacoluna
COLCT	Une termos semelhante de uma expressão.	‘symb’1	⇨	‘symb’2
COLΣ	Seleciona as colunas de estatística independente e dependentes	nind ndepen	⇨	

COMB	Calcula as combinações de n elementos tomados de m em n	n m	⇨	Cn,m
CON	Cria uma matriz constante.	{dim} z	⇨	[array]
CONIC	Seleciona o modo de traçado cônico.			
CONJ	Devolve o conjugado de um complexo.	[array]	⇨	[array*]
CONT	Continua a execução de um programa.			
CONVERT	Executa conversão de unidades.	x1_unit1 x2_unit2	⇨	x3_unit2
CORR	Calcula o coeficiente de correlação.		⇨	xcorrelacion
COS	Cosseno.	z	⇨	cos(z)
COSH	Cosseno hiperbólico.	z	⇨	cosh(z)
COV	Calcula a variança.		⇨	xcovariança
CR	Retorno do carro/salto de linha.			
CRDIR	Cria um diretório.	'nome'	⇨	
CROSS	Produto vetorial de vetores.	[vetor1] [vetor2]	⇨	[vetor3]
C→PX	Conversão de unidades do usuário a coordenadas na tela.	(x,y)	⇨	{#n #m}
C→R	Conversão de número complexo em real.	(x,y)	⇨	x y
DATE	Devolve a data corrente.		⇨	date
DATE+	Soma a uma data um determinado número de dias.	date ndias	⇨	date2
→DATE	Ajusta a data corrente.	date	⇨	
DDAYS	Número de dias entre duas datas.	date1 date2	⇨	ndias
DEC	Fixa o modo decimal.			
DECR	Decrementa e devolve o valor de uma variável especificada.	'nome'	⇨	x
DEFINE	Cria uma variável ou função definida pelo usuário.	'symb'	⇨	
DEG	Fixa o modo de graus.			
DELALARM	Apaga um alarme da lista de alarmes do sistema.	nalarme	⇨	
DELAY	Fixa o tempo entre as linhas impressas.	x	⇨	
DELKEYS	Desativa as ligações de programas às teclas do modo usuário.	lc.p	⇨	
DEPND	Especifica o nome da variável dependente para o plot.	'nome'	⇨	
DEPTH	Devolve o número de objetos do stack.		⇨	n
DET	Determinante de uma matriz.	[matriz]	⇨	xdet
DETACH	Desconecta uma biblioteca do diretório atual.	LID	⇨	
DISP	Mostra o objeto na linha especificada.	obj nlinha	⇨	
DO	Começa a estrutura DO...UNTIL...END.			
DOERR	Aborta a execução de um programa, mostra a msg de erro.	"string"	⇨	

DOT	Produto escalar de dois vetores.	[array] [array]	⇨	x
DRAW	Traçado da função em EQ.			
DRAX	Traçado dos eixos.			
DROP	Elimina o objeto do nível 1 do stack.	obj	⇨	
DROPN	Elimina n objetos do stack.	obj1...objn n	⇨	
DROP2	Elimina dois objetos do stack.	obj1 obj2	⇨	
DTAG	Apaga todas as etiquetas do objeto.	:tag:obj	⇨	obj
DUP	Duplica o objeto do nível 1 do stack.	obj	⇨	obj obj
DUPN	Duplica n objetos do stack.	obj1...objn n	⇨	obj1...objn obj1...objn
DUP2	Duplica os objetos dos níveis 1 e 2 do stack.	obj1 obj2	⇨	obj1 obj2 obj1 obj2
D→R	Conversão de graus em radianos.	xgraus	⇨	xrad
e	Constante simbólica e (2.71828182846).			
ELSE	Começa a estrutura ELSE.			
END	Acaba estruturas de programas.			
ENG	Fixa o modo de engenharia.	ncasas	⇨	
EQ→	Separa equações nos lados direito e esquerdo.	‘symb1=symb2’	⇨	‘symb1’ ‘symb2’
ERASE	Apaga PICT.			
ERRM	Retorna a última mensagem de erro.		⇨	“mensagem de erro”
ERRN	Devolve o número do último erro.		⇨	#n
ERRO	Apaga o útimo número da mensagem de erro.			
EVAL	Avalia um objeto	obj	⇨	. . .
EXP	Eleva a potência exponencial o objeto do nível 1.	z	⇨	e^z
EXPAN	Expande o objeto algébrico.	‘symb1’	⇨	‘symb2’
EXPFIT	Fixa o modelo de regressão para a curva exponencial.			
EXPM	Exponencial naturalo menos 1.	x	⇨	e^x-1
FC?	Comprova se o flag especificado está desativado.	mflag	⇨	T/F
FC?C	Comprova se o flag está desativado, se não o ativa.	mflag	⇨	T/F
FINDALARM	Devolve o alarme da hora especificada.	date	⇨	nalarm
FINISH	Termina com o modo server do Kermit.			
FIX	Seleciona o modo FIX.	ncasas	⇨	
FLOOR	Inteiro seguinte mais pequeno.	x	⇨	n
FOR	Começa a estrutura FOR ... NEXT ou FOR ... STEP			
FP	Devolve a parte fracionária de um número.	x	⇨	FP(x)
FREE	Libera a memória ligada.	{ name/LID } nporta	⇨	T/F

FREEZE	Congela uma determinada área do display.	n	⇨	
FS?	Comprova se o flag especificado está ativo.	mflag	⇨	T/F
FS?C	Comprova se o flag especificado está ativo, se não o ativa.	mflag	⇨	T/F
FUNCTION	Seleciona o traçado de gráficos no modo FUNCTION.			
GET	Obtem elementos de uma formação ou lista.	[array] index	⇨	z
GETI	Obtem elementos de uma formação ou lista e incremente o índice.	[array] index1	⇨	[array] index2 z
GOR	Sobrepõe o gráfico do nível 1 sobre o do nível 3 no ponto especif.	grob1 point grob2	⇨	grob3
GRAD	Seleciona o moo radianos.		⇨	
GRAPH	Ativa o modo gráfico.		⇨	
→GROB	Converte uma cadeia de caracteres em um objeto gráfico.	“string” nsize	⇨	grob
GXOR	Usa um OR-exclusivo para sobrepor dois gráficos.	grob1 point grob2	⇨	grob3
HALT	Suspende a execução de um programa.		⇨	
HEX	Fixa o modo hexadecimal.		⇨	
HMS+	Soma em formato HMS.	hms1 hms2	⇨	hms3
HMS-	Subtração no formato HMS.	hms1 hms2	⇨	hms3
HMS→	Converte HMS para o formato decimal.	hms	⇨	x
→HMS	Converte um número decimal para o formato hms.	x	⇨	hms
HOME	Seleciona o diretório HOME.		⇨	
i	Constante simbólica i.		⇨	
IDN	Cria uma matriz identidade de um tamanho especificado.	n	⇨	[matriz]
IF	Começa a estrutura de decisão IF...THEN..[ELSE]...END		⇨	
IFERR	Começa a estrutura de erro IFERR...THEN..[ELSE]...END		⇨	
IFT	Comando IFT/THEN	T/F objverd	⇨	
IFTE	Começa a estrutura de decisão IFTE...THEN..[ELSE]...END	T/F objverd objfalso	⇨	
IM	Devolve a parte imaginária de um número.	(x,y)	⇨	y
INCR	Incrementa e devolve o valor da variável especificada.	‘nome’	⇨	x
INV	Inverso.	z	⇨	1/z
IP	Parte inteira de um número.	x	⇨	IP(x)
ISOL	Isola a variável especificada de um lado da equação.	‘symb1’ ‘nome’	⇨	‘nome=symb2’
KEY	Devolve um número indicando a última tecla pressionada.		⇨	0 ou lc.p 1
KILL	Aborta todos os programas suspensos na memória.		⇨	
LAST	Igual a LASTARG		⇨	
LASTARG	Devolve o último argumento da pilha.		⇨	
LCD→	Obtem a imagem atual do display em forma de tela gráfica.		⇨	grob

→LCD	Mostra um objeto gráfico.	grob	⇨	
LINE	Traça uma linha entre dois pontos.	point1 point2		
→LIST	Cria uma lista a partir dos objetos do stack.	obj1...objn n	⇨	{obj1...objn}
LIST→	Separa uma lista.	{obj1...objn}	⇨	obj1...objn n
LN	Logarítmo neperiano.	z	⇨	LN(z)
LNP1	Logarítmo neperiano de (x+1).	x	⇨	LN(x+1)
LOG	Logarítmo base 10.	z	⇨	LOG(z)
MAX	O maior número entre dois.	x y	⇨	MAX(x,y)
MEM	Memória disponível ao usuário.		⇨	x
MENU	Cria um menu de usuário.	{ obj... }	⇨	
NEG	Negativo do argumento do nível 1.	z	⇨	-z
NEWOB	Cria uma nova cópia do objeto.	obj	⇨	obj
NOT	NOT lógico ou binário.	#n1	⇨	NOT #n1
NUM	Devolve o código do primeiro caractere da string.	“string”	⇨	n
→NUM	Avalia objetos no modo numérico.	obj	⇨	
OBJ→	Devolve os componentes do objeto na pilha.		⇨	
OCT	Fixa a base octal.		⇨	
OFF	Desliga a calculadora.		⇨	
OR	OR lógico ou binário.	#n1 #n2	⇨	#n1 OR #n2
ORDER	Reordena os diretórios.	{ name .... }	⇨	
OVER	Duplica o objeto do nível 2.	obj1 obj2	⇨	obj1 obj2 obj1
PATH	Devolve o caminho do diretório atual.		⇨	{ HOME name ... }
PDIM	Muda o tamanho da tela gráfica.	#nlargura #naltura	⇨	
PERM	Permutação de n objetos tomados de m em m.	n m	⇨	P n,m
PGDIR	Elimina um diretório.	‘nome’	⇨	
PICK	Copia o objeto do nível n para o nível 1.	obj1...objn n	⇨	obj1...objn obj1
PICT	Devolve a PICT para o nível 1.		⇨	PICT
PIXOFF	Apaga um determinado ponto da tela gráfica.	point	⇨	
PIXON	Acende um determinado ponto da tela gráfica.	point	⇨	
PIX?	Verifica se um determinado ponto da tela gráfica está aceso.	point	⇨	T/F
POS	Devolve a posição de um objeto em uma lista ou string.	{obj ....} objproc	⇨	npos
PURGE	Apaga um ou mais variáveis.	‘nome’	⇨	
PUT	Insere em uma lista ou formação um determinado objeto.	[array1] indez z	⇨	[array2]
PUTI	Igual a PUT, porém incrementa o índice.	[array1] indez z	⇨	[array2] indez2

PVIEW	Mostra a tela gráfica a partir de um determinado ponto.	point	⇨	
PX→C	Conversão de unidades de pontos para unidades de usuário.	{ #n #m }	⇨	(x,y)
→Q	Converte números a suas frações equivalentes.	x	⇨	'symb'
→Qπ	Compara coecientes e tenta por o número sob forma de pi*a/b	x	⇨	'symb'
RAD	Fixa o mode radianos		⇨	
RAND	Devolve um número aleatório 0 < x < 1.		⇨	x
RCL	Devolve o conteúdo de uma variável para a pilha.	'nome'	⇨	obj
RCLALARM	Devolve um alarme da lista de alarmes do sistema.	nalarm	⇨	{ alarm }
RCLF	Devolve o estado dos flags do sistema e do usuário.		⇨	{ #system #user }
RCLKEYS	Devolve a lista da ligações das teclas de usuário.		⇨	{obj lc.p... objn lc.pn}
RCLMENU	Devolve a posição da página do menu atual.		⇨	x
RCWS	Obtém o tamanho do maior inteiro binário.		⇨	n
RDM	Redimensiona uma formação.	[array1] {dim}	⇨	[arry2]
RE	Devolve a parte real de um número	(x,y)	⇨	x
RECN	Recebe um programa via Kermit e o guarda na variável especific.	'nome'	⇨	
RECV	Recebe um programa via Kermit. (entre HP-HP ou PC-HP)		⇨	
REPL	Adiciona em uma lista, string, objeto gráfico ou PICT o objeto especificado no nível 1, na posição no nível 2.	obj1 objpos obj2	⇨	obj3
RESTORE	Restaura um Backup armazenado via Kermit.	:n: nome	⇨	
ROLL	Move o objeto do nível n para o nível 1.	obj1...objn n	⇨	obj1...objn obj1
ROLLD	Move o objeto do nível 1 para o nível n.	obj1...objn n	⇨	objn obj1...objn-1
ROT	Move o objeto do nível 3 para o nível 1.	obj1 obj2 obj3	⇨	obj2 obj3 obj1
R→B	Conversão de real para binário.	n	⇨	#n
R→C	Conversão de real para complexo.	x y	⇨	(x,y)
R→D	Conversão de radianos para graus.	xrad	⇨	xgraus
SAME	Testa a igualdade de dois objetos.	obj1 obj2	⇨	T/F
SCI	Fixa o modo científico.		⇨	
SCONJ	Conjuga o conteúdo de uma variável.	'nome'	⇨	
SEND	Envia um programa, um diretório, ou uma lista de programas e diretórios via Kermit. (entre HP-HP ou HP-PC) 'nome' ⇨	{ 'nome1' 'nome2' ... }		
SERVER	Seleciona o modo servidor via Kermit.		⇨	
SF	Habilita o flag indicado.	m	⇨	
SIGN	Devolve o sinal do número.	z1	⇨	z2
SIN	Calcula o seno de um número.	z	⇨	sin(z)

SINH	Calcula o seno hiperbólico de um número.	z	⇨	sinh(z)
SINV	Atualiza o conteúdo de uma variável com a sua inversa.	‘nome’	⇨	
SIZE	Obtem o tamanho do objeto.	obj	⇨	n ou {dim}
SNEG	Nega o conteúdo de uma variável	‘nome’	⇨	
SQ	Eleva ao quadrado o objeto do nível 1.	z	⇨	z^2
START	Começa a estrutura START...NEXT ou START...STEP		⇨	
STD	Seleciona o modo standar.		⇨	
STEP	Termina o laço definido incrementando o contador em n.	n	⇨	
STEQ	Armazena a equação em ‘EQ’	‘symb’	⇨	
STO	Armazena um objeto em uma variável especificada.	obj ‘nome’	⇨	
STOALARM	Armazena um alarme na lista de alarmes do sistema.	time	⇨	nalarm
STOF	Restaura os flags do sistema e do usuário.	{ #system #user }	⇨	
STOKEYS	Faz múltiplas ligações de objetos a teclas de usuário.	{obj lc.p... objn lc.pn}	⇨	
STO+	Atualiza o valor da variável com o valor especificado (soma).	z ‘nome’	⇨	
STO-	Atualiza o valor da variável com o valor especificado (subtração).	z ‘nome’	⇨	
STO*	Atualiza o valor da variável com o valor especificado (multiplic).	z ‘nome’	⇨	
STO/	Atualiza o valor da variável com o valor especificado (divisão).	z ‘nome’	⇨	
STR→	Executa objetos contidos em cadeias.	“string”	⇨	
→STR	Converte o objeto do nível 1 em uma cadeia de caracteres.	obj	⇨	“string”
STWS	Fixa o tramanho das palavras em binário.	n	⇨	
SUB	Extrae parte de uma lista, cadeia, mastriz, obj gráficos ou PICT	obj1 ninicio nfim	⇨	obj2
SWAP	Troca de posição os objetos dos níveis 1 e 2.	obj1 obj2	⇨	obj2 obj1
SYSEVAL	Avalia um objeto do sistema interno da calculadora a partir de um endereço especificando o início do objeto external ou code.	#n	⇨	BUM
→TAG	Liga a um objeto do nível 2 uma etiqueta identificando o obj.	obj “tag”	⇨	:tag: obj
TAN	Tangente.	z	⇨	tan(z)
TANH	Tangente Hiperbólica.	z	⇨	tanh(z)
TICKS	Devolve a hora como inteiro binário em unidades de TICKS.		⇨	#ticks
TIME	Devolve a hora atual como um número no formato HMS.		⇨	time
→TIME	Atualiza o clock da calculadora.	time	⇨	
TLINE	Desenha uma linha entre dois pontos usando XOR.	point1 point2	⇨	
TMENU	Cria um menu de usuário temporário.	{ obj1 ... objn }	⇨	
TSTR	Converte a data e a hora para o formato de string: “date time”	data time	⇨	“data time”
TVARS	Devolve uma lista das variáveis de um determinado tipo.	ntipo	⇨	{ global ... }

TYPE	Devolve o número identificando o tipo do objeto.	obj	⇨	ntipo
UBASE	Converte o objeto de unidades em unidades básicas do SI.	x_unit	⇨	y_base-unts
UFACT	Fatora uma determinada unidade em um objeto de unidades.	x1_unit1 x2_unit2	⇨	x3_unit2*unit3
→UNIT	Pega a unidade do objeto do nível 2 e coloca no obj do nível 1.	x_units y	⇨	y_units
UNTIL	Começa a estrutura UNTIL.		⇨	
UPDIR	Sobe um diretório do nível atual.		⇨	
UVAL	Devolve a parte numérica de um objeto de unidade.	x_unts	⇨	x
VARS	Devolve a lista de variáveis do diretório atual.		⇨	{ global ...}
VTYPE	Devolve o número do tipo do obj armazenado na variável.	'nome'	⇨	ntipo
→V2	Cria um vetor a partir de dois números.	x1 x2	⇨	[x1 x2]
→V3	Cria um vetor a partir de três números.	x1 x2 x3	⇨	[x1 x2 x3]
V→	Separa um vetor em seus componentes.	[x1 x2]	⇨	x1 x2
WAIT	Detém a execução de um programa por n segundos.	n	⇨	
XOR	Ou-exclusivo lógico ou binário.	#n1 #n2	⇨	#n1 XOR #n2
!	Fatorial de um número.	n	⇨	n!

Index

S

V