1.0.1

CALCULADORA HP48G/GX
PROGRAMADOR - BÁSICO
(USERRPL)

PABLO DE AVILA SALDO

SUMÁRIO

O AMBIENTE

EMULADOR EMU48

Esta é uma tela de trabalho do Emu48[i], uma HP48G/GX perfeita[ii], com alguns toques a mais!!

Além do mouse, você pode usar o teclado para acionar o emulador. As teclas da calculadora que possuem uma letra têm suas correspondentes no teclado do PC, assim como . , SPC , DEL , ← e ENTER . Para as teclas + , − , × e ÷ devem ser utilizadas as correspondentes do teclado numérico. As setas direcionais do teclado funcionam também, correspondendo as teclas ◄P ▲K ▼Q ►R . As teclas restantes têm a seguinte correspondência:

Tecla da HP48G/GX	Tecla do PC
ON	Esc
↱	Caps Lock
↰	Tab
α	~ `

Os botões superiores do emulador também são de muita utilidade:

 Inicia uma nova sessão do Emu48

 Abre uma sessão

 Salva a sessão corrente

 Copia a tela

 Reseta a “calculadora”

 Configurações

 Troca o script

 Carrega um arquivo na pilha

 Salva em um arquivo externo

 Diminui o contraste

 Aumenta o contraste

 Tela de *Copyright*

Todos estes botões têm seus correspondentes nos menus.

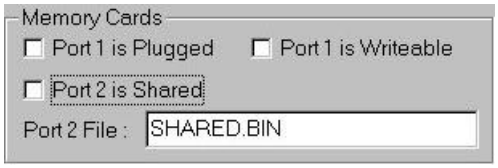
Os único submenus ausentes nos botões são “Save As... “ e “Backup”

Uma característica importante do emulador é a possibilidade da comunicação de dois emuladores, em computadores separados, ou entre um emulador e uma HP48G/GX!

Para isso, nas configurações do emulador é necessário indicar por qual porta serial de comunicação (COM), será feita a transmissão/recepção dos dados. A porta mais comum é a COM2, já que a COM1 geralmente é utilizada pelo mouse, e as portas COM3 ou COM4, pelo modem.

A comunicação entre emuladores é feita através de um cabo serial nulo, enquanto que a comunicação entre um emulador e uma HP48G/GX é feito através de um cabo especial.

O emulador permite também a simulação de cartões de memória, através de um programa externo chamado CREATE.EXE, que permite utilizarmos um cartão de até 4MB de memória (o sonho de todo usuário)! A porta 1 já é automaticamente disponibilizada com 128KB.

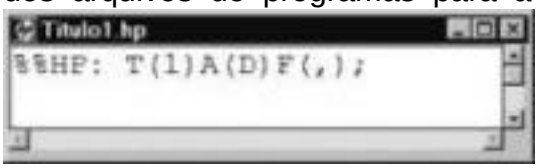


O emulador só trabalha com arquivos no formato binário, por isso é necessário, as vezes, converter-se um arquivo ASCII. Isso pode ser feito pelo programa ASCTOBIN (A2B), presente dentro do ambiente do emulador.

HP48 PAD

O HP PAD é um editor de textos especial para produzir textos que irão ser carregados na HP48G/GX. Nele podem ser confeccionados tanto texto, quanto programas. Seu funcionamento é semelhante ao dos editores de texto mais comuns, com opções para recortar, colar, desfazer, trabalho em múltiplas janelas, lista de arquivos recentes, etc. Seu diferencial está nas características que o fazem ideal para trabalhos de digitação para a HP48G/GX:

- Cria novos arquivos já com o cabeçalho dos arquivos de programas para a HP48G/GX, "%%HP: T(1)A(D)F(,);". Esse cabeçalho é necessário para a calculadora identificar o arquivo como um programa;



- Barra de ferramantas com caracteres especiais e lista de comandos da HP48G/GX. O problema de outros editores é que os símbolos da HP48G/GX não correspondem aos símbolos normais destes editores;

- Opção de abir e limitar arquivos a 22 colunas. Utilizado principalmente para textos Assim todo o texto digitado será visualisado na tela da HP48G/GX sem necessidade de rolagem;

O editor gera arquivos ASCII, o que permite a abertura destes arquivos em outros editores. O editor é em espanhol, o que ajuda para aqueles que não conhecem o inglês.

A LINGUAGEM USER RPL

USER RPL & SYS RPL

Você pode programar para as calculadoras HP48G/GX utilizando duas linguagens distintas: linguagem montadora (assembly) ou Reverse Polish LISP (RPL). A linguagem RPL pode ainda ser trabalhada de duas formas: Sys RPL e User RPL.

A linguagem Sys RPL é a linguagem utilizada junto com o assembly para construir a calculadora. Somente podemos programar nesta linguagem utilizando um compilador. É o compilador que transformará nosso código-fonte em algo compreensível para a calculadora, e uma vez compilado, o código não pode mais ser alterado (semelhante a um programa feito em Pascal ou C).

Já a linguagem User RPL (UserRPL) está disponível diretamente nas calculadoras HP48G/GX e é com ela que iremos trabalhar neste curso. A UserRPL nada mais é do que uma linguagem *scripting* (de roteiro), onde podemos armazenar sequências de comandos que utilizamos normalmente, com algumas facilidades adicionais, tais como uso de variáveis locais e estruturas de controle. A linguagem é interpretada, isto é, o código-fonte fica disponível em uma forma compreensível para o programador, e é analisado cada vez que o programa é executado, o que permite que façamos alterações no código na própria pilha da calculadora e executemos o programa imediatamente.

Você provavelmente já se deparou com um programa em UserRPL, ou talvez já tenha até construído um. Um programa em UserRPL é um objeto que pode ser armazenado na calculadora e é delimitado pelos caracteres « e ».

DEPURANDO SEUS PROGRAMAS

A fim de que você possa analisar e testar seus programas, a HP48G/GX provê uma série de ferramentas utilitárias, que permitem o acompanhamento passo a passo da execução do programa, localizadas em PRG NXT

DBUG

Armazena o programa do nível 1 da pilha na memória, para depuração.

SST

Avança o programa armazenado na memória passo a passo, considerando subrotinas como um passo único.

SST↓

Avança o programa armazenado na memória passo a passo, executando rotinas também passo a passo.

NEXT

Mostra o próximo passo (ou até os dois próximos passos), sem executá-los

HALT

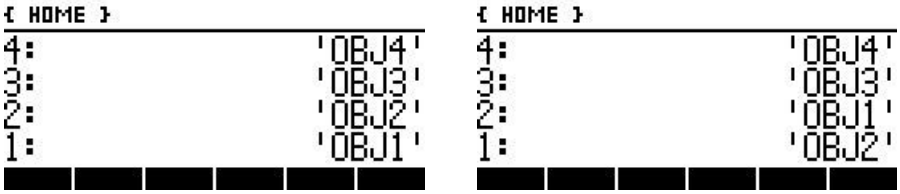
Colocado em um trecho do programa, para a execução, que pode ser continuada com SST ou SST↓.

KILL

Desliga o anúncio HALT, cancelando a execução do programa.

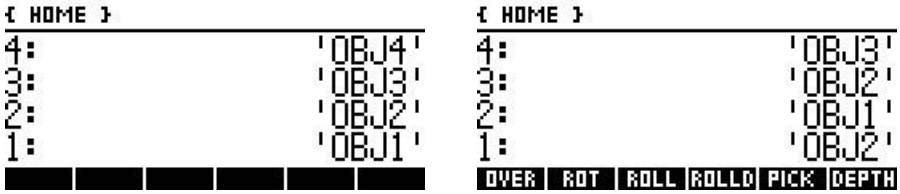
MANIPULAÇÃO DA PILHA (STACK)

SWAP



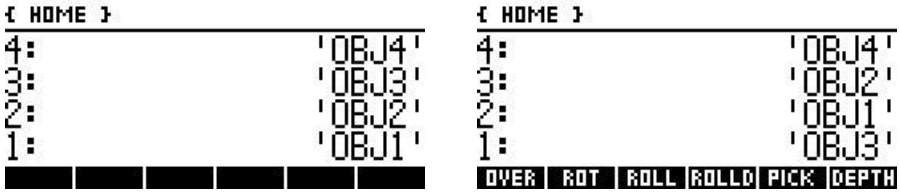
Troca de posição os objetos dos niveis 1 e 2 da pilha.

OVER



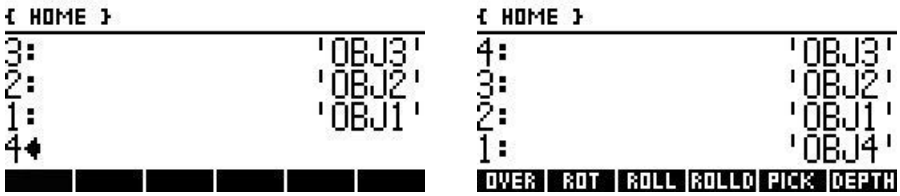
Retorna uma cópia do objeto do nível 2.

ROT



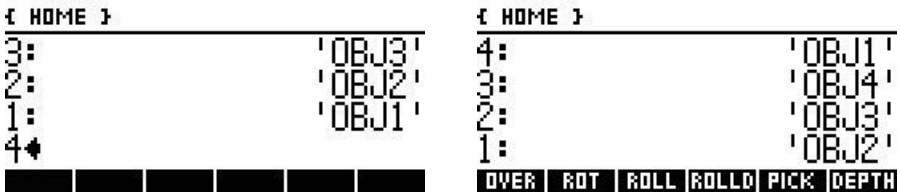
Rotaciona os 3 primeiros objetos, movendo o objeto do nível 3 para o nível 1.

ROLL



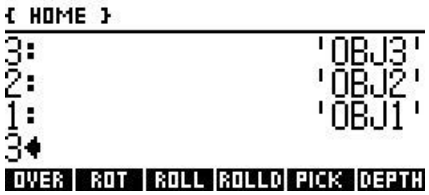
Move o objeto de um nível especificado para o nível 1.

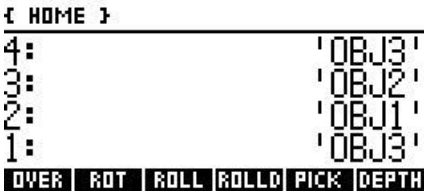
ROLLD



Move o objeto do nível 1 para um nível especificado.

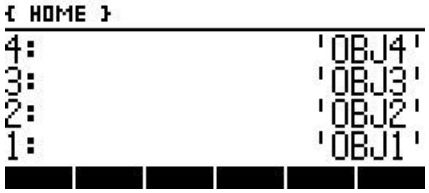
PICK

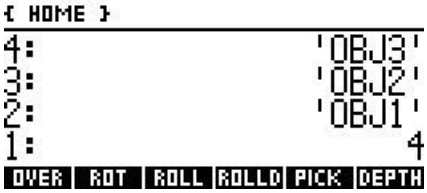




Retorna a cópia do objeto de um nível especificado.

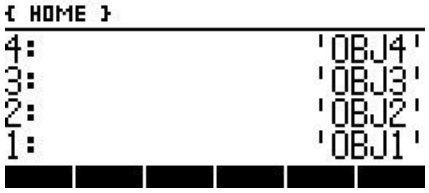
DEPTH

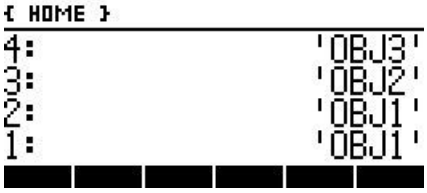




Retorna o tamanho (profundidade) da pilha.

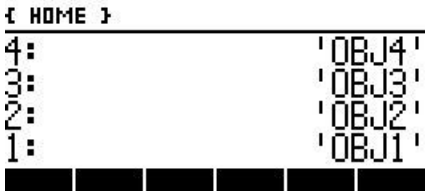
DUP





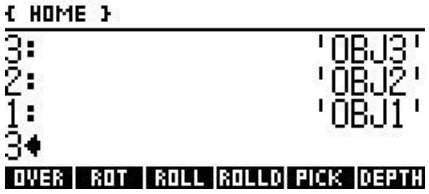
Retorna uma cópia do elemento do nível 1.

DUP2



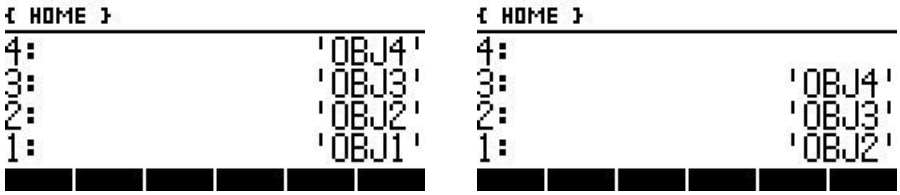
Retrorna a cópia dos 2 primeiros objetos da pilha.

DUPN



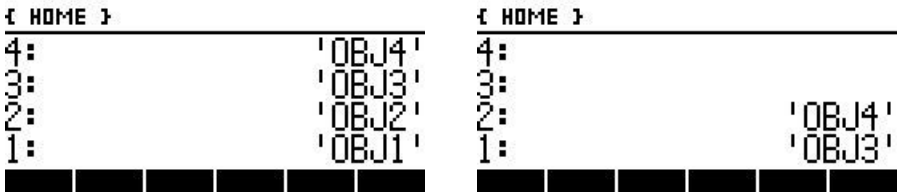
Retorna a cópia dos primeiros “n” objetos da pilha.

DROP



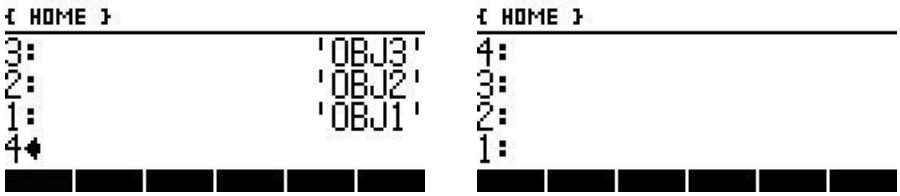
Remove o primeiro elemento da pilha.

DROP2



Remove os 2 primeiros elementos da pilha.

DRPN



Remove os “n” primeiros elementos da pilha.

VARIÁVEIS LOCAIS

Os programas que fizemos até agora não necessitaram de variáveis, assim como não necessitaram de entradas do usuário. Mas a maioria, para não dizer a totalidade dos programas que desenvolvemos, necessita de entradas do usuário, que no decorrer da execução do programa serão utilizadas, alteradas e descartadas. Para um pequeno número de variáveis fica fácil controlá-las, mas quando esse número começa a aumentar, a coisa começa a piorar.

Poderíamos armazená-las em variáveis com o comando STO e utilizá-las com RCL, STO+, STO-, etc., e apagá-las ao fim do programa usando PURGE. Mas e se o nome da variável que iremos utilizar já existir, ou se o programa terminar com um erro. No primeiro caso, perderíamos informações importantes, enquanto que no segundo ficaríamos com variáveis armazenadas que não fazem nenhum sentido para nós.

A solução do nosso problema é a utilização de variáveis locais dentro de nossos programas, ou apenas em trechos deles. Uma variável local pode ter qualquer nome válido de variáveis e só é visível dentro do bloco de programa exatamente após a sua declaração. A declaração se dá utilizando o operador →:

« → var1 var2 « *bloco local* » *fim do bloco* »

As variáveis var1 e var2 só são visíveis dentro do bloco local. Toda informação sobre elas deve ser deixada na pilha, se quisermos utilizá-las no fim do bloco. O valor das variáveis é retirado da pilha. No exemplo acima se tivéssemos uma pilha

```
3: 3
2: 2        (var1)
1: 1        (var2)
```

var1 seria inicializada com o valor 2, e var2 com 1.

Programas que utilizam variáveis locais são mais simples e mais fáceis de depurar, porém são mais lentos do que programas que manipulam a pilha diretamente.

O ideal é mesclarmos em um programa trechos que manipulam a pilha com trechos que utilizam variáveis locais.

ESTRUTURAS DE REPETICAO

START ... NEXT

« *início fim* START *comandos* NEXT »

A estrutura recebe dois números da pilha, e executa *comandos* até que o valor recebido como início seja igual ao valor recebido como fim. A cada execução da série de comandos, o valor de início é incrementado.

```
« → n
 « 1 10 START
    n 1 'n' STO+
   NEXT
 »
»
```

O programa acima tem como saída n, n+1, n+2, ..., n+9.

FOR ... NEXT

« *início fim* FOR *var comandos* NEXT »

A estrutura recebe dois números da pilha, inicializa a variável *var* e executa *comandos* até que *var* seja igual ao valor recebido como *fim*. A cada execução da série de comandos, *var* é incrementada.

A principal diferença entre START ... NEXT e FOR ... NEXT é que, usando FOR, temos acesso ao valor do contador:

```
« → n
 « n DUP 9 + FOR n
    n
   NEXT
 »
»
```

O programa acima tem a mesma saída do programa anterior.

É possível também mudar o incremento do laço, utilizando STEP ao invés de NEXT em qualquer um das estruturas acima:

```
« → n
 « 10 1 START
    n 1 'n' STO-
   -1 STEP
 »
»
```

Esse programa devolve a seqüência n, n-1, n-2, ..., n-9.

```
« → n
 « n 9 + n FOR n
    n
   -1 STEP
 »
»
```

O programas devolve a seqüência n+9, n+8, n+7, ..., n.

DO ... UNTIL ... END

« DO *comandos* UNTIL *condição* END »

Nesta estrutura, *comandos* é executado pelo menos uma vez, e seguirá sendo executado até que *condição* seja verdadeiro.

O programa abaixo devolve a soma dos números pares de 100 a 200:

```
«0 100 → soma n
 « DO 'soma' n STO+
    2 'n' STO+
   UNTIL
    n 200
   END
   soma
 »
»
```

WHILE ... REPEAT ... END

« WHILE *teste* REPEAT *comandos* END »

Nesta estrutura, *comandos* só é executado se *teste* for verdadeiro, e repetirá enquanto assim o for.

O programa abaixo é equivalente ao exemplo da estrutura DO ... UNTIL ... END:

```
«0 100 → soma n
 « WHILE n 200 <
   REPEAT
    'soma' n STO+
    2 'n' STO+
   END
   soma
 »
»
```

ESTRUTURAS CONDICIONAIS

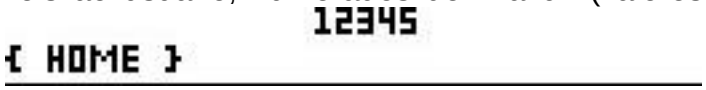
TESTES

A calculadora HP48G/GX não provê tipos lógicos, utilizando os números reais. Qualquer valor diferente de 0 é considerado como verdadeiro. Existem quatro operações básicas com valores lógicos: NOT, AND, OR e XOR. As tabelas-verdade estão demonstradas abaixo:

NOT		AND			OR			XOR		
A	Saída	A	B	Saída	A	B	Saída	A	B	Saída
F	V	F	F	F	F	F	F	F	F	F
V	F	F	V	F	F	V	V	F	V	V
		V	F	F	V	F	V	V	F	V
		V	V	V	V	V	V	V	V	F

Existem ainda sete funções lógicas: == (igual), SAME (igual, mais rápido), ≠ (diferente), < (menor), > (maior), ≤ (menor ou igual) e ≥ (maior ou igual).

Temos ainda os *flags*, registradores especiais, utilizados como marcadores pelo usuário. São 64 *flags* disponíveis ao usuário, numerados de 1 a 64 (valores negativos representam *flags* do sistema), e os primeiro cinco ficam sinalizados no visor.



Manipulações com os *flags* podem ser feitas utilizando-se os seguintes operadores:

SF (*Set Flag)*: Aciona o sinal do *flag*.

CF (*Clear Flag*): Limpa o sinal do *flag*.

FS? (*Flag Set?*): Testa se o sinal do *flag* está acionado.

FC? (*Flag Clear?*): Testa se o sinal do *flag* está limpo.

FS?C (*Flag Set? Clear*): Testa se o sinal do *flag* está acionado e limpa o *flag*.

FC?C (*Flag Clear? Clear*): Testa se o sinal do *flag* está limpo e limpa o *flag*.

IF ... THEN ... END

« IF *teste* THEN *comandos* END »

Se ***teste*** é verdadeiro, então ***comandos*** é executado.

```
« → X « IF 'X 0' THEN "Positivo" END » »
```

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

« IF *teste* THEN *comandos1* ELSE *comandos2* END »

Se ***teste*** é verdadeiro, então ***comandos1*** é executado, senão é executado *comandos2*.

```
« → X « IF 'X 0' THEN "Positivo" ELSE "Negativo" END » »
```

IFT

```
« teste objeto IFT »
'IFT( teste, objeto)'
```

O mesmo que IF ... THEN ... END, mas em uma forma mais enxuta e rápida. Se *objeto* for um comando, ele deve ser encapsulado em uma lista.

```
« 0   "Positivo" IFT »
```

IFTE

```
« teste objetoV objetoF IFTE »
'IFTE( teste, objetoV, objetoF)'
```

A versão IFT de IF ... THEN ... ELSE ... END. Se *teste* é verdadeiro, *objetoV* é avaliado, senão *objetoF* é avaliado. Segue as mesmas regras de IFT quanto aos uso de comandos.

```
« 0   "Positivo" "Negativo"  IFTE »
```

CASE ... THEN ... END

```
« CASE teste1 THEN comandos1 END
       teste2 THEN comandos2 END
        ...
       testen THEN comandosn END
              comandos_senão END
»
```

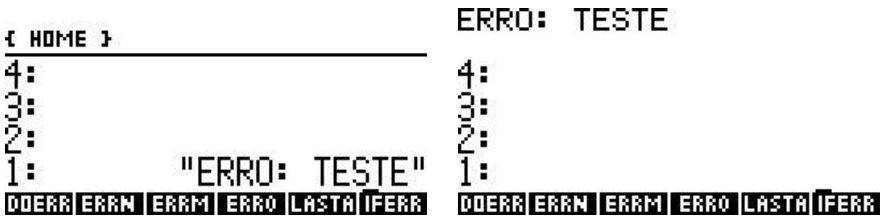
Utilizada para substituir IF's aninhados, para manter a clareza do código, esta estrutura primeiro testa de *teste1* é verdadeiro; se for executará *comandos1*, descartando o restante da estrutura. Se *teste1* é falso, testa *teste2* e, se este for verdadeiro, executa *comandos2*, descartando o resto da estrutura, e assim subseqüentemente. Se nenhum dos teste for verdadeiro, executa *comandos_senão* (opcional).

O programa abaixo diz se o objeto do nível 1 é menor, maior ou igual ao objeto do nível 2

```
«  → X Y
 « CASE 'X>Y' THEN "Menor"  END
        'X<Y' THEN "Maior"  END
       X Y SAME THEN "Igual"  END
 »
»
```

TRATAMENTO DE EXCEÇÕES

DOERR



Recebe da pilha uma cadeia de caracteres e devolve uma mensagem de erro no alto do visor, com os caracteres da cadeia

IFERR .. THEN ... END

« IFERR *comandos1* THEN *comandos2* END »
« IFERR *comandos1* THEN *comandos2* ELSE *comandos3* END »

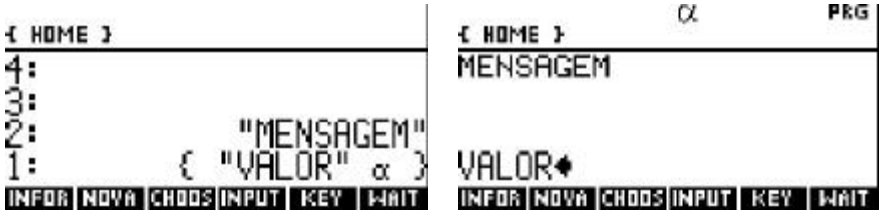
Nestas duas estruturas, semelhantes às estruturas IF ... THEN ... END e IF ... THEN ... ELSE ... END, uma seqüência de comandos é executada conforme o resultado de um teste. A diferença é de que, neste caso, a condição que aciona a execução do código é uma condição de erro durante a execução de outro trecho de código, isto é se ocorrer um erro durante a execução de *comandos1*, *comandos2* é executado, e na segunda estrutura, *comandos3* é executado se nenhum erro ocorrer.

O programa abaixo dá uma mensagem de erro personalizada caso não haja valores na pilha, caso contrário, devolve uma cópia do objeto do nível 1 avaliado:

```
« IFERR  DUP
  THEN   "ERRO: Pilha Vazia" DOERR
  ELSE   EVAL
  END
»
```

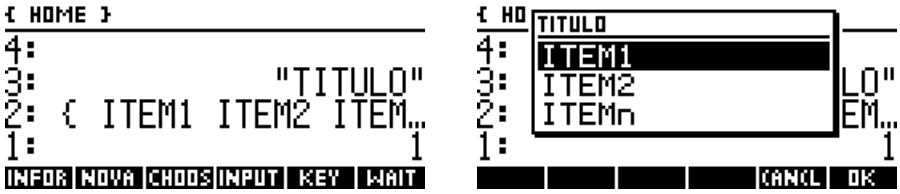
ENTRADA

INPUT



Trava a execução do programa e solicita uma entrada do usuário. O formato do comando é `"mensagem" "valor" INPUT` ou `"mensagem" {"valor" opções posição}`. As opções podem ser `ALG`, que põe a calculadora no modo algébrico, `α`, que liga o teclado alfanumérico`, ou `V`, que checará a sintaxe da entrada do usuário. A saída se dá na forma de uma cadeia de caracteres. ***Dica***: Após `INPUT`, utilize o comando `OBJ→`, que "abrirá" o string, traduzindo a saída para o formato desejado.

CHOOSE



Disponibiliza a escolha de um item. A entrada tem o formato `"Título" { itens } item_inicial`. Se `item_inicial` for `0`, então o formulário serve como informativo, não permitindo nenhuma seleção. A saída pode ser `{ item } 1`, se o usuário efetuou a seleção, ou `0`, caso o usuário tenha pressionado `CANCEL`.

INFORM

Cria um formulário personalizado, semelhante aos das aplicações imbutidas. A entrada tem o formato `"Título" { campos } formato {Valores_limpa} {valores_iniciais}`, onde cada `campo` pode assumir uma dentre as quatro formas:

```
"nome"
{ }
{ "nome" "Texto de Ajuda"}
{ "nome" "Texto de Ajuda" tipo1 tipo2 tipon}
```

Onde `tipon` segue a correspondência numérica do comando `TYPE`. Dessa forma só são aceitos valores de um dos tipos especificados.

formato pode ter uma das quatro formas:
{ }
colunas
{colunas}
{colunas distância}, onde distância é a distância entre o nome e o valor do campo.

{valores_limpa} são os valores correspondentes dos campos quando se pressiona DEL .

{valores_iniciais} não necessita demais explicações

SAÍDA

CLLCD ... DISP ... FREEZE ... HALT

A combinação CLLCD DISP FREEZE HALT mostra no visor uma mensagem até que ON seja pressionado. O formato de uso é CLLCD "mensagem" linha DISP área FREEZE HALT:

CLLCD apenas limpa o visor

DISP recebe como argumentos uma cadeia de caracteres e a linha de exibição (de 1, a linha superior, até 7, a linha inferior do visor)

FREEZE congela uma parte do visor de acordo com o valor do número passado como argumento:

NÚMERO	ÁREA DO VISOR
1	Área de status
2	Pilha
3	Área de status e Pilha
4	Menu
5	Área de status e Menu
6	Menu e Pilha
0 ou 7	Todo Visor

MSGBOX

Produz uma bonita caixa de mensagem no centro do visor. Recebe como entrada uma cadeia de caracteres. ***Dica***: Utilize CLLCD para dar um visual mais limpo.

COMANDOS - GUIA DE REFERÊNCIA

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	Devolve expressões avaliadas como argumentos a nomes locais sem avaliar	'nome' {symb ...}	
ARC	Desenha um arco em pict com um ponto central especificado, raio, e ângulo de início e fim	Point xr x1 x2	
ARCHIVE	faz uma cópia de segurança do diretório home	:n: name	
ARG	Devolve o ângulo polar	(x,y)	zΦ
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 sdat		
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 sdat 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	n	"string"

	um caracter		
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		
CLS	Apaga sdat		
CNRM	Calcula a norma da coluna de uma matriz	[array]	xnormacoluna
COLCT	Une termos semelhante de uma expressão	'symb1'	'symb2'
COLS	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 natural menos 1	x	e^x-1
FC?	Comprova se o flag especificado está desativado	mflag	0/1
FC?C	Comprova se o flag está desativado, se não o ativa	mflag	0/1
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	0/1
FREEZE	Congela uma determinada área do display	n	
FS?	Comprova se o flag especificado está ativo	mflag	0/1
FS?C	Comprova se o flag especificado está ativo, se não o ativa	mflag	0/1
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 especificado	grob1 point grob2	grob3
GRAD	Seleciona o modo 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	grob3

		point grob2	
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	0/1 objverd	
IFTE	Começa a estrutura de decisão ifte...then..[else]...end	0/1 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		ou lc.P 0 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	Pn,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	0/1
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] index z	[array2]
PUTI	Igual a put, porém incrementa o índice	[array1] index z	[array2] index2
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 π*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}	[array2]
RE	Devolve a parte real de um número (x,y)	x	
RECN	Recebe um programa via kermit e o guarda na variável especificada	'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	(x,y)

		y	
R→D	Conversão de radianos para graus	xrad	xgraus
SAME	Testa a igualdade de dois objetos	obj1 obj2	0/1
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)	{ 'nome1' 'nome2' ... } ou 'nome'	
SERVER	Seleciona o modo servidor via kermit		
SF	Habilita o flag indicado	n	
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 standard		
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 1c.p ... objn 1c.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	Extrai parte de uma lista, cadeia, matriz, 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
ÀTIME	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!

[i] Emu48 1.04 - Copyrigth 1997 Sébastien Carlier.

[ii] O autor utiliza o script Enhanced, de autoria de Casey Patterson, Sebastien Carlier e Eran Rivlis, com algumas alterações de punho próprio.