



Lenguaje de Programación Orientado a Humanos

2006

<http://www.reda4.org>
pabloreda@gmail.com

Colaboraron

Maria Jose Aguirre
Jose Maria Fantasia
Sebastian Desimone
Javier Gil Chica
Manuel Cornes
Dutra da Lacerda

Este Software esta licenciado bajo la CC-GNU GPL

Introducción

La constante aparición en el mercado de tecnología vinculada a las computadoras, con equipos cada vez más potentes y económicos, ha llevado a que los desarrolladores se inclinen, por cuestiones de mercado, a usar lenguajes de programación y otros recursos que se apartan de los principios básicos de la programación, que tienen sus bases en la arquitectura de microprocesadores, introduciendo una complejidad y un costo alto con respecto al rendimiento y dificultad de desarrollo, que se ve compensado con la incorporación de más soporte.

A fines de los 60 Charles Moore desarrolla el lenguaje Forth bajo el paradigma de una maquina de pilas, con una perspectiva diferente a la mayoría de los lenguajes. Forth obliga a mirar la construcción de sistemas como un proceso de síntesis más que de agrupamiento de componentes, apartándose de la idea de construir capas virtuales entre la computadora y el programador, considerando que esto no reduce el problema sino que lo incrementa.

El lenguaje Forth no prospera en el desarrollo de software pero si en el hardware, los microcontroladores se benefician de las características únicas y la brevedad del código generado, pero la industria del software lo ignora por motivos que no se pueden precisar con claridad.

A fines del 2005, la búsqueda de un lenguaje que planteara una alternativa a determinados problemas vinculados con la programación en C, como ser la incoherencia de las practicas bien vistas (no usar variables globales) y la obtención de un código optimizado (usar variables globales), hizo que el autor comenzara el estudio de FORTH.

En un principio la intención fue crear un lenguaje para agregarle características al FORTH, luego de un tiempo otra fue la perspectiva al considerar que el camino era el inverso, o sea, encontrar algo más simple.

:R4 es un lenguaje inspirado en ColorForth, con un diccionario basico de 109 palabras ampliable, una pila de datos, una de direcciones y 6 prefijos con distintas funciones.

Este manual es la primera documentacion del lenguaje. En el se incluye una descripcion del funcionamiento, el comportamiento de cada palabra, unos ejemplos y una referencia rapida del lenguaje.

Manual del Humano para aprender :R4

Se usa una computadora cuando se siguen las reglas que ésta provee, por ejemplo, apretar el botón de grabar.... para grabar.

Se programa una computadora cuando se crea la manera en que la máquina se comporta.

Así como el ser humano se comunica con un lenguaje, para indicarle a la computadora que queremos que haga, tambien se necesita un lenguaje. El

lenguaje es el mecanismo que permite expresar el comportamiento de la computadora.

El programa que vamos a escribir va a estar compuesto por un conjunto de palabras. Palabra, en :R4, es cualquier conjunto de letras y símbolos separadas por espacio, por ejemplo + (signo de suma) es una palabra, la distinción entre signos y letras aquí es irrelevante.

:R4 Tampoco hace distinción entre mayúsculas y minúsculas al definir o usar palabras, la única excepción es el uso del “(comillas) que, definido como prefijo, permite expresar la cadena de carácter finalizando con otras “comillas.

Existe un diccionario base, con un conjunto de palabras que son los componentes del lenguaje.

Programar en :R4 es definir nuevas palabras partiendo del diccionario base hasta poder expresar el programa que queremos hacer.

Lo que haremos será definir palabras con prefijos para variables (#datos) y código (:acciones)

Si el prefijo es # (símbolo numera) entonces la palabra es un DATO, tiene un nombre (la palabra que tiene prefijo #) y una dirección, que es el lugar de memoria donde guarda lo que contiene.

ejemplo:

#posicion 3

Si el prefijo es : (símbolo dos puntos) entonces la palabra es una ACCION, tiene un nombre, el resto de la palabra, también tiene una dirección que es donde está el código que ejecuta la máquina cuando la llama.

ejemplo:

:caminar 1 'posicion +! ;

Prefijos

Los 6 prefijos que existen en el lenguaje son

^ Incluye vocabulario

^modulo.txt | incluye el archivo modulo.txt

El prefijo ^ se utiliza para incluir un código dentro de otro, esto es útil para crear palabras en un código y utilizarlas en otro, de esta manera se evita tener que reescribir código que será utilizado en otros códigos, permitiendo la construcción de lo que llamaremos conjuntos de palabras reutilizables o vocabulario específico.

No todas las palabras están disponibles en el código que incluye otro, solo las que se exportan.

: (dos puntos) Define acción

:nueva | define nueva como la lista hasta el ; ::nuevag | define y exporta esta definición

La definición de palabras con : (dos puntos) asigna la lista de palabras a continuación como el nombre definido con el prefijo.

:: (Doble dos puntos) define palabra exportada, aparece en otro código si se incluye el archivo donde esta.

(numeral) Define dato

#memoria | define memoria como una variable#:memoriag | define y exporta una variable

La definición de variables con # (numeral) acepta los siguientes números como su contenido, los corchetes “[]” cambian a 16 bits y los paréntesis “()” a 8 bits, por último la palabra)(permite indicar la cantidad bytes que se reservan para la variable definida.

#: (numeral dos puntos) exporta la variable.

‘ (comilla simple) Apila dirección de palabra

'hola | apila la dirección de hola (ya definida)

| (línea vertical) Comentario

|blabla comentario, ignora hasta el fin de la línea

“ (comilla doble) Define memoria con texto y apila dirección

"bla bla" | texto (incluye espacios), apila su dirección

El código y la pila de datos

El Código es el guión que se escribe para programar la computadora, en :R4 el código es un archivo de texto.

Cuando se pone un número en el código para que sea procesado por la máquina dicho número será agregado en una pila (apilado). Se llama pila a una secuencia de números que permite agregar y quitar número del tope de la pila.

Se comienza con la pila vacía y se dibuja la base sin número alguno.

(

Si en el código se agrega un número, digamos 3, el dibujo de la pila es el siguiente:

(3

Al agregar otro número, por ejemplo 2, el dibujo de la pila es:

(3 2

El ejemplo anterior escrito con código fuente es:

```
prueba.txt
3 2
Produce la pila
( 3 2
```

Aunque el dibujo de la pila y el código fuente sean casi iguales ya que en el código fuente no se dibuja la base de la pila "(", son dos cosas distintas y están en diferentes lugares.

El código fuente es escrito por el programador y la pila es construida por la computadora en el momento en que el código es evaluado. El programador construye la pila indirectamente.

Es necesario tener siempre presente que relación existe entre el código y el comportamiento de la maquina. En definitiva se construye un guión para que la computadora lo realice, y este código será el producto del ingenio del programador para definir el comportamiento deseado.

Un número en el código provoca que dicho número sea apilado, cuando en el código se encuentra una palabra, interviene el diccionario.

El diccionario es una lista de palabras con su correspondiente significado. Si se escribe una palabra en el código, esta se busca en el diccionario y si se encuentra, el significado de la misma es evaluado, hay que recordar que esta evaluación o llamada a una palabra no se produce cuando se escribe el código sino cuando la computadora lo ejecuta.

En el caso de que no exista en el diccionario una palabra del código se incurre en un error, toda palabra a usar debe estar definida antes.

Al comenzar la ejecución de un código, el diccionario contiene un conjunto breve de palabras básicas que permiten hacer todo lo posible con la computadora. Estas palabras están clasificadas por su función: Manejo de pila, Aritméticas, Lógicas, Memoria, etc.

Palabras para el manejo de la pila

DROP
quita el tope de la pila

```
3 2 drop
( 3
ya que 2 es sacado por drop.
```

SWAP

Intercambia el primer y segundo numero de la pila

3 2 swap

(2 3

DUP

Duplica el tope de la pila

3 2 dup

(3 2 2

OVER

Agrega una copia del segundo número de la pila

3 2 over

(3 2 3

Notar aquí que el código y la pila ahora no son iguales y además se debe decir que la computadora hará nuestras palabras o números paso a paso, esto significa que cada palabra actúa sobre la pila en el instante en que es evaluada por la maquina y al final de la evaluación de todas las palabras se obtiene la pila indicada.

Es importante saber en todo momento el estado de la pila, como se mencionó, cada palabra puede producir un cambio, es conveniente tener algún modo de representarlo y para esto se utiliza la notación de cambio de pila.

La forma de esta notación es la siguiente:

| antes -- después

Se agrega el símbolo | para indicar en el código que a partir de este carácter y hasta el fin de la línea todo lo que el programador escribe va a ser un comentario y será ignorado por la maquina, o sea, es un comentario para el programador.

Pondremos en antes el estado de la pila ANTES de hacer o ejecutar la palabra seguido de -- y luego qué quedará después en después.

Los valores que están en la pila se representan con letras y la cantidad de letras que colocadas antes y después va a depender de lo que haga la palabra.

El listado completo, y comportamiento de las palabras para el manejo de pila es el siguiente

DUP | a -- a a

DROP | a --

OVER | a b -- a b a

PICK2 | a b c -- a b c a

PICK3 | a b c d -- a b c d a

```

PICK4 | a b c d e -- a b c d e a
SWAP  | a b -- b a
NIP   | a b -- b
ROT   | a b c -- b c a
2DUP  | a b -- a b a b
2DROP | a b --
3DROP | a b c --
4DROP | a b c d --
2OVER | a b c d -- a b c d a b
2SWAP | a b c d -- c d a b

```

Se puede decir que la pila es la memoria de corto plazo de la computadora, aunque en un principio resulte incómodo pensar en la pila, ésta es la clave de todo el lenguaje, no se desanime si al principio le resulta un poco complicado, la práctica hace transparente este mecanismo y se convierte en algo común

Ejercicios

(Soluciones al final del documento)

Con números, DROP, SWAP, DUP y OVER escriba el código para obtener las siguientes pilas utilizando la menor cantidad de números posible.

- 1.- (2 3 2 3 2 3
- 2.- (2 2 2 2 2 2
- 3.- (1 2 3 4 5 6
- 4.- (
- 5.-)

Comienzo de programa

El código en:R4 se guarda en archivos de texto, para ejecutar el código el :R4 carga este código, lo compila y luego lo ejecuta.

La ejecución comienza en el lugar donde se encuentra la palabra : (dos puntos), no hay que confundir con la definición de las palabras donde : (dos puntos) está pegado a la palabra que define.

```

:inicio ;
:continua ;
:termina ;

```

```

: inicio continua termina ;

```

Si el compilador encuentra un error graba en el archivo "debug.err" el código, el error, y el número de línea, cuando existe "debug.txt" este programa lo usa.

Palabras para cálculo

Las palabras utilizadas para realizar cálculos aritméticos, comúnmente se llaman signos pero ya se vio que esta distinción aquí no es necesaria.

+ (signo de suma)

2 3 +
(5

La palabra + (suma) toma los dos primeros números de la pila, los suma y agrega el resultado a la pila (habiendo quitado los valores anteriores). El resto de las operaciones básicas funciona de igual manera con su significado correspondiente.

+	a b -- c	c = a + b
-	a b -- c	c = a - b
*	a b -- c	c = a * b
/	a b -- c	c = a / b
*/	a b c -- d	d = a * b / c (multiplica en 64 bits)
>>	a b c -- d	d = (a * b) >> c (multiplica en 64 bits)
/MOD	a b -- c d	c = a / b d = a resto b
MOD	a b -- c	c = a resto b
NEG	a -- b	b = -a
1+	a -- b	b = a + 1
1-	a -- b	b = a - 1
2/	a -- b	b = a / 2
2*	a -- b	b = a * 2
<<	a b — c	c = a << b desplaza bits a izquierda
>>	a b — c	c = a >> b a derecha (arrastra signo)

Palabras Lógicas

Al igual que las palabras aritméticas, se dispone de un conjunto de palabras para el cálculo lógico

AND	a b -- c	c = a AND b
OR	a b -- c	c = a OR b
XOR	a b -- c	c = a XOR b
NOT	a -- c	c = NOT a

Tanto las palabras aritméticas como las palabras lógicas hacen sus cálculos sobre la pila de datos, como se ve, todo cálculo sucedará aquí.

Ampliación del diccionario

Como mencionamos anteriormente el diccionario está compuesto por un conjunto de palabras básicas. Programar en :R4 es definir palabras y para esto se utiliza el prefijo : (dos puntos), toda palabra comenzada con : se define y agrega al diccionario.

La palabra :hola escrita en el código no indica que se busque :hola sino que se agregue al diccionario la palabra HOLA (sin los :) y su definición vendrá a continuación hasta la palabra ; (punto y coma). Es muy importante recordar los espacios. Veamos una definición en un código

```
:s3 3 + ; | a -- b ( b sera a+3 )
```

Esto no quiere decir que cuando tengamos este código la pila contendrá un número resultado de agregarle 3 al tope, sino que solo definirá la palabra S3 en el diccionario. El comportamiento sera computado cuando se llame DESPUES de haberlo definido.

```
:s3 3 + ;2 s3
```

Entonces sí, se realiza la suma, nótese que aunque no exista la palabra S3 en el diccionario básico, una vez definida puede ser utilizada (siempre debe estar definida antes de utilizarse).

: (dos puntos), existe en :R4 como palabra y como prefijo, ya se vio que se pueden definir nuevas palabra utilizándolo como prefijo. Al final del código se puede definir donde comenzara a ejecutarse el programa con la palabra : (dos puntos), el código anterior estará listo cuando se agregue donde comenzara el programa.

```
:s3 3 + ;; 2 s3 ;  
( 5
```

Las definiciones : (dos puntos) son acciones que se le asigna un nombre, programar es crear estos nombres y sus definiciones. Cada definicion consume o produce numeros que se almacenan en la pila, eso es lo que indica la notacion de pila.

Es importante encontrar los nombres correctos para cada situacion.

Ejercicios

(Soluciones al final del documento)

Definir las palabras para obtener el siguiente cambio en la pila

1.- Defina cuadrado de un número con el siguiente comportamiento:

```
:**2 | a -- a*a
```

2.- Defina la suma de los 3 números del tope de la pila con el siguiente comportamiento:

```
| a b c -- a+b+c
```

3.- Suponga que no existen en nuestro diccionario 2drop y 2dup. Plantee una definición para estas palabras validando la respuesta con la notación de cambio de pila.

4.- Escriba una palabra que invierta el orden de los números de la pila

| 1 2 3 4 --- 4 3 2 1

5.- Escriba 3dup que duplique los últimos 3 números de la pila

| 1 2 3 -- 1 2 3 1 2 3

6.- Defina -rot como

:-rot | a b c -- c a b

7.- Defina las siguientes ecuaciones mostrando los efectos que se producen en la pila,

a) a^2+b^2 | a b -- c

b) a^2+ab+c | c a b -- d

c) $(a-b)/(a+b)$ | a b - c

8.- Haga un dibujo que le guste para olvidarse de las cuentas anteriores.

Las variables del programa

Así como el prefijo : (dos puntos) sirve para definir palabras que van a realizar algún tipo de acción, el prefijo # (numeral) sirve para definir palabras que van a guardar datos, también llamadas variables.

#dato

La línea de arriba, define la palabra dato o la variable dato como una posición de memoria que permite guardar un número, dicho número puede ser inicializado con un valor como se muestra a continuación:

#dato 22

Se necesitan dos palabras en el diccionario para realizar la operación de guardar un número en una variable y recuperar el número que se encuentra en dicho lugar, así:

Carga Memoria o Fetch

@ | dirección - valor

Dada una dirección de memoria devuelve el valor que se encuentra en dicha dirección.

Graba Memoria o Store

! | valor dirección --

Dado un valor y una dirección, pone en la dirección el valor correspondiente.

Es preciso en este punto conocer otro prefijo que es ' (comilla simple), este se utiliza para conocer la dirección de una palabra ya definida que es asignada por el lenguaje, esta dirección se usa para poner y sacar valores de las variables.

#dato 8'dato @ | apila un 82 'dato ! 'dato @ | ahora apila un 2

Una palabra definida con # tiene un funcionamiento distinto a una definida con :. Una palabra : es llamada cuando se la nombra en el código, en cambio, una variable (palabra definida con #) apila el valor que posee cuando es nombrada en el código.

Por lo tanto @ es innecesario la mayoría de las veces ya que dato apilara su valor y no su dirección. Las dos líneas siguientes producen el mismo resultado, porque la segunda línea apila su dirección y luego obtiene el valor con @.

dato | apila valor de dato'dato @ | apila valor de dato

En los ejemplos anteriores se define una palabra, llamada dato, que contiene un valor, por simplicidad se dice que se define la variable dato. El valor de la variable dato es el número que contiene esta variable y la dirección de la variable dato es la posición física en la memoria donde se guarda este dato, notar aquí que la dirección también es un número, pero nunca va a ser necesario conocer este valor en el código sino a través de su nombre (en este caso 'dato).

El código:

```
0 10 !
```

es válido pero completamente inútil (salvo que la posición 10 de memoria signifique algo más en la computadora donde se programe).

Una construcción muy común que se presenta es cambiar el valor de la variable con una variación de su propio valor, así, para sumar 1 al valor de la variable dato debemos hacer:

```
#dato 0:suma1 dato 1 + 'dato ! ;
```

Esta definición puede reescribirse de la siguiente manera

```
#dato 0:suma1 1 'dato +! ;
```

Ya que +! incrementa el valor de la posición de memoria.

En la memoria se encuentran las definiciones del diccionario y las variables de :R4, cuando se obtiene la dirección de una palabra esta se refiere a la memoria donde fue ubicada por el lenguaje.

Es importante adquirir práctica en el uso de las variables, los valores y las direcciones, siempre un dibujo ayuda a entender estas diferencias.

En la definición de variables es posible asignar varios números a una variable, ejemplo:

```
#lista 10 20 30 40 50
```

Aquí lista apilará el primer número, o sea 10. Cómo se supone que se obtiene el resto?

Pues bien, con la dirección de la variable lista.

```
'lista 4 + @
```

En este caso se apilará el segundo número, o sea 20, si reemplazamos el 4 con 8 obtendremos el tercero y así sucesivamente. Para entender porque es 4 y no 1 debemos saber como se guardan los números en la memoria.

:R4 define cada número con 4 bytes, pero es posible tomar menos bytes en la memoria. Así como @ toma los 4 bytes, c@ toma 1 byte y w@ toma dos bytes, también existen c! que pone un byte y w! pone 2 bytes.

Recordar siempre estas cantidades ya que trabajaremos con direcciones y serán necesarias para calcular la posición correcta.

Se puede indicar la cantidad de bits de los numeros utilizando corchetes y parentesis

```
#de32bits 1 2 3 | números de 32 bits#de16bits [ 1 2 3 ] | números de 16 bits#de8bits ( 1 2 3 ) | números de 8 bits
```

La primera variable ocupa $4*3=12$ bytes de memoria, la segunda ocupa 6 bytes y la tercera 3 bytes

Existe otra construcción para determinar cuantos bytes debe tener la variable, por ejemplo para definir un lugar de 1024 bytes o 1K:

```
#de1KBYTE )( 1024
```

Así como los números son guardados en 4 bytes, los caracteres son guardados en 1 byte.

Un detalle más respecto de los números en :R4, existen dos prefijos para poder indicar los números en base binaria y hexadecimal, estos son % y \$ respectivamente:

Decimal	Binario	Hexadecimal
0	%0	\$0
2	%10	\$2
10	%1010	\$A
15	%1111	\$F
16	%10000	\$10

En la memoria está todo lo que la computadora hace, guarda, muestra y también existe un lugar de memoria vacío para trabajar, sin este lugar la maquina no puede crear sino solo mostrar.

La única palabra que se encuentra en el diccionario base es MEM, esta deja en la pila la dirección de comienzo de la memoria libre, haga lo que quiera con esta memoria, es suya, le vino con la maquina.

Estructuras de control

La decisión es el mecanismo por el cual se elige una opción o un camino. Para representar una decisión se debe indicar cual será la condición y luego que camino o palabras va a llamarse de acuerdo a esta condición.

La condición se evalua, adivine donde... si, en la pila de datos, se tienen 4 palabras que evalúan esto.

0? Salta si el tope de la pila es 0
1? Salta si el tope de la pila NO es 0
+? Salta si el tope de la pila es positivo
-? Salta si el tope de la pila es negativo

Estas palabras usan el tope de la pila para comparar, sin modificar la pila. Un segundo grupo de palabras comparan los dos números al tope de la pila consumiendo el primero

=?	a b -- a	a = b ?
<>?	a b -- a	a <> b ?
>?	a b -- a	a > b ?
<?	a b -- a	a < b ?
<=?	a b -- a	a <= b ?
>=?	a b -- a	a >= b ?
and?	a b — a	a and b tiene los bits en 1?
nand?	a b – a	a and b tiene los bits en 0?

El salto condicional

El salto al que se hace referencia se indica con paréntesis, sin embargo recuerde que los paréntesis también son palabras, veamos un ejemplo:

0? (drop)

Esta linea en el código saca el tope de la pila si es 0, de lo contrario no hará nada ya que continuara sin hacer drop. A esta construcción se la llama IF, ya que representa un si condicional.

Otro tipo de salto condicional es indicando las dos posibles acciones, cuando es verdadero y cuando no lo es.

+? (dup)(drop)

Conviene pensar en bloques de palabras encerradas entre paréntesis, aunque en realidad los paréntesis abierto “(“, cerrado-abierto “)(“ y cerrado “)” son tres palabras, en el ejemplo anterior si el tope de la pila contiene un número negativo SALTA hasta la palabra siguiente de “)(“ y si es positivo continua hasta “)(“ y luego de aquí salta hasta “)”. Entonces, si el tope es positivo duplica este número, en caso contrario lo quita. Los bloques de palabras deben cerrarse siempre, dejar un bloque abierto es un error y será indicado por el lenguaje.

Repetir siempre

Para repetir una lista de palabras, estas deben estar encerradas entre paréntesis, por ejemplo:

```
:limpiar | ... definimos limpiar ;  
:ensuciar | .. definimos ensuciar ;  
: ( ensuciar limpiar ) ;
```

Al ejecutar el programa, este repetirá indefinidamente las dos palabras ensuciar y limpiar.

Repetir mientras

El nombre de la repetición surge directamente de lo que significa. Mas útil que repetir algo indefinidamente es definir una condición de corte, para esta definición, note aquí que la posición del condicional define el tipo de construcción:

```
:printn | n -- imprime el tope de la pila consumiéndola ; 5 ( 1? )( 1 – dup printn  
) drop ; |imprime 4 3 2 1 0
```

La última línea se lee.

Apile 5 y MIENTRAS no sea 0 repita... restele 1 dupliquelo e imprimalo, es necesario duplicar ya que imprimirlo lo consume.

Como ve es necesario cuidar el estado de la pila en cada repetición, los condicionales que consumen la pila generalmente son más fáciles de leer:

```
: 0 ( 5 <? )( 1 + dup printn ) drop ; |imprime 1 2 3 4 5
```

Esto es: apile 0 y MIENTRAS sea menor a 5 sumele 1 e imprimalo (ya sabemos que lo duplicamos para imprimir).

Repetir Hasta

```
: 5 ( 1 – dup printn 0? ) drop ; | imprime 4 3 2 1 0
```

Esta construcción se repetirá HASTA que el tope de la pila sea cero.

Estas construcciones de control deben estar correctamente anidadas, es decir, debe existir la misma cantidad de paréntesis que abren y cierran, además “;” (punto y coma) dentro de paréntesis no terminan la definición sino que termina la ejecución de la palabra.

Pila R

Cuando se llama un palabra, se apilar en la pila R la direccion donde debe continuar ejecutando cuando finalice la palabra llamada.

Para usar la pila R contamos con las siguientes palabras.

RDROP	--	r: a --
R>	a --	r: -- a
>R	-- a	r: a -
R	-- a	r: a - a
R+	v --	r: a - a+v
R!+	v --	r: a - a+4
R@+	-- v	r: a - a+4

La pila R registra el flujo de ejecucion del programa, asi, en el siguiente ejemplo

```
:hace "hace" print rdrop ;
:llendo hace "no paso" print ;
:quehace? llendo ;
```

Nunca imprime "no paso" porque rdrop en hace quita una direccion de la pila.

La palabra EXEC tambien puede definirse con el manejo de la pila R.

```
:exec 0? ( drop ; ) >r ;
```

Es posible y conveniente utilizar esta pila como una pila auxiliar para datos, para guardar resultados intermedios o saltar calculos.

Aquí R funciona como un lugar donde poner un dato para despejar la pila.

```
:hlink | x1 y1 x2 y2 -
      over >r op r> over line line ;
```

Para recorrer porciones de memoria escribiendo (R!+) o leyendo (R@+). Aquí recorre la memoria que empieza en inicio hasta que encuentra un 0, copia cada numero a partir de la direccion que le pasamos como parametro.

```
:recorre | a -
      >r inicio ( @+ 1? )( r!+ ) r!+ rdrop ;
```

Vectores de ejecucion

Se llama vector a una variable que contiene una dirección, pero en esta dirección se encuentra el código de una palabra y no un dato, lo que permite cambiar su significado en cualquier momento y da una importante herramienta de construcción. A continuación se presenta un ejemplo:

```
:suma1 1 + ;
:suma2 2 + ;
#suma
:define1 'suma1 'suma ! ;
:define2 'suma2 'suma ! ;
: 3 define1 suma exec define2 suma exec ;
```

(6)

Hay una forma de definir comportamiento sin nombre que se denominan palabras anónimas, estas definiciones tienen sentido cuando estamos definiendo un comportamiento que se usa solamente una vez y se referencia con su dirección, note aquí que esta dirección será llamada con EXEC, ya sea inmediatamente o en otra definición

Los corchetes permiten construir estas definiciones

[exit ;] >esc< | comportamiento asignado a la tecla esc

Tanto [(abrir corchete) como] (cerrar corchete) son palabras, la primera indica que el código a continuación no se ejecutará inmediatamente, la segunda finaliza el código y apila la dirección de este código definido.

Punto Fijo

:R4 usa como alternativa a los números enteros los números decimales almacenados en punto fijo

Para esto transforma la notación de dos números con un punto en un entero.

Se utiliza 16.16 o se 16 bits para entero, 16 bits para decimal.

así
2.0 es \$20000
1.0 es \$10000
0.5 es \$7ffff

La suma y la resta no necesitan ajuste.

2.0 1.5 + 0.03 -

la multiplicación necesita correrse 16 bits a la derecha

```
:. 16 *>> ;
```

la division necesita 16 bits a la izquierda en el primer operando (¿?)

```
:/ swap 16 << swap / ;
```

El Sistema

El siguiente grupo de palabras manejan los aspectos referidos al sistema, las primeras 3 palabras obtienen información y las dos siguientes controlan la maquina virtual.

MSEC | -- a
Apila los milisegundos actuales del sistema

TIME | -- h m s
Apila la hora, los minutos y los segundos del sistema

DATE | -- d m a
Apila el día, el mes y el año del sistema

END | --
Detiene la maquina virtual, cuando :R4 se ejecute en una maquina real esta palabra apagara la computadora, hoy, retorna al SO base que ejecuta la maquina virtual.

RUN | d -
Carga el código indicado con el nombre del archivo y lo ejecuta. Esto permite encadenar programas.

La pantalla

La pantalla de la computadora es una cuadrícula de luces donde cada una de ellas puede tomar cualquier color, cada punto de luz se llama píxel.

El ancho y el alto de la pantalla varían de acuerdo a la elección de quien utiliza un programa en :R4 por el momento dispone de dos versiones 640x480 y 1027x768.

Si se requiere más poder de computo se logra con la pantalla más chica ya que la maquina necesitara menos esfuerzo para dibujarla.

SW	-- w	Ancho de pantalla (ScreenWidth)
SH	-- h	Alto de pantalla (ScreenHeight)
CLS	--	Borra la pantalla con el color de fondo
REDRAW	--	Copia la pantalla virtual a la real
FRAMEV	-- a	Inicio de la memoria de video
UPDATE	--	Actualiza los eventos internos del SO

Dibujo

Para dibujar en la pantalla se utilizan estas palabras. Las coordenadas son pasadas a través de la pila y los números corresponden con la posición física en pantalla, no es conveniente utilizar constantes aquí, sino variable calculadas de acuerdo al tamaño de la pantalla para que el dibujo sea independiente de la resolución.

OP	x y —	Punto de origen
CP	x y —	Punto de control para curva
LINE	x y —	Traza línea
CURVE	x y —	Traza curva
PLINE	x y --	Marca línea polígono
PCURVE	x y --	Marca curva polígono
POLI	--	Traza polígono
PAPER	a —	Asigna color de fondo
INK	a --	Asigna color de dibujo
INK@	-- a	Apila color de dibujo
ALPHA	a --	Asigna transparencia de color

Interaccion

Para la interacción se asignan acciones a las teclas y al dispositivo apuntador en pantalla.

XYMOUSE	-- x y	Coordenadas del apuntador
BMOUSE	-- b	Estado del apuntado
KEY	-- s	Ultima tecla pulsada
IPEN!	v —	Evento de Mouse
IKEY!	v —	Evento del teclado

Archivos

El manejo de la memoria auxiliar externa con :R4 se realiza con las siguientes palabras.

DIR	"" —	Cambia la carpeta actual
FILE	nro -- ""	Obtiene el nombre dado el numero
LOAD	's "" -- 'h	Carga un archivo en memoria
SAVE	's c "" —	Graba una porción de memoria

Hasta aquí se describen los mecanismos básicos del lenguaje, las palabras descriptas a continuación completan el diccionario base.

Extensión del vocabulario

El diccionario base esta construido en la maquina virtual que ejecuta el lenguaje, para facilitar la programación se crean librerías o vocabularios específicos para cada aspecto.

Librerias

reda4.txt – sistema
gui.txt – botones, pantalla
fontv8.txt – fuente vectorial
sprites.txt – dibujos multicolores vectoriales

¿ Instalar ?

:R4 no necesita ser instalado, simplemente descomprima en una carpeta (en cualquier lugar) el .zip que bajo del sitio llame al reda4.exe

¿ Como arranca :R4 ?

Cuando se llama a r4.exe en win o r4 en lin, este carga el programa main.txt y lo ejecuta, por favor, modifique este archivo para agregar, quitar y modificar programas, no tenga miedo de romper algo, guarde una copia de la carpeta si quiere restaurar el estado inicial.

Observe que dentro de un programa puede llamar a otro usando:
"otro.txt" RUN

note que otro.txt es el nombre del archivo del código fuente del programa que quiere llamar, el nuevo programa comenzara a ejecutarse o indicara en la pantalla el error ocurrido con numero de linea y problema.

El código de cada programa es el archivo de texto en la carpeta descomprimida.

Palabras finales

Es el deseo del autor encontrar gente que utilice, mejore y aumente la cantidad de programas para :R4.

No se desanime si al comienzo nota que no sabe por donde empezar un programa, es normal, tanto en FORTH como en :R4 la brevedad obliga a pensar detenidamente cada cosa.

Si un conjunto de palabras se repiten mucho, haga otra palabra y vea como se reduce el codigo, a veces algunas variables desaparecen, acostumbrese a borrar codigo tanto como a escribirlo.

Utilice solamente numeros enteros, si quiere fracciones multiplique la unidad para dividirla en partes, no use punto flotante, esta agregando incertidumbre en la representacion de sus numeros.

Trate de reducir el codigo siempre, menos palabras siempre es mas facil de entender que muchas palabras, por supuesto que tambien andara mas rapido.

Como dijo alguien, la simplicidad es un punto de llegada, no de partida.

Programando en :R4

Hace algún tiempo 64 kb era mucha memoria, los programas venían en revistas, se podían copiar tecleando y se guardaban en casetes, esto ultimo no era para nada divertido pero lo primero si, ya que se aprendía mientras se tecleaba el código.

Veán el siguiente ejemplo, los números están como referencia a las líneas y no deben ser copiados.

```
1] ^reda4.txt
2] ^gui.txt
3] :inicio
4]     'exit >esc<
5]         show cls
6]         16 16 screen blanco
7]         "Hola Mundo" print ;
8]
9] : inicio ;
```

La línea 1 y 2 indica que incluya las palabras para gráficos, animación, teclado y fuentes, casi siempre esta línea está presente.

La línea 3 define la palabra inicio que es llamada al comienzo del programa en la línea 9.

La línea 4 asigna la acción de salir a la pulsación de la tecla ESC.

La línea 5 utiliza SHOW para indicar las palabras que dibujan la pantalla hasta el ; (punto y coma) de la línea 7, como primera palabra se borra la pantalla con CLS.

La línea 6 define el tamaño de las letras que se dibujarán, screen traduce la cantidad de columnas y filas que entrarán en la pantalla, luego se indica el color blanco para dibujar.

La línea 7 imprime en pantalla el texto entre comillas.

Ejemplo de animación en :R4

Veamos un ejemplo para realizar una animación de una pelota rebotando.

```
01] ^reda4.txt
02] ^gui.txt
03] #pelota $cc004 $7493FE85 $6BCE50D7 $22DC7 $93D250D7
04] $8B73FE87 $9435B557 $1CF37 $6C31B3C7 $7493FE87 0
05] #xb 0 #yb 0 #vx 8 #vy 0 #ay 2
07] :toc
08]     vx neg 'vx ! ;
09] :tic
10]     vy neg 'vy ! ;
11] :pant
12]     'exit >esc<
13]         show cls
14]         100 100 dim xb yb pos 'pelota sprite
15]         xb vx + sw >? ( toc ) 0 <? ( toc ) 'xb !
```

```

16]          ay 'vy +!
17]          yb vy + sh >? ( tic drop sh ) 'yb ! ;
18]
19] : pant ;

```

La línea 1 y 2 incluye las extensiones.

Las líneas 3 y 4 definen un espacio de 11 números de 32 bits, estos números representan el dibujo de la pelota, dicho dibujo puede realizarse con el programa GARABATOR y luego tomarse del archivo dibujos.txt que este genera al salir.

La línea 5 define las variables que se utilizan, note que XB YB son las coordenadas que en la línea 14 se indican como la posición del dibujo (sprite en la jerga de los viejos microcomputadores). VX VY representan la velocidad de cada coordenada y AY es la aceleración vertical, algo así como la gravedad. En las líneas 7 y 8 definimos TOC como el cambio de signo de VX (cuando choca contra los bordes)

En las líneas 9 y 10 definimos TIC como el cambio de signo de VY (cuando choca contra el "piso")

Note aquí la línea 19 como el comienzo del programa indicado con : con un espacio y a continuación la palabra definida entre las líneas 13 y 19.

La línea 14 asigna a la pulsación de la teclas escape (ESC) el comportamiento de salir del dibujo de la pantalla.

Note aquí que la asignación de acciones a teclas es directo, no hay comportamiento mas rápido y simple que asignar a cada evento externo una acción

La línea 13 utiliza SHOW para indicar el inicio de las palabras de dibujo de la pantalla, este dibujo se realiza 30 veces por segundo repitiendo las palabras que se encuentran entre SHOW y el fin de la definición CLS por su parte borra la pantalla, prueba quitando CLS y vea que pasa.

La línea 14 primero define las dimensiones del dibujo, cambie los 100 antes de DIM (que indica la dimensión) y vea que ocurre, luego la posición esta indicada por las variables XB y YB como habíamos visto, antes de POS (por posición) y luego dibuja con SPRITE el dibujo definido en 3.

La línea 15 calcula el movimiento horizontal, visto paso a paso ocurre esto.

```

xb      | apila el valor de XB
vx      | apila vx
+       | suma los dos
sw      | apila el ancho de la pantalla
>?     | es el valor sumado mayor al ancho ?
(       | Si así es entonces haga lo que esta entre ( )
toc     | rebote nomas ( definido arriba )
)       |
0       | apila un 0
<?     | es el valor sumado menor a 0 ?
(       | Si así es entonces haga lo que esta entre ( )
toc     | rebote
)       |
'xb    | apile dirección de xb
!      | guarde en xb la suma que realizo al comienzo

```

La línea 18 suma AY en VY

La línea 19 es similar a la 17 pero solamente comprueba el piso, note aquí que se reemplaza el valor sumado si supera el piso.

Para probar:

Agregue entre la 7 y la 8 la siguiente línea

```
rand 4 << 4 or 'pelota !
```

Que ocurre con la pelota ?, y si quiero que haga lo mismo cuando pica ?

Soluciones

Ejercicio 1

```
2 3 over over over over
2 dup dup dup dup dup
1 2 3 4 5 6
```

¿?

Ejercicio 2

```
**2 dup * ;
suma3 + + ;
:2drop drop drop ; :2dup over over ;
:inv4 1 2 3 4 swap 2swap swap ;
:3dup dup 2over rot ;
:-rot rot rot ;
:resa **2 swap **2 + ; :resb over + * + ; :resc 2dup - -rot + / ;
```

Referencia :R4 Base

Prefijos					
^modulo.txt			incluye el archivo modulo.txt		
:nueva			define nueva como la lista hasta el ;		
::nuevag			define y exporta esta definición		
#memoria			define memoria como una variable		
#:memoriag			define y exporta una variable		
'hola			apila la dirección de hola (ya definida)		
blabla...			comentario, ignora hasta el fin de la línea		
"bla bla"			texto (incluye espacios), apila dirección		
Control					
(...)			Repetir las PALABRAS dentro de paréntesis		
?? (.v.)(.f.)			Condición ?? (.v.) tambien es valido		
(.f. ??)			Repetición <i>hasta</i> condición (pasa 1 vez al menos)		
(.. ??)(.v.)			Repetición <i>mientras</i> condición		
[...]			Definición anónima (apilando su dirección)		
EXEC	d --		Llama a la direccion en pila		
Condicionales ??					
0?	--	1?	--	Es 0/distinto de 0 el tope de la pila?	
+?	--	-?	--	Es positivo/negativo el tope de la pila ?	
=?	ab - a	<>?	ab - a	a = b ? a <> b ?	
>?	ab - a	<?	ab - a	a > b ? a < b ?	
<=?	ab - a	>=?	ab - a	a <= b ? a >= b ?	
and?	ab - a	nand?	ab - a	test de bits con mascara b	
Pila de Datos					
DUP	a -- aa	ROT	abc -- bca		
DROP	a --	2DUP	ab -- abab		
OVER	ab -- aba	2DROP	ab --		
PICK2	abc -- abca	3DROP	abc --		
PICK3	abcd -- abcda	4DROP	abcd --		
PICK4	abcde - abcdea	2OVER	abcd -- abc dab		
SWAP	ab -- ba	2SWAP	abcd -- cdab		
NIP	ab - b				
Lógicas					
AND	ab -- c	c = a AND b	XOR	ab -- c	c = a XOR b
OR	ab -- c	c = a OR b	NOT	a -- b	b = NOT a
Aritméticas					
+	ab -- c	c=a+b	NEG	a -- b	b= -a
-	ab -- c	c=a-b	ABS	a - b	b= a
*	ab -- c	c=a*b	1+	a -- b	b=a+1
/	ab -- c	c=a/b	1-	a -- b	b=a-1
*/	abc -- d	d=a*b/c	2/	a -- b	b=a/2
/MOD	ab -- c d	c=a/b	2*	a -- b	b=a*2
MOD	ab - d	d=a resto b	<<	ab -- c	c=a<<b
*>>	ab - d	d=(a*b)>>c	>>	ab -- c	b=a>>b

Memoria					
@	a - b	b=32(a)	@+	d - d+4 v	
C@	a -- b	b=8 (a)	C@+	d -- d+1 byte(v)	
W@	a -- b	b=16(a)	W@+	d -- d+2 word(v)	
!	vd --	32(d) =v	!+	vd -- d+4	
C!	vd --	8(d) =v	C!+	vd -- d+1	
W!	vd --	16(d)=v	W!+	vd -- d+2	
+	vd --	32(d)=32(d) + v	W+!	vd -- 16(d)=16(d) + v	
C+!	vd --	8(d)=8(d) + v	MEM	-- d	memoria libre
Pila de Direcciones					
>R	a --	R: --a	R<	-- a	R:a --
R	-- a	R:a--a	R+	v --	R:a - a+v
R@+	-- v	R:a--a+4	R!+	v --	R:a -- a+4
			RDROP	--	R:a --
Sistema					
MSEC	-- a	Milisegundos del sistema			
TIME	-- h m s	Hora minutos y segundos			
DATE	-- d m a	Dia mes y año			
RESET	--	Sale del r4, finaliza y apaga la maquina virtual			
RUN	d --	Carga, compila y ejecuta el archivo con nombre en d			
Pantalla					
SW	-- w	Apila ancho de pantalla			
SH	-- h	Apila alto de pantalla			
CLS	--	Limpia pantalla			
REDRAW	--	Dibuja la pantalla real con la virtual			
FRAMEV	-- a	Apila direccion de inicio de la pantalla virtual			
UPDATE	--	Actualiza los eventos internos del SO			
Dibujo					
OP	xy --	Punto de origen			
CP	xy --	Punto de control para curva			
LINE	xy --	Traza línea			
CURVE	xy --	Traza curva			
PLINE	xy --	Marca línea polígono			
PCURVE	xy --	Marca curva polígono			
POLI	--	Traza polígono			
Color					
PAPER	a --	Pone color de fondo			
INK	a --	Pone color actual			
INK@	-- a	Apila color actual			
ALPHA	a --	pone alpha (0-255)			
Almacenamiento externo					
DIR	"" --	Cambia la carpeta actual			
FILE	n -- ""	Obtiene el nombre dado el numero			
LOAD	a"" -- b	Carga un archivo en memoria			
SAVE	ac"" --	Graba una porción de memoria			