Cadence skill 语言简介

Cadence提供二次开发的SKILL语言，它是一种基于通用人工智能语言—Lisp的交互式高级编程语言(LISP即List Processing—表处理，是最早和最重要的符号处理编程语言之一，它于1958年由美国的J. McCarthy提出，LISP在人工智能AI方面获得广泛应用)。

SKILL语言支持一套类似C语言的语法，大大降低了初学者学习的难度，同时高水平的编程者可以选择使用类似Lisp语言的全部功能。所以SKILL语言既可以用作最简单的工具语言，也可以作为开发任何应用的、强大的编程语言。 SKILL可以与底层系统交互，也提供了访问Cadence各个工具的丰富接口。用户可以通过Skill语言来访问，并且可以开发自己的基于Cadence平台的工具。

1. Skill语言和Lisp语言的关系

Skill函数提供两种表示法，一种是代数表示法，现在大多数语言采取这种方式，即func(*arg1 arg2 ...*)，另一种是前缀表示法，类似于Lisp语言，即(func *arg1 arg2* ...)。这里举个例子作为对比：

1．代数表示法

```
procedure( fibonacci(n)
  if( (n == 1 || n == 2) then
    1
  else fibonacci(n-1) + fibonacci(n-2)
  )
)
```

2．前缀表示法

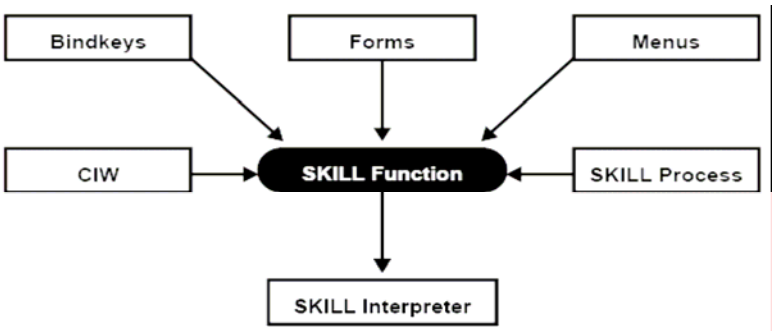
```
(defun fibonacci (n)
  (cond
    ((or (equal n 1) (equal n 2)) 1)
      (t (plus (fibonacci (difference n 1))
        (fibonacci (difference n 2))))
  )
)
```

这里可以看到类似Lisp语言的表示法后面有很多右括号，而且函数和参数容易混淆，所以一般推荐还是用常用的类C语言代数表示法

Skill程序就像一个list表，类似Lisp语言，程序的操作就像操作数据(list)一样，可以生成，修改，求值等

2. 关于 Skill 函数

SKILL 语言支持一套类似 C 语言的语法，初学者有了一定的 C 语言基础，入门是很容易的。Cadence 的工具可以通过 CIW，Bindkey，Form，Menu 等多种方式调用 skill 函数，送到 skill 语言的解释器来执行各种操作。



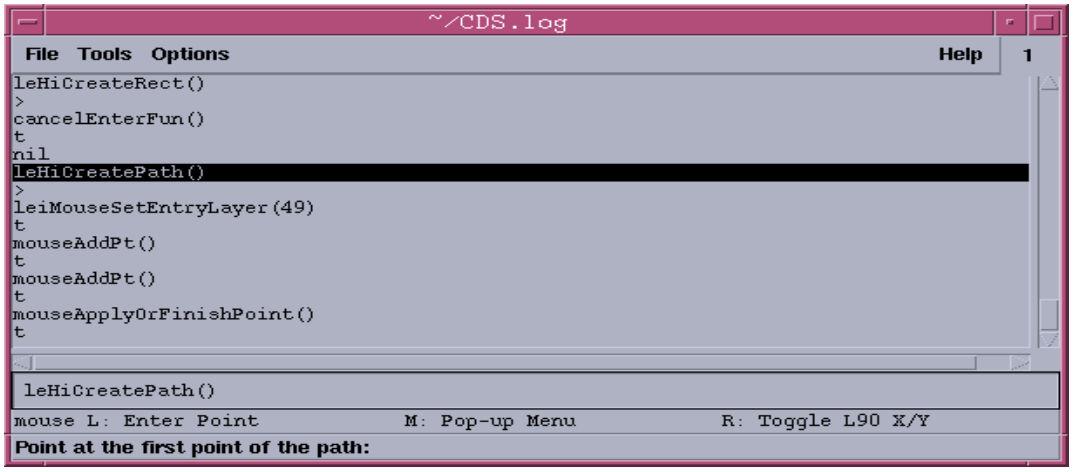
其中CIW(Command Interpreter Window)，即启动Icfb的第一个窗口，包含一些常用的menu，一个输入行，以及一个输出区域，这里是常用的debug skill程序的地方，当然cadence也提供了Tools->Skill Development，有兴趣的话可以深入研究一下

2.1 Skill 函数的查阅方法

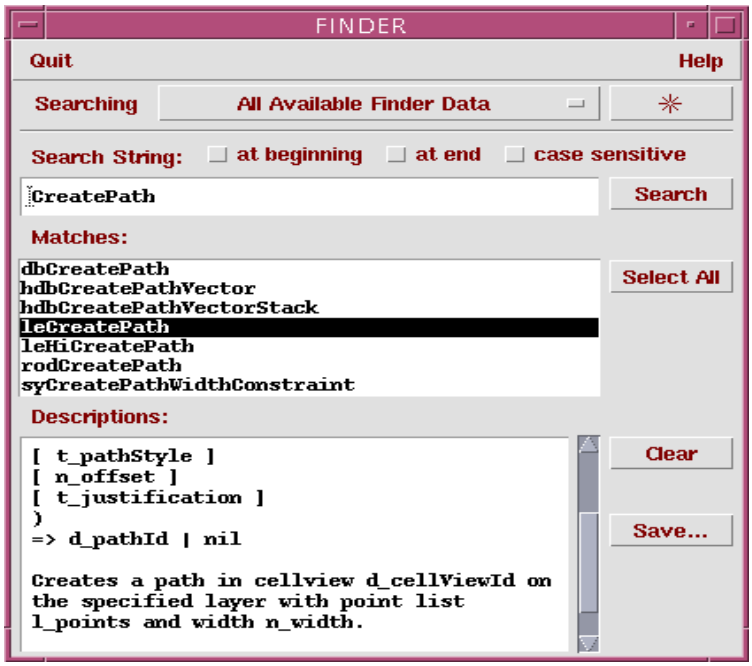
Skill语言有n多函数，加上众多工具的接口函数，可以用成千上万来形容，初学者可能会感到晕头转向，无从下手。其实只要了解基本的变量，控制语句，输入输出以及一些常用工具的基本函数就可以了，大多数函数都可以用到再查阅。开始学习skill时可以仔细看看SKILL Language User Guide，其他的可以用到时再查阅，这里列出几个经常会用到的文档：SKILL Language Reference －Skill语言相关的函数，Cadence User Interface SKILL Functions Reference －Cadence应用程序图形界面接口函数，Design Framework II SKILL Functions Reference，SKILL Custom Layout Reference，SKILL Schematic Composer Reference分别是Cadence的Design Framework II，版图，电路的接口函数。

当然还有其他众多工具的接口函数，因此Skill需要方便的函数查阅方法，这里列出3种：

1．最简单的方法，看~/CDS.log 文件或者 CIW 的输出区域，把 Options-> LogFilter 都选上。这里一般可以查阅在图形界面下运行的函数，如图，我们可以看到画 path 线的函数以及输出结果等



2．在 CIW 里输入命令 startFinder，或者在 Terminal 上输入 cdsFinder& ．Finder 适用于至少知道函数名的一部分，这里可以查到函数的用法简介。例如上图我们看到画 path 的函数是 leHiCreatePath()，但这是图形方式的函数，对应 skill 方式的函数呢用法呢，可以在 Finder 中输入 CreatePath，这样我们就可以看到包含有 CreatePath 的所有函数，如图，leCreatePath 即是我们想找的函数，这里可以看到简单的用法介绍



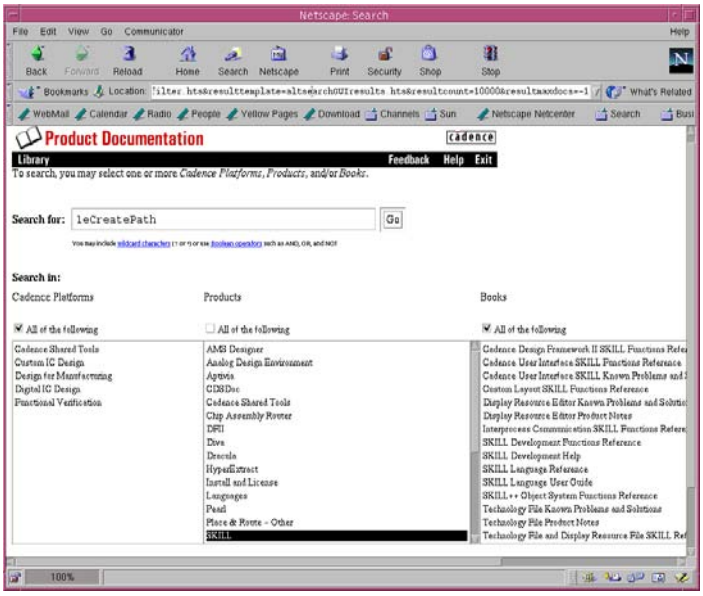
3．上面我们查到的关于函数用法的介绍很简单，如何知道其详细介绍呢。当然如果你知道函数在那个文档里，直接打开就行了，如果不知道，就需要打开 help 的 search 功能，这里可以查到详细的功能用法。

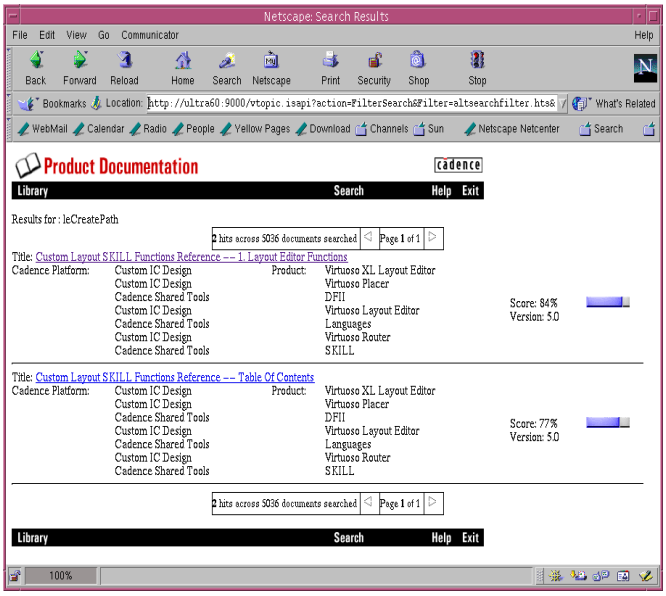
运行 cdsdoc&，点击 Search 打开 Cadence help 的搜索功能，工具会自动启动一个 http 服务，同时打开搜索网页，启动服务后，你也可以在 windows 的 IE 里输入 http://hostname:9000/search.htm，同样可以打开搜索页面，需要注意的是，在 C:\WINNT\system32\drivers\etc\hosts 文件中需要加入 hostname 对应的 ip，如：

127.0.0.1　　localhost

10.0.10.2　　host1

Cadence help 的搜索功能默认是用 Netscape 打开搜索网页，如果没有装 Netscape，可能会打不开网页。可以先打开 firefox，mozila 等浏览器，再启动 cadence doc 搜索 http 服务，这样就可以打开搜索网页了





3. Skill 语言简介

Skill 语言的很多地方和 C 语言差不多，如变量，函数，控制结构，输入输出等，详细的介绍可以参考 User Guide，这里只作简单介绍。

3.1. Skill 的变量不需要事先声明，Skill 第一次用到是会自动生成变量。变量可以由字符、数字、“_” 和 “?” 组成，注意第一个字符不能是数字和”?”。由于 Cadence 所开发的 Skill 中的变量、函数都是第一个字母小写，以_为开头的是 Cadence 的专用函数，为了避免冲突，建议大家函数和变量命名都以大写字母开头。

3.2. Skill 的函数的调用方式有三种，

```
strcat( "Hello" "," " everyone" "!" )      ;常见的类C格式
( strcat "Hello" "," " everyone" "!" )     ;类Lisp语言的格式
strcat "Hello" "," " everyone" "!"         ;上面的括号可以省略
```

返回的结果都是 `=> "Hello, everyone!"`

推荐使用第一种方式，需要注意的是函数和第一个括号之间没有空格，否则会报错

如：

```
strcat ( "Hello" "," " everyone" "!" )
    => *Error* eval: not a function - "Hello"
```

这是调用 skill 函数的一个常见问题，其它的问题还有：

a. 软件没响应，比如在 CIW 中输入段代码，软件没有反映，什么结果也没有，一般是因为( )或者” “不成对造成的，一般可以通过键入] 来解决，它表示补充完不对称的括号(可以代替任意多个右括号)，如果还没有响应，键入 “] 这时大部分情况下，系统会有响应

b. 数据类型不匹配，如：

```
strcat( "Mary had a" 5 )
=>*Error* strcat: argument #2 should be either a string
  or a symbol (type template = "S") - 5
```

3.3. Skill List 是 Skill 基于 Lisp(List Processing)语言的表现，它是 skill 数据对象的一个有序集合，skill 数据甚至程序本身都可以看作是一个 list，这是 C 语言中所没有的概念。下面是 skill list 的简单例子

List	Explanation
(1 2 3)	A list containing the integer constants 1, 2, and 3
(1)	A list containing the single element 1
( )	An empty list (same as the special atom *nil*)
(1 (2 3) 4)	A list containing another list as its second element

A). 创建 list 有以下几种基本的方法：

- Specify all the elements of the list literally with the single quote (‘) operator.
- Specify all the elements as evaluated arguments to the `list` function.
- Add an element to an existing list with the `cons` function.
- Merge two lists with the `append` function.

1). 用 ' 和 list 定义一个list，注意两者的差别

```
a = 1                          => 1
b = 2                          => 2
'( a b 3 )                     => ( a b 3 )
list( a b 3 )                  => ( 1 2 3 )
```

2). 用cons命令添加一个元素到一个list的头部

```
result = '( 2 3 )            => ( 2 3 )
result = cons( 1 result )    => ( 1 2 3 )
```

3). 用append命令合并两个list

```
Lista = '( 4 5 6 )            => ( 4 5 6 )
Listb = '( 1 2 3 )            => ( 1 2 3 )
Listc = append( Lista Listb) => ( 4 5 6 1 2 3 )
```

B). 访问 list 或者 list 中某些元素的方法：

car 访问 list 的第一个元素

```
numbers = '( 1 2 3 ) => ( 1 2 3 )
car( numbers ) => 1
```

cdr 访问 list 除了第一个元素外的其他元素，注意返回仍然是个 list

```
numbers = '( 1 2 3 ) => ( 1 2 3 )
cdr( numbers ) => ( 2 3 )
```

nth 用索引访问 list 的某个元素，注意索引从 0 开始

```
numbers = '( 1 2 3 ) => ( 1 2 3 )
nth( 1 numbers ) => 2
```

member 检查指定的元素是否在指定的 list 中，它只检查顶层元素的元素，返回值是从搜到值开始到结尾的 list

```
numbers = '( 1 2 3 ) => ( 1 2 3 )
member( 4 numbers ) => nil
member( 2 numbers ) => ( 2 3 )
```

length 计算 list 所包含元素的个数

```
numbers = '( 1 2 3 ) => ( 1 2 3 )
length( numbers ) => 3
```

C). 关于 xy 坐标或者 bBox 边界 list 的访问

在版图设计中，关于坐标的 list 是最长见的，它是一组 2 维的 list，常见的表示方法有：

用 : 表示一个坐标的list，其结果和list命令一样，用xCoord和yCoord命令可以访问xy坐标

```
xValue = 300
yValue = 400
aCoordinate = xValue:yValue => ( 300 400 )
xCoord( aCoordinate ) => 300
yCoord( aCoordinate ) => 400
```

用list命令和 ' 来表示一个bBox，list命令先计算变量或者表达式，然后赋给list，' 表示的list和字面的一样，不会计算变量或者表达式的值

```
bBox = list( 300:400 500:450 )    ; 含有 : 的bBox
含有变量用list
lowerLeft = 300:400
upperRight = 500:450
bBox = list( lowerLeft upperRight )
' 表示的list严格按字面意思
bBox = '(( 300 400 ) ( 500 450 ))
```

通过 car 和 cdr 的组合可以访问 bBox 每一个元素，而且有相关的简化函数，如下表：

Using car and cdr with Bounding Boxes			
Functions	Meaning	Example	Expression
car	car(...)	lower left corner	ll = car(bBox)
cadr	car(cdr(...))	upper right corner	ur = cadr(bBox)
caar	car(car(...))	x-coord of lower left corner	llx = caar(bBox)
cadar	car(cdr(car(...)))	y-coord of lower left corner	lly = cadar(bBox)
caadr	car(car(cdr(...)))	x-coord of upper right corner	urx = caadr(bBox)
cadadr	car(cdr(car(cdr(...]	y-coord of upper right corner	ury = cadadr(bBox)

List的相关操作有很多，这里就不详细介绍了，可以参考User Guide里的Advanced List Operations

3.4. Skill 的输入输出

1). 输出显示数据：

print 和 *println* 函数都可以用来显示单个数据，*println*可以在显示的数据后多加一个回车

```
for( i 1 3 print( "hello" )) ;Prints hello three times.
"hello""hello""hello"
for( i 1 3 println( "hello" )) ;Prints hello three times.
"hello"
"hello"
"hello"
```

printf 函数是格式化的输出，下面的例子是一定格式输出图形层的统计

```
printf( "\n%-15s %-15s %-10d %-10d %-10d %-10d"
layerName purpose rectCount labelCount lineCount miscCount
)
```

对应参数的意义如下，*printf* 需要注意输出类型的对应

```
%[-][width][.precision]conversion_code
[-] = left justify
[width] = minimum number of character positions
[.precision] = number of characters to be printed
conversion_code :
d - decimal(integer)
f - floating point
s - string or symbol
c - character
n - numeric
L - list (Ignores width and precision fields.)
P - point list (Ignores width and precision fields.)
B - Bounding box list (Ignores width and precision.)
```

2). 输出数据到一个文件：*outfile* 定义输出接口文件，*print println fprintf* 输出到接口文件，*close* 关闭打开的接口，见下面的例子

```
myPort = outfile( "/tmp/myFile1" )
for( i 1 3
  println( list( "Number:" i) myPort )
)
close( myPort )
```

输出到文件 /tmp/myFile1.

```
("Number:" 1)
("Number:" 2)
("Number:" 3)
```

```
myPort = outfile( "/tmp/myFile2" )
for( i 1 3
  fprintf( myPort "Number: %d\n" i )   ; 注意printf函数不能输出到port
)
close( myPort )
```

输出到文件/tmp/myFile2.

```
Number: 1
Number: 2
Number: 3
```

3). 从文件读取数据：*intfile* 定义输入接口文件，*gets* 一次从接口文件读取一行字符串，*fscanf* 根据指定的格式从接口文件读取，*close* 关闭打开的接口

打开~/.cshrc，输出文件的每一行

```
inPort = infile( "~/.cshrc" )
when( inPort
  while( gets( nextLine inPort )
    println( nextLine )
  )
  close( inPort )
)
```

打开~/.cshrc，输出文件中的每一个字符串

```
inPort = infile( "~/.cshrc" )
when( inPort
  while( fscanf( inPort "%s" word )
    println( word )
  )
  close( inPort )
)
```

3.5. Skill 的控制结构

1). 关系操作符，如下表：

Sample Relational Operators				
Operator	Arguments	Function	Example	Return Value
<	numeric	lessp	3 < 5	t
			3 < 2	nil
<=	numeric	leqp	3 <= 4	t
>	numeric	greaterp	5 > 3	t
>=	numeric	geqp	4 >=3	t
==	numeric	equal	3.0 == 3	t
	string list		abc == "ABc"	nil
!=	numeric	nequal	abc != "ABc"	t
	string list			

2). 逻辑操作符，如下表：

Sample Logical Operators				
Operator	Arguments	Function	Example	Return Value
&&	general	and	3 && 5	5
			5 && 3	3
			t && nil	nil
			nil && t	nil
||	general	or	3 || 5	3
			5 || 3	5
			t || nil	t
			nil || t	t

SKILL中只有 *nil* 是假(FALSE)，其余的任何值都是真(TRUE)。

与/或逻辑操作只有在需要计算第二个表达式时，才计算第二个表表达式，比如&&操作，当第一个表达时为假时，就不会再计算第二个表达式，|| 操作，当第一个表达时为假时，才会再计算第二个表达式。

返回的结果是最后一个计算的表达式，因此与/或逻辑操作可以代替繁琐的if / when等控制语句，例如：C语言中的操作符， a>b ? c=a : c=b; 即c取a b中较大的一个Skill中没有类似的操作符，可以用下面语句来完成此操作：

```
if( a>b  then
   c=a
else
   c=b
)
```

也可以用逻辑操作符： c = (a>b)&&a || (a<b)&&b

当然Skill还提供的有max(a b …)的函数，举这个例子是为了说明&& || 可以代替if then else之类的控制语句

3). 控制语句：*if(…then…else….)，when，unless，case，cond，*循环语句 *for，forecah* 等，控制语句和 C 语言类似，都是先判断某个变量或者表达式是否为真，然后执行下面的操作，需要注意的是：

- 关键词(其实也是 skill 的函数)和左括号之间不能有空格，
- *if…then…else* 的 then 是不能省略的(除了只有一个 if，没有 else 的情况)，
- case 的判断可以是数字和字符串，也可以是它们组成的 list，但不支持变量和表达式
- 如果有很多判断语句，用 *cond* 代替 *if…then…else* 组合，代码比较清晰而且执行效率比较高，下面的两种代码是等效的。
- 把最可能出现的情况放在最前面，如果出现的几率都一样，把计算量最大的放在最后面，这样可以有效的提高代码效率。

```
cond(
  ( condition1 exp11 exp12 ... )
  ( condition2 exp21 exp22 ... )
  ( condition3 exp31 exp32 ... )
  ...
  ( t expN1 expN2 ... )
) ; cond

if condition1 then exp11 exp12 ...
  else if condition2 then exp21 exp22 ...
    else if condition3 then exp31 exp32 ...
      ...
      else expN1expN2 ....
```

for 和 *foreach* 是循环控制语句，*for* 和 C 语言中的基本一样，*foreach* 经常用于对 list 的每个元素作循环操作，每个循环依次把各个元素的值赋给一个变量，如下面的例子，你会注意到 *foreach* 的返回值是循环的 list

```
rectCount = lineCount = polygonCount = 0
shapeTypeList = '( "rect" "polygon" "rect" "line" )
foreach( shapeType shapeTypeList
  case( shapeType
    ( "rect" ++rectCount )
    ( "line" ++lineCount )
    ( "polygon" ++polygonCount )
    ( t ++miscCount )
  ) ;case
) ; foreach
=> ( "rect" "polygon" "rect" "line" )
```

4. Skill 语言中常用知识汇总列表

上面对 skill 语言作了一个大概的介绍，当然学习 skill 需要了解的很多，这里就不再一一作详细的介绍了，只是对一些需要注意的地方做一下总结，具体的还是要学习 User Guide

1）Skill 中的特殊字符(除了字母数字以及 _ 以外的其他字符)都有各自的含义，见下表，如果要用到这些字符的话，用 \ 可以去掉其特殊含义。

Special Characters in SKILL

Character	Name	Meaning
\	backslash	Escape for special characters
( )	parentheses	Grouping of list elements, function calls
[ ]	brackets	Array index, super right bracket
{ }	braces	Grouping of expressions using *progn*
'	single quote	Quoting the expression to prevent its evaluation
"	double quote	String delimiter
,	comma	Optional delimiter between list elements; also used within the scope of a backquoted expression to force the evaluation of the expression
;	semicolon	Line-style comment character
:	colon	Bit field delimiter, range operator
.	period	getq operator
+, –, *, /	arithmetic	For arithmetic operators; the /* and */ combinations are also used as comment delimiters
!,^,&,|	logical	For logical operators
<,>,=	relational	For relational and assignment operators; < and > are also used in the specification of bit fields
#	pound sign	Signals special parsing if it appears in the first column
@	"at" sign	If first character, implies reserved word; also used with comma to force evaluation and list splicing in the context of a backquoted expression
?	question mark	If first character, implies keyword parameter
`	backquote	Quoting the expression prevents its evaluation, with support for the comma (,) and comma-at (,@) operators to allow evaluation within backquoted forms
%	percent sign	Used as a scaling character for numbers
$	–	Reserved for future use

2）Skill 支持多种数据类型，除了基本整数型、浮点型、字符串、数组之外，还有灵活的 List，用来表示一个数据的集合体，symbol 相当与 C 语言中的变量，但是它有比变量更多的用法，对于输入输出有 I/O port 类型，具体如下表：

Data Types Supported by SKILL

Data Type	Internal Name	Single Character Mnemonic
array	array	a
Cadence database object	dbobject	d
floating-point number	flonum	f
any data type	general	g
linked list	list	l
integer or floating point number		n
user-defined type		o
I/O port	port	p
defstruct		r
relative object design (ROD) object	rodObj	R
symbol	symbol	s
symbol or character string		S
character string (text)	string	t
function object		u
window type		w
integer number	fixnum	x
binary function	binary	y

3）Skill 的数字表示，除了科学计数法(如：1.1e6)外，还可以再数后面加一比例因子，如：5.5u = 5.5e-6，具体见下表：

Scaling Factors

Character	Name	Multiplier	Examples
Y	Yotta	10^{24}	10Y [10e+25]
Z	Zetta	10^{21}	10Z [10e+22]
E	Exa	10^{18}	10E [10e+19]
P	Peta	10^{15}	10P [10e+16]
T	Tera	10^{12}	10T [10e+13]
G	Giga	10^{9}	10G [10,000,000,000]
M	Mega	10^{6}	10M [10,000,000]
k or K	kilo	10^{3}	10k [10,000]
%	percent	10^{-2}	5% [0.05]
m	milli	10^{-3}	5m [5.0e-3]
u	micro	10^{-6}	1.2u [1.2e-6]
n	nano	10^{-9}	1.2n [1.2e-9]
p	pico	10^{-12}	1.2p [1.2e-12]
f	femto	10^{-15}	1.2f [1.2e-15]
a	atto	10^{-18}	1.2a [1.2e-18]
z	zepto	10^{-21}	1.2z [1.2e-21]
y	yocto	10^{-24}	1.2y [1.2e-24]

4）关于 Skill 的自定义函数

定义方法：a. 大部分自定义函数用 procedure 就够了，如：

```
procedure( trAdd( x y )
printf( "Adding %d and %d ... %d \n" x y x+y )
    x+y
)              => trAdd   定义函数trAdd
trAdd( 6 7 )  => 13       调用函数trAdd
```

b. lambda 用来定义一个没有函数名的函数，它在一些很小的函数里很方便，例如：

```
signalList = '(
    ( nil strength 1.5 )
    ( nil strength 0.4 )
    ( nil strength 2.5 )
)
sort( signalList
  'lambda( ( a b ) a->strength <= b->strength )
  ;定义一个用来按signalList的strength属性排序的内置函数，它没有具体的函数名
)
```

参数：函数的参数是在调用函数时用来传递一定的参数，Skill的参数除了基本的参数传递功能外，还有其他的@option来控制参数传递的方法：

@rest, 调用函数时允许任意多个参数传递给函数，

```
procedure( trTrace( fun @rest args )
let( ( result )
  printf( "\nCalling %s passing %L" fun args )
  result = apply( fun args )
  printf( "\nReturning from %s with %L\n" fun result )
  result
) ; let
) ; procedure
```

调用函数及返回结果：

```
trTrace( 'plus 1 2 3 )   => 6 ;返回值
  =>Calling plus passing (1 2 3)
    Returning from plus with 6
```

@optional, 可以传递一个可选参数，否则会用默认值，

```
procedure( trBuildBBox( height width @optional
                        ( xCoord 0 ) ( yCoord 0 ) )
  list(
    xCoord:yCoord                ;;; lower left
    xCoord+width:yCoord+height )  ;;; upper right
) ; procedure
```

调用函数及返回结果：

```
trBuildBBox( 1 2 )      => ((0 0) (2 1))
trBuildBBox( 1 2 4 )    => ((4 0) (6 1))
trBuildBBox( 1 2 4 10)  => ((4 10) (6 11))
```

@key，可以指定参数是如何传递的，应此调用是参数的顺序不固定

```
procedure( trBuildBBox(
     @key ( height 0 ) ( width 0 ) ( xCoord 0 ) ( yCoord 0 ) )
     list(
       xCoord:yCoord                ;;; lower left
       xCoord+width:yCoord+height ) ;;; upper right
) ; procedure
trBuildBBox()                       => ((0 0) (0 0))
trBuildBBox( ?height 10 )           => ((0 0) (0 10))
trBuildBBox( ?width 5 ?xCoord 10 )  => ((10 0) (15 0))
```

其中*@key*和*@optional*不能同时出现在一个函数的定义中，另外的情况可以组合在一起：

```
procedure(functionname([var1 var2 ...]
     [@optional opt1 opt2 ...]
     [@rest r])
..
)
procedure(functionname([var1 var2 ...]
     [@key key1 key2 ...]
     [@rest r])
..
)
```

局部变量，Skill 中的变量默认都是全局变量，为了避免命名冲突，要尽量减少全局变量的使用，而采用局部变量。Skill 可以用 *let 和 prog* 定义局部变量，如下面的例子：

```
procedure( trGetBBoxHeight( bBox )
  let( ( ( ll car( bBox ) ) ( ur cadr( bBox ) ) lly ury )
    lly = cadr( ll )
    ury = cadr( ur )
    ury - lly
  ) ; let
) ; procedure
```

局部变量默认初始化为 *nil*，当然也可以初始化为别的值或表达式，表达式中不能有别的局部变量，prog 和 let 的区别在于，prog 支持函数 *go 和 return*，可以显示的循环和返回多个值，而 let 返回值是最后一个表达式的值，例子的返回值是 `ury - lly`. 除了有必要用 prog，一般用 let，更加简洁快速。

5）Skill 的算术逻辑操作

Skill 含有众多的算术逻辑操作符，如下表，各种操作符的优先级从高到低排列，一般数据访问操作优先级最高，其次是一元操作符，二元操作符优先级最低，每个运算符都有对应的函数。

Arithmetic and Logical Operators

Name of Function(s)	Synopsis	Operator
Data Access		
arrayref	a[index]	[ ]
setarray	a[index] = expr	
bitfield1	x<bit>	<>
setqbitfield1	x<bit>=expr	
setqbitfield	x<msb:lsb>=expr	
quote	'expr	'
getqq	g.s	.
getq	g->s	->
putpropqq	g.s = expr, g->s = expr	~>
putpropq	d->s, d->s = expr	
Unary		
preincrement	++s	++
postincrement	s++	++
predecrement	--s	--
postdecrement	s--	--
minus	-n	-
null	!expr	!
bnot	~x	~
Binary		
expt	n1 ** n2	**
times	n1 * n2	*
quotient	n1 / n2	/
plus	n1 + n2	+
difference	n1 - n2	-
leftshift	x1 << x2	<<
rightshift	x1 >> x2	>>
lessp	n1<n2	<
greaterp	n1>n2	>
leqp	n1<=n2	<=
geqp	n1>=n2	>=
equal	g1 == g2	==
nequal	g1 != g2	!=
band	x1 & x2	&
bnand	x1 ~& x2	~&
bxor	x1 ^ x2	^
bxnor	x1 ~^ x2	~^
bor	x1 | x2	
bnor	x1 ~| x2	|, ~|
and	rel. expr && rel. expr	&&
or	rel. expr || rel. expr	||
range	g1 : g2	:
setq	s = expr	=

Skill 还定义了不少预定义的函数，直接调用它们，不仅方便，而且运算的效率更高，下面是一些预定义函数的列表：

Predefined Arithmetic Functions

Synopsis	Result
General Functions	
add1(n)	n + 1
sub1(n)	n – 1
abs(n)	Absolute value of n
exp(n)	e raised to the power n
log(n)	Natural logarithm of n
max(n1 n2 ...)	Maximum of the given arguments
min(n1 n2 ...)	Minimum of the given arguments
mod(x1 x2)	x1 modulo x2, that is, the integer remainder of dividing x1 by x2
round(n)	Integer whose value is closest to n
sqrt(n)	Square root of n
sxtd(x w)	Sign-extends the rightmost w bits of x, that is, the bit field x<w-1:0> with x<w-1> as the sign bit
zxtd(x w)	Zero-extends the rightmost w bits of x, executes faster than doing x<w-1:0>
Trigonometric Functions	
sin(n)	sine, argument n is in radians
cos(n)	cosine
tan(n)	tangent
asin(n)	arc sine, result is in radians
acos(n)	arc cosine
atan(n)	arc tangent
Random Number Generator	
random(x)	Returns a random integer between 0 and x-1. If random is called with no arguments, it returns an integer that has all of its bits randomly set.
srandom(x)	Sets the initial state of the random number generator to x

SKILL和C语言在算术逻辑操作上的区别：

- Skill中增加了幂级数运算符(**).
- C语言中的取模运算符 “%” 由函数*mod*代替，如 mod(i j).
- C中的条件表达式的三元算符 “?” “:” ，在Skill中没有类似的操作符，可以用 *if/then/else* 控制结构来代替。
- Skill不支持C中的指针和地址算符 “*” and “&”.
- 按位操作符增加了函数nand (~&), nor (~|), and xnor (~^).

逻辑表达式的返回值是*nil*或者一个non-*nil*的值(通常是*t*)，它取决于表达式计算的结果是false还是true.

Skill 有很多判定函数，用来判断数据的类型，结构等等，他们大都是以 p 结尾，如下面两个表是常见的判定函数

Arithmetic Predicate Functions

Synopsis	Result
minusp(n)	*t* if n is a negative number, *nil* otherwise
plusp(n)	*t* if n is a positive number, *nil* otherwise
onep(n)	*t* if n is equal to 1, *nil* otherwise
zerop(n)	*t* if n is equal to 0, *nil* otherwise
evenp(x)	*t* if x is an even integer, *nil* otherwise
oddp(x)	*t* if x is an odd integer, *nil* otherwise

Type Predicates

Function	Value Returned
arrayp(g)	*t* if g is an array, *nil* otherwise
bcdp(g)	*t* if g is a binary function, *nil* otherwise
dtpr(g)	*t* if g is a non-empty list, *nil* otherwise (note that dtpr (*nil*) returns *nil*)
fixp(g)	*t* if g is a fixnum, *nil* otherwise
floatp(g)	*t* if g is a flonum, *nil* otherwise
listp(g)	*t* if g is a list, *nil* otherwise (note that listp(*nil*) returns *t*)
null(g)	*t* if g is *nil*, *nil* otherwise
numberp(g)	*t* if g is a number (that is, fixnum or flonum), *nil* otherwise
otherp(g)	*t* if g is a foreign data pointer, *nil* otherwise
portp(g)	*t* if g is an I/O port, *nil* otherwise
stringp(g)	*t* if g is a string, *nil* otherwise
symbolp(g)	*t* if g is a symbol, *nil* otherwise
symstrp(g)	*t* if g is either a symbol or a string, *nil* otherwise
type(g)	a symbol whose name describes the type of g
typep(g)	same as type(g)

6）Skill 中的格式化输出，printf 和 fprintf 可以对输出结果先作一定的处理，然后再以一定的格式输出，如 0.000005 表示成 5e-6，更简单易读。

Common Output Format Specifications

Format Specification	Type(s) of Argument	Prints
%d	fixnum	Integer in decimal radix
%o	fixnum	Integer in octal
%x	fixnum	Integer in hexadecimal
%f	flonum	Floating-point number in the style [-]ddd.ddd
%e	flonum	Floating-point number in the style [-]d.ddde[-]ddd
%g	flonum	Floating-point number in style f or e, whichever gives full precision in minimum space
%s	string, symbol	Prints out a string (without quotes) or the print name of a symbol
%c	string, symbol	The first character
%n	fixnum, flonum	Number
%L	list	Default format for the data type
%P	list	Point
%B	list	Box

7）Skill中的List操作总结，List是Skill中最重要的数据结构，对于List相关的操作也十分丰富，方便用户对List及其元素的各种操作，如建立、访问、排序、删除、搜索、转换等等，下表是部分操作的一个简介，具体详细介绍可以参考User Guide中的Advanced List Operations.

List Operations

Operation	Function	Non-destructive	Destructive
Altering List Cells	rplaca, rplacd		x
Accessing a List	nthelem, nthcdr, last	x	
Building a List	cons, ncons, xcons, append1	x	
	tconc, nconc, lconc		x
Reorganizing a List	reverse	x	
	sort, sortcar		x
Removing Elements	remove, remq	x	
	remd, remdq		x
Searching Lists	member, memq, exists	x	
Filtering Lists	setof	x	
Substituting	subst	x	
Traversal	mapc, map, mapcar, maplist, mapcan	x	
Traversal	mapcan		x

5. Skill 程序实例解析

下面以一个 PutOnGrid.il 的实例，来具体分析一个 Skill 程序，这个程序是用来把不在格点上的图形，移到格点上，这里对每段代码都作了详细的解释，大家可以作个参考。

```
/*******这里是多行注释开始标志，程序的开始一般写上一些介绍的信息，结束为******/
/*  我们画 45 度的 path 线时，中心线的点在格点上，拐角处图形肯定不在格点上，本程序是通过把 path 转换成 polygon，再对 polygon 的点做处理，CIW 中 load 此程序，选中 off grid 图形，运行 PutOnGrid(Grid), Grid 工艺规定的最小格点，如 0.005 */
;这里是单行注释，到行尾
;定义 OnGrid 函数
procedure( OnGrid( Point ) ;用 procedure 定义函数 OnGrid
  if( listp(Point) then
    ;如果 Point 是 list 的话，对 list 的每个元素递归重复 OnGrid()函数，
    ;lambda 定义一个没有函数名的函数
    ;mapcar 函数用来根据一定的函数转换一个 list，这里用递归的方法保留原来 list 的结构
    ;如'((1.2 2.6) (3.3 4.8)),如果 Grid=0.5,则结果为'((1.0 2.5) (3.5 5.0))
    mapcar( lambda( (x) OnGrid(x) ) Point)
```

```
  else
    ;如果是一个数字的话，则通过 Point/Grid 取整，然后再乘以 Grid，使图形的每个点移到格点上
    round((Point/Grid))*Grid
  );if then … else 语句结束
);procedure

;定义 PutOnGrid 函数
procedure( PutOnGrid( Grid )
let((SelectObj )   ;定义局部变量
SelectObj = geGetSelSet()   ;版图中选定图形的函数，返回包含所有选中图形信息的一个 list
foreach( Object SelectObj   ;循环对每个选定的图形操作
  cond(
    (Object~>objType == "path"  ;判定图形的类型是不是 path
       Polygon = leConvertShapeToPolygon(Object) ;path 转成 polygon
       ;Polygon 对应的 points 属性(是一个 list)中每个元素通过 OnGrid 函数移动到格点上，
       ;然后把转换后的 list 重新赋给 Polygon~>points 属性
       Polygon~>points = OnGrid(Polygon~>points)
     )
    (Object~>objType == "polygon"  ;判定图形的类型是不是 polygon
       Object~>points = OnGrid(Object~>points)
     )
    (Object~>objType == "rect"  ;判定图形的类型是不是 rect
       ;需要改变的是 rect 的 bBox 属性(这里和 polygon 的 points 不同)
       Object~>bBox = OnGrid(Object~>bBox)
     )
    (Object~>objType=="inst" || Object~>objType=="mosaic"
       ;判定图形的类型是不是 inst/mosaic，这里改变的是 inst/mosaic 的 xy 坐标
       Object~>xy = OnGrid(Object~>xy)
     )
   );cond  ;你可以加上其他的情况
);foreach
);let
);procedure 加上关键词结尾的注释是个良好的习惯
```

程序的主要代码就是上面的这些，如果你还想要更方便的图形界面方式，一个快捷键就可以操作的话，可以参考 Cadence User Interface 中的 Form 相关的内容。