call a dynamic library from J

a C library

```
void foo(int *x){x[0] += 3;}
int bar(int *a){return a[1];}
int baz(int *a, int b){return a[b];}
```

build as a dynamic library

Note: using MacOS in this example, so file extension is .dylib.

```
rm -f foobar.o foobar.dylib
gcc -c foobar.c
g++ -dynamiclib foobar.o -o libfoobar.dylib
```

call from J

The cd function returns a list of boxes. The first item is the return value from the foreign function call. The remaining values are the arguments which were given to cd. This seemed pointless to me at first, until I realized that C can't return arrays. Instead, you can pass a pointer to an array, modify the array inside the function, and examine the array after the function returns.

This "modify by reference" behavior is demonstrated by this first example:

```
a =: <0 1 2
'./libfoobar.dylib foo n *i' cd a


 0 3 1 2
```

Interestingly, the J value a was not actually changed, indicating that J has a second copy. Perhaps this is what JfC means by "adequate only for simple functions".

```
a =: <0 1 2
result =: './libfoobar.dylib foo n *i' cd a
(a);<result
```

```
 0 1 2  0 3 1 2
```

In the next example, the return value is a[1]:

```
'./libfoobar.dylib bar i *i' cd <1 2 3
```

```
 2 1 2 3
```

Pass multiple arguments to `cd` as a list of boxes:

```
'./libfoobar.dylib baz i *i i' cd 1 2 3;2
```

```
 3 1 2 3 2
```