

INNER INTERPRITER

NOTES

1. Please follow along with the simplified code in `./misc/inner-interpriter.c`
2. `[&&x]` syntax is "labels as values" GCC extension.

Words

Words are running code through a double redirect.

```
[word] -> [&&word] -> [code]
```

You run this by

```
> goto ***word
> (code)
```

Direct Threading

Direct threading is an array of words.

```
[a] [b] [c]
```

Iterating over them using a pointer.

```
[a] [b] [c]
^ip
```

You run this just the same, but the (code) now has iteration and a goto at the end.

```
> goto ***ip

[a]
> (code)
> ++ip;          // Iterate
> goto ***ip    // Run B

[b]
> (code)
> ++ip;          // Iterate
> goto ***ip;   // Run C

etc...
```

The `NEXT(xx)` macro is shorthand for the iterate-run structure.

i.e. `NEXT(++)` -> `goto ***(++ip)`

Indirect Threading

Indirect threading is an array of arrays.

```
[a] [b] [c]
```

```
[a] -> [x] [y] [z]
```

Using a stack to keep track of instruction pointer positions.

```
[a] -> [ . . . ]  
^ip          *push()
```

```
[a] -> [ . . . ]  
      ^ip...^ip *runs
```

```
[a] -> [ . . . ]  
^ip          *pop()
```

By putting a special code at the beginning and end of the array.

```
[a] -> [&&push] [...] [pop]
```

You still run this just the same.

```
> goto ***ip  
  
[&&push]  
  > PUSH(rstack, ip) // Save ip current position  
  > ip = *ip;        // Change the frame of reference  
  > NEXT(++);        // Run [...]  
  
[...]  
  > (...)  
  > (...)  
  > (...)  
  
[pop]  
  > ip = POP(rstack) // Resets the old ip  
  > NEXT(++);        // Run B
```