The “Clockwise/Spiral Rule”

By David Anderson

There is a technique known as the “Clockwise/Spiral Rule” which enables any C programmer to parse in their head any C declaration!
There are **three** simple steps to follow:

1. Starting with the unknown element, move in a spiral/clockwise direction; when encountering the following elements replace them with the corresponding english statements:
   - `[X]` or `[]` => **Array X size of...** or **Array undefined size of...**
   - `(type1, type2)` => **function passing type1 and type2 returning...**
   - `*` => **pointer(s) to...**
2. Keep doing this in a spiral/clockwise direction until all tokens have been covered.
3. Always resolve anything in parenthesis first!

Example #1: Simple declaration

```
     +-------+
     | +-+   |
     | ^ |   |
char *str[10];
 ^   ^   |   |
 |   +---+   |
 +-----------+
```

Question we ask ourselves: What is str?
“str is an...

- We move in a spiral clockwise direction starting with ‘str’ and the first character we see is a ‘[‘ so, that means we have an array, so…
  “str is an array 10 of...
- Continue in a spiral clockwise direction, and the next thing we encounter is the ‘*’ so, that means we have pointers, so…
  “str is an array 10 of pointers to...
- Continue in a spiral direction and we see the end of the line (the ‘;’), so keep going and we get to the type ‘char’, so…
  “str is an array 10 of pointers to char”
- We have now “visited” every token; therefore we are done!

Example #2: Pointer to Function declaration

```
     +--------------------+
     | +---+              |
     | |+-+|              |
     | |^ ||              |
char *(*fp)( int, float *);
 ^   ^ ^  ||              |
 |   | +--+|              |
 |   +-----+              |
 +------------------------+
```

Question we ask ourselves: What is fp?
“fp is a...

- Moving in a spiral clockwise direction, the first thing we see is a ‘)’; therefore, fp is inside parenthesis, so we continue the spiral inside the parenthesis and the next character seen is the ‘*’, so…
  “fp is a pointer to...
- We are now out of the parenthesis and continuing in a spiral clockwise direction, we see the ‘(‘; therefore, we have a function, so…
  “fp is a pointer to a function passing an int and a pointer to float returning...
- Continuing in a spiral fashion, we then see the ‘*’ character, so…
  “fp is a pointer to a function passing an int and a pointer to float returning a pointer to...
- Continuing in a spiral fashion we see the ‘;’, but we haven't visited all tokens, so we continue and finally get to the type ‘char’, so…
  “fp is a pointer to a function passing an int and a pointer to float returning a pointer to a char”

Example #3: The “Ultimate”

```
      +-----------------------------+
      |                  +---+      |
      |  +---+           |+-+|      |
      |  ^   |           |^ ||      |
void (*signal(int, void (*fp)(int)))(int);
 ^    ^      |      ^    ^  ||      |
 |    +------+      |    +--+|      |
 |                  +--------+      |
 +----------------------------------+
```

Question we ask ourselves: What is ‘signal’?
Notice that signal is *inside* parenthesis, so we must resolve this first!

- Moving in a *clockwise* direction we see ‘(‘ so we have…
  “signal is a function passing an int and a...
- Hmmm, we can use this same rule on ‘fp’, so... What is fp? fp is also inside parenthesis so continuing we see an ‘*’, so…
  fp is a pointer to...
- Continue in a spiral clockwise direction and we get to ‘(‘, so…
  “fp is a pointer to a function passing int returning…”
- Now we continue out of the function parenthesis and we see void, so…
  “fp is a pointer to a function passing int returning nothing (void)”
- We have finished with fp so let's catch up with ‘signal’, we now have…
  “signal is a function passing an int and a pointer to a function passing an int returning nothing (void) returning...
- We are still inside parenthesis so the next character seen is a ‘*’, so…
  “signal is a function passing an int and a pointer to a function passing an int returning nothing (void) returning a pointer to...
- We have now resolved the items within parenthesis, so continuing clockwise, we then see another ‘(‘, so…
  “signal is a function passing an int and a pointer to a function passing an int returning nothing (void) returning a pointer to a function passing an int returning...
- *Finally* we continue and the only thing left is the word ‘void’, so the final complete definition for signal is:
  “signal is a function passing an int and a pointer to a function passing an int returning nothing (void) returning a pointer to a function passing an int returning nothing (void)”

The same rule is applied for const and volatile. For Example:

```
const char *chptr;
```

- Now, what is chptr??
  “chptr is a pointer to a char constant”

How about this one:

```
char * const chptr;
```

- Now, what is chptr??
  “chptr is a constant pointer to char”

Finally:

```
volatile char * const chptr;
```

- Now, what is chptr??
  “chptr is a constant pointer to a char volatile.”

Practice this rule with the examples found in K&R II on page 122.
