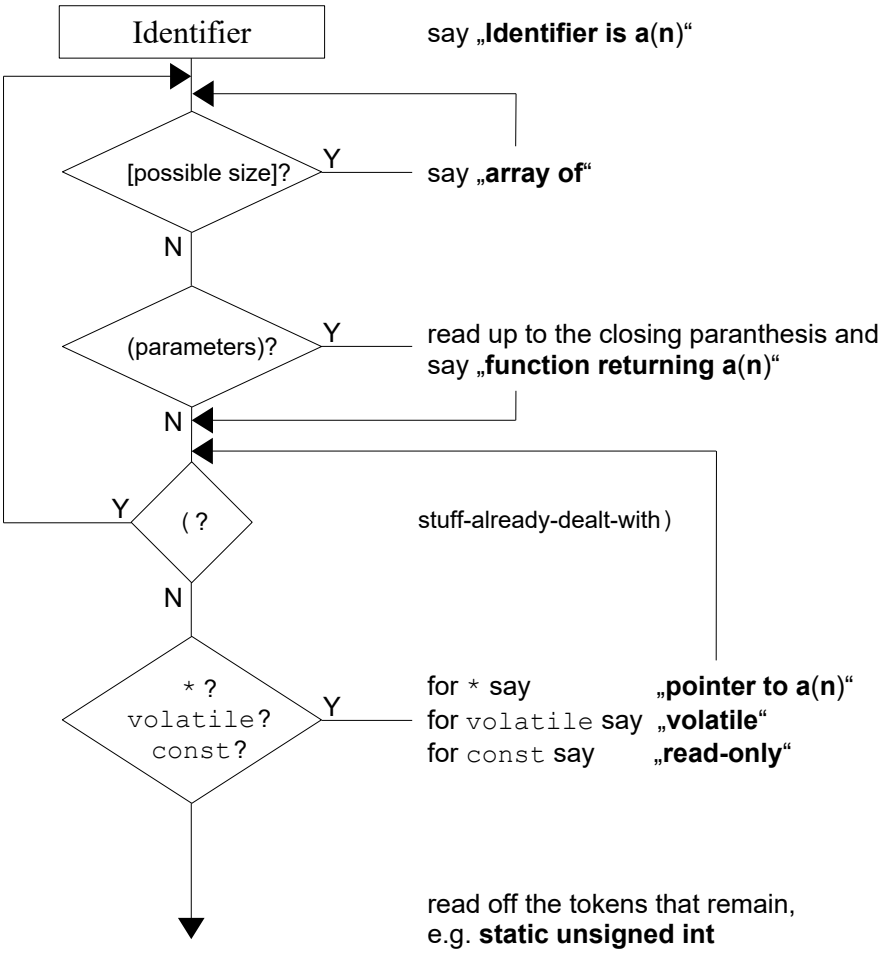
Decoding C Declarations[1]

Declarations in C are read boustrophedonically, i.e. alternating right-to-left with left-to right. And who'd have thought there would be a special word to describe that! Start at the first identifier you find when reading from the left. When we match a token in our declaration against the diagram, we erase it from further consideration. At each point we look first at the token to the right, then to the left. When everything has been erased, the job is done:

Step number	Token to match	How to read
1. Go to the leftmost identifier	Identifier	say „**Identifier is a**(**n**)“
2. Look at the next token to the **right** if it is a square bracket	[possible size]? Y	say „**array of**“
3. or if it is an opening parenthesis	(parameters)? Y	read up to the closing paranthesis and say „**function returning a**(**n**)“
4. If the token to the **left** is an opening parenthesis	(? Y	stuff-already-dealt-with)
5. If the token to the **left** is any of `*`, `volatile`, `const`	`* ? volatile? const?` Y	for `*` say „**pointer to a**(**n**)“ for `volatile` say „**volatile**“ for `const` say „**read-only**“
6. The tokens that remain form the basic type of the declaration		read off the tokens that remain, e.g. **static unsigned int**



Examples:

```
int i;
     ↑ i is an
 ↑ integer
```

```
int *i;
      ↑ i is a
    ↑ pointer to
 ↑ integer
```

1 Taken from the section titled “Magic Decoder Ring for C Declarations”, as can be found in chapter 3 of Peter van der Linden’s excellent book "Expert C Programming – Deep C Secrets", 1st Edition, 1994, Prentice Hall

```
int *i [3];
      ↑ i is an
        ↑ array of 3
    ↑ pointer to
 ↑ integer
```

```
int (*i) [3];
       ↑ i is a
      ↑ pointer to an
            ↑ array of 3
 ↑ integer
```

```
int *i (void);
      ↑ i is a
        ↑ function returning a
    ↑ pointer to
 ↑ integer
```

```
int (*i) (float f);
       ↑ i is a
      ↑ pointer to a
            ↑ function (taking an argument of type
                        float) returning an
 ↑ integer
```

```
int **i;
       ↑ i is a
      ↑ pointer to
    ↑ pointer to
 ↑ integer
```

```
int *(*i) (void);
       ↑ i is a
      ↑ pointer to a
            ↑ function returning a
    ↑ pointer to
 ↑ integer
```

```
int *(*i []) (void);
       ↑ i is an
         ↑ array of
      ↑ pointer to
              ↑ functions returning a
    ↑ pointer to
 ↑ integer
```

```
int *(*(*i)(void)) [10];
          ↑ i is a
        ↑ pointer to a
            ↑ function returning a
      ↑ pointer to an
                ↑ array of 10
    ↑ pointer to
 ↑ integer
```

```
int *(*i [5]) [10];
       ↑ i is an
         ↑ array of 5
      ↑ pointer to
              ↑ arrays of 10
    ↑ pointer to
 ↑ integer
```