cdecl

The reason C declarations are so hard to read is that they don't read from left to right like English. They read **inside out**.

Here is the "Right-Left Rule" (sometimes called the Spiral Rule). Once you learn this algorithm, reading any declaration—even "boss level" ones like **(k)** in your image—becomes a mechanical process rather than a guessing game.

The Golden Rule: "Right, then Left"

Start at the variable name, then follow these steps repeatedly:

1. **Look RIGHT**: Do you see brackets `[...]` or parentheses `(...)`?

- `[...]` means "Array of..."
- `(...)` means "Function returning..."
- *Note: Always consume the entire bracket/paren contents before moving on.*

1. **Look LEFT**: Do you see an asterisk `*`?

- `*` means "Pointer to..."

1. **Stop at Parentheses**: If you hit a closing parenthesis `)`, jump out to the matching opening parenthesis `(` and continue.
2. **Final Step**: Say the base type (e.g., `int`, `char`).

The Warm-ups

a) `int f;`

1. **f**: "f is..."
2. **Right**: (Nothing)
3. **Left**: `int` "...an integer."

- **Result:** f is an integer.

b) `int *f;`

1. **f**: "f is..."
2. **Right**: (Nothing)
3. **Left**: `*` "...a pointer to..."
4. **Left**: `int` "...an integer."

- **Result:** f is a pointer to an integer.

c) `int *f();`

1. **f**: "f is..."
2. **Right**: `()` "...a function returning..."
3. **Left**: `*` "...a pointer to..."
4. **Left**: `int` "...an integer."

- **Result:** f is a function returning a pointer to an integer.

The Parenthesis Traps

d) `int (*f)(int);`

1. **f**: "f is..."
2. **Right**: Hit `)`. Stop.
3. **Left**: `*` "...a pointer to..."
4. *(Jump out of parens)*
5. **Right**: `(int)` "...a function (taking an int) returning..."
6. **Left**: `int` "...an integer."

- **Result:** f is a pointer to a function returning an integer.

e) `int *(*f)(int);`

1. **f**: "f is..."
2. **Right**: Hit `)`. Stop.
3. **Left**: `*` "...a pointer to..."
4. *(Jump out of parens)*
5. **Right**: `(int)` "...a function (taking an int) returning..."
6. **Left**: `*` "...a pointer to..."
7. **Left**: `int` "...an integer."

- **Result:** f is a pointer to a function returning a pointer to an integer.

The Function Argument (Special Case)

f) `int f(int (*fp)(int*, int*));`
This looks scary, but just focus on `f` *first.*

1. **f**: "f is..."
2. **Right**: `( ... )` "...a function (taking arguments...) returning..."
3. **Left**: `int` "...an integer."

- **Result:** f is a function returning an integer.
  (The argument `fp` *is a pointer to a function taking two int pointers and returning an int).*

Arrays vs Pointers

g) `int *f[];`

1. **f**: "f is..."
2. **Right**: `[]` "...an array of..."
3. **Left**: `*` "...pointers to..."
4. **Left**: `int` "...integers."

- **Result:** f is an array of pointers to integers.

h) `int (*f)[5];`

1. **f**: "f is..."
2. **Right**: Hit `)`. Stop.

3. **Left**: `*` "...a pointer to..."
4. *(Jump out of parens)*
5. **Right**: `[5]` "...an array of 5..."
6. **Left**: `int` "...integers."

- **Result:** f is a pointer to an array of 5 integers.

The Complex Ones

i) `int (*f[])();`

1. **f**: "f is..."
2. **Right**: `[]` "...an array of..." *(Inside the parens, `[]` beats ``)*
3. **Left**: `*` "...pointers to..."
4. *(Jump out of parens)*
5. **Right**: `()` "...functions returning..."
6. **Left**: `int` "...integers."

- **Result:** f is an array of pointers to functions returning integers.

j) `int *(*f[])();`

1. **f**: "f is..."
2. **Right**: `[]` "...an array of..."
3. **Left**: `*` "...pointers to..."
4. *(Jump out of parens)*
5. **Right**: `()` "...functions returning..."
6. **Left**: `*` "...a pointer to..."
7. **Left**: `int` "...an integer."

- **Result:** f is an array of pointers to functions returning pointers to integers.

The Final Boss

k) `int (*(*f[3])())[5];`

1. **f**: "f is..."
2. **Right**: `[3]` "...an array of 3..."
3. **Left**: `*` "...pointers to..."
4. *(Jump out of inner parens)*
5. **Right**: `()` "...functions returning..."
6. **Left**: `*` "...a pointer to..."
7. *(Jump out of outer parens)*
8. **Right**: `[5]` "...an array of 5..."
9. **Left**: `int` "...integers."

- **Result:** f is an array of 3 pointers to functions returning a pointer to an array of 5 integers.