In the programming language C, a variable is nothing but a name given to a storage area that programs can manipulate. Each variable in C has a specific type, which determines the size and layout of the variable's memory, the range of values that can be stored within that memory, and the set of operations that can be applied to the variable. We will look at the following basic types:

**char** Denoting a single character. For example, “a”.

**int** Denoting an integer. For example, 5.

**float** Denoting a decimal number. For example, 7.89

**double** Denoting a precise decimal number; like a float, but with more significant digits

**void** Represents the absence of type

A simple variable definition takes the form **<type> <variable name>**. For example:

**int bar** denotes an integer called bar

**float foo** denotes a float called foo

The "basic types" are augmented with "derived types", and C has three of them.

The first is functions. A function is a group of statements that together perform a task. Functions process input(s), and have an output.

In c, function declarations take the form **<return type> <function name> <parameter list>**. The return type is the data type (float, int, etc) that the function returns. A void return type means that nothing is returned. The parameter list is a list of variables input into the function. So,

**int max(int num1, int num2)**

Denotes a functions with two integer inputs (num1 and num2), and an integer output.

These inputs do not need to be named, however, so **int max(int, int)** is also valid.

The second derived type is the array. An array is a collection of variables of same type. They are denoted with square brackets around a number. This number is the number of elements in the array. So,

**int x[10]** is an array of 10 integer

**float y[25]** is an array of 25 decimal numbers

The last is the pointer. In C, every variable has a location associated with it (similar to a postal address). When a pointer to an integer is declared, it does not directly hold an integer, but instead holds the location of an empty integer. So, the pointer “points” to an integer, similar to how a postal address “points” to a house. Pointers are denoted with an asterisk like so

**char\* tac** Denotes that tac is a pointer to a character

The difficulty in C is with multiple levels of these derived types. So, for example, arrays of pointers to functions.

To work these out, there are three steps to follow:

1. Starting with the unknown variable name, move in a spiral/clockwise direction; when encountering the following elements replace them with the corresponding english statements:

[X

=> Array X size of

(type1, type2)

=> function passing type1 and type2 returning...

\*

=> pointer(s) to...

1. Keep doing this in a spiral/clockwise direction until all tokens have been covered.
2. Always resolve anything in parenthesis first!

**Example #1**

+-------+

| +-+ |

| ^ | |

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**

+--------------------+

| +---+ |

| |+-+| |

| |^ || |

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''

With some variables, a keyword is used. The only keyword we will look at is **const**. This denotes that something cannot change. So, for example:

**const int x** means that x is an integer which cannot changes

The same rules we saw also work for keywords. We

**Example #3**

const char \*chptr;

* Now, what is chptr??

``chptr is a pointer to…

* Continuing…

``chptr is a pointer to a char constant

So chptr is a pointer to a constant char

**Example #4**

char \* const chptr;

* Now, what is chptr??

``chptr is a constant pointer to…

* Now, what is chptr??

``chptr is a constant pointer to a char