|  |
| --- |
| **Given the following complex C declarations, what does each one mean? Also, what are some rules for reading complex C declarations?** |
|  |
| |  | | --- | | 1.) int (\*p)[ 5 ];  2.) char \*x[ 3 ][ 4 ];  3.) int \*(\*a[ 10 ])();  4.) int (\*t[])(); |  |  | | --- | |  |   C declarations can sometimes be quite complex to read and to understand. I remember sometimes getting a headache just looking at some of the C declarations in other people’s code until I realized that there is a rule that can translate the declarations into plain and simple English. Interviewers also love to test out the ability of candidates to break down and understand those declarations. The way to approach this problem is to use that general rule that will help you read any C declaration – otherwise, you can easily find yourself very confused. Here is a good rule to follow when trying to make sense of a C declaration:  A Rule for Reading Complex C Declarations  ***Start at the variable name, also known as the identifier. Without skipping a right parenthesis, look right and say what you see – even if you see nothing continue. Without skipping a left parenthesis look left again and say what you see. If you are currently inside a pair of parentheses, jump outside this level. Look right and say what you see. Look left and say what you see. Keep doing this until you say the variable type or return type.***  ***An important note: you should always remember that functions and arrays are read left-to-right and have precedence over pointers, which are read right-to-left.***  Before reading the answers, it is best to use the rule given, and try it out for yourself. For the first example, we start at p. We look right and there is a parenthesis, which we don’t want to skip until we look left. So we look left and get that p is a pointer. Now, we can jump outside the parentheses, and we look right and find an array of size of 5, so now we can add to our statement that p is a pointer to an array of size 5. Now, we look left and find the ‘int’, so we can finish our declaration by saying that p is **"a pointer to an array of 5 ints"**.  For the second example, we start with x. We look right and find an array composed of 3 arrays of size 4. You may have been tempted to jump back left to the pointer after reading out "array of size 3", but you must note that arrays take precedence over pointers, so continuing on with the second array is important. So now we see that x is an array composed of 3 arrays of size 4, and the next step is to move left. We see a pointer, so now we can say that **"x is an array composed of 3 arrays of 4 pointers to char"**.  For the last two examples, we’ll just be giving the answers, since you should be able to figure them out on your own now – try figuring them out on your own before reading any further. In the third example, a is **"an array of 10 pointers to functions returning an int pointer"**. For the fourth example, t is **"an array of pointers to functions returning an int"**. |