int (*func)(int, int) = get_op_func(operator); how we pass operator to pointer to func

The expression `int (*func)(int, int) = get_op_func(operator);` is performing a few different actions:

1. Declaration of a Function Pointer: `int (*func)(int, int)` is declaring a variable named `func` that is a pointer to a function that takes two `int` parameters and returns an `int`.
2. Assignment: The `=` part is assigning a value to the `func` variable.
3. Function Call: `get_op_func(operator)` is calling the function `get_op_func` and passing `operator` as an argument to it.

Let's break down what happens when you call `get_op_func(operator)`:

- `get_op_func` is defined to take a `char *s`, which is a string or a string literal representing the operator. It's expected to be one of the strings `"+"`, `"-"`, `"*"`, `"/"`, or `"%"`.
- Inside `get_op_func`, it uses the provided `operator` to determine which function to return. It does this by looping through an array of `op_t` structures until it finds a structure where the `op` member matches the provided `operator`.
- When it finds a match, it returns the `f` member of that `op_t` structure. This `f` member is a pointer to a function that performs the corresponding operation.
- If no match is found, `get_op_func` returns `NULL`.

So, after `get_op_func(operator)` is called, the `func` variable holds the address of the function that corresponds to the operator provided. If `operator` was `"+"`, then `func` would point to the `op_add` function. If `operator` was `"-"`, then `func` would point to the `op_sub` function, and so on.

Once you have this function pointer in `func`, you can use it to call the function that it points to, like so:

```
int result = func(a, b);
```

Here, `a` and `b` would be `int` values that are the operands for the operation. The actual function that gets called will depend on what `operator` was passed to `get_op_func` in the first place, thereby determining which function address was stored in `func`.

int (*get_op_func(char *s))(int, int)
why we (int, int) it just operator

The notation `int (*get_op_func(char *s))(int, int)` is used in C to declare a function that returns a pointer to another function. It may look confusing at first, but let's break it down:

1. `get_op_func` is the name of the function you are declaring.
2. `(char *s)` indicates that `get_op_func` takes a single argument, which is a pointer to a `char` (typically this would be a string in C).
3. The return type of `get_op_func` is where it gets interesting: `int (*)(int, int)`.

This last part describes a pointer to a function that itself takes two `int` arguments and returns an `int`. The notation `int (*)(int, int)` can be read as:

- `(*)`: a pointer to
- `(int, int)`: a function that takes two integers as parameters
- `int`: and returns an integer

So when you see `int (*get_op_func(char *s))(int, int)`, it means:

- "`get_op_func` is a function that takes a string (`char *s`) and returns a pointer to another function. This other function takes two integers as arguments and returns an integer."

Here's an example of how you might use this in practice:

```c
// Assume these functions are defined somewhere
int add(int a, int b) { return a + b; }
int subtract(int a, int b) { return a - b; }

// This is how you might implement get_op_func
int (*get_op_func(char *s))(int, int) {
if (*s == '+') return add;
if (*s == '-') return subtract;
// ... additional checks for other operations
return NULL;
}

// And this is how you might use it
int main() {
char *op = "+";  // This is the operator we're using
// Get the function pointer for the operator
int (*operation)(int, int) = get_op_func(op);
if (operation) {
// Call the function through the pointer with arguments
int result = operation(3, 4);
printf("Result is: %d\n", result);
}
return 0;
}
```

In the example above, `get_op_func` returns a pointer to the `add` function when you pass it the `"+"` operator. You can then use this function pointer to call the `add` function indirectly through `operation(3, 4)`, which in this case will compute `3 + 4` and print `Result is: 7`.