5 Next, define the second custom function to call a regular function from a received function pointer, and pass it a received integer value

```
int caller( int (*function)(int), int b )
{
  return (*function)(b);
}
```

6 Back in the main function block, assign a value to the integer variable by calling the regular function via the function pointer, and output the value it returns

```
num = (*fptr)(10);
printf( "Returned Value: %d\n" , num ) ;
```

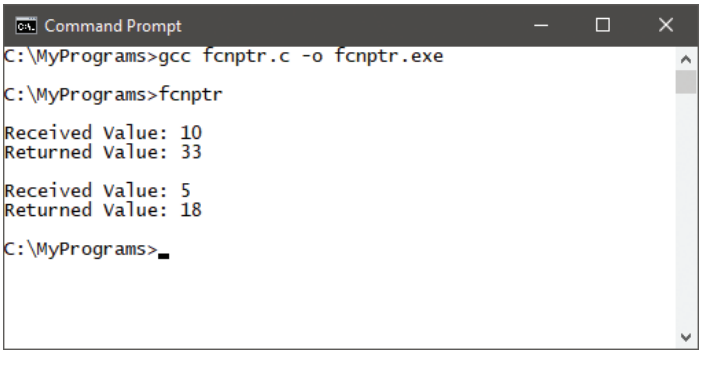
7 Now, assign a new value to the integer variable by passing the function pointer and an integer to another function, which in turn calls the regular function, then output the value it returns

```
num = caller( fptr, 5 ) ;
printf( "Returned Value: %d\n", num ) ;
```

8 At the end of the main function block, return a zero integer value, as required by the function declaration

```
return 0 ;
```

9 Save the program file, then compile and execute the program to see the values output via function pointers



The first argument to the **caller()** function in this example may be a pointer to any function that receives one integer argument and returns an integer value – as specified in the prototype declaration.

Like other pointers in C programming, a function pointer simply stores a memory address. When a function pointer is dereferenced with * the function at that address to which the pointer points gets called.