ENGG1003 - PASS Session 7

Mitchell Deltoer

Monday	**14-15**	**ES238**
Wednesday	**12-13**	**ES238**
Thursday	**10-11**	**MCLG42**

Pointers

A *pointer* is a variable whose value is the address of another variable. The syntax for declaring a pointer variable is

```
data_type *var_ptr;
```

where `data_type` is the base data type which the pointer is pointing at and `var_ptr` is the name of the pointer variable.

You have already seen *pointers* before when using `scanf`. Using a variable name with an ampersand (`&`) before it accesses the address of the variable rather than the value that it holds. So an example pointer declaration would look something like:

```
int var  = 12;       // initialising an integer variable with the value 12
int *var_ptr = &var; // initialising a pointer-to-int with the address of var,
                     // i.e. var_ptr is pointing at var
```

In the context of pointers, using a star (`*`) before a pointer variable name *de-references* a pointer. *De-referencing* is just jargon for saying "use the value of the variable located at the address being pointing at". **Continuing from the previous code segment**, here are some examples of *pointer de-referencing*:

```
*var_ptr = 15;             // the value of var is now 15
(*var_ptr)++;              // increase the value of var by 1
*var_ptr = *var_ptr - 6;   // subtract 6 from var and store it at var
printf("The value of var is now: %d\n", *var_ptr); // print what var_ptr is
pointing at
```

For all intents and purposes, a *de-referenced* pointer can be used anywhere that you would use a regular variable; it just makes it look a little bit more complicated.

The most common use for *pointers* in the context of this course is using them in functions.

Pointers and Functions

Functions in C can only *return* at most one value back to where it was called, i.e. it can only modify one value at a time in the main function. To get around this, we can use pointers as arguments to functions. When pointers are sent to functions, we are sending the address of the original variable, so when we modify what the pointer is pointing at, we are actually modifying the original variable itself. The following code segment is a short example of using pointers in a function to modify the value of more than one variable.

```
// my function with two pointers to int as input
void my_function(int *x, int *y);

int main(void){
   int a, b;
   // calling my function and sending the address of two ints to the pointers
   my_function(&a, &b);
   printf("a: %d, b: %d\n", a, b); // confirming the values have changed
   return 0;
}
void my_function(int *x, int *y){
   *x = ??;    // here you would modify the values by
   *y = ??;    // de-referencing the pointers
   return;
}
```

Practice Programming

Some things to consider when using functions within your program:

- What *arguments* (inputs) are there to the function? What are their data types?
- What is the *return value* (output) of the function? What is its data type?
- Have I remembered to (correctly) include the *function prototype* and the *function definition*?

Task 1: Swap Function Using Pointers

Write a C program that uses a function to swap the values of two int type variables. Your function should make use of pointers as function arguments to accomplish this task.

Task 2: Temperature Function

Write a C program that uses a function to convert a temperature in Celsius to both Kelvin and Fahrenheit. Your function should make use of pointers as function arguments to accomplish this task.

$$F = \frac{9}{5}C + 32, \qquad K = C + 273$$

Task 3: Array Statistics

Write a C program that uses a function to calculate both the sum and average value of an array. Your function should make use of pointers as function arguments to accomplish this task.