ECM2433: The C family

C Workshop 2

Exercises

1. Loops and functions

Create two different functions to calculate the factorial of a given integer (*n*! for a given number *n*). The first function should do this using a loop; the second should do it using recursion. The value of n! should be the functions' return value.

2. Arrays and structures

You have three products, with associated codes, prices and quantities, as follows:

```
Product description   Code    Price Quantity
--------------------- ---- -------- --------
laptop                 A     499.99        1
carrying case          B      12.50        2
pencils                C       0.09      130
```

(a) Define a structure to hold the values for one product.

(b) Define a variable *productList* as an array of these structures and populate it with the values shown.

(c) Create a procedure called ***PrintProduct*** to print out one element of this array in the format shown. Call it once for each element of the array to print out all the product details.

(d) Create a procedure called ***PrintProductList*** that takes the *productList* array as a parameter, prints out the report header showing the column titles and calls the ***PrintProduct*** procedure to print out each element.

(e) Add a new column to the report to display the number of characters in the product description. This value should be calculated from the existing data at the time the report is generated, not included as a new field in the structure. The new report should look like this:

```
Product description   Code    Price Quantity   Count
--------------------- ---- -------- --------   -----
laptop                 A     499.99        1   6
carrying case          B      12.50        2   13
pencils                C       0.09      130   7
```

3. Basic pointers

Create a new program to do the following:

(a) Define an integer variable called `x` and initialise it with the value 4.

(b) Print out the value of `x`.

(c) Print out the memory address of where `x` is stored.

(d) Define `xPtr` as a pointer to `x`.

(e) Update the value of `x` to 6 using only `xPtr`.

(f) Print out the new value of `x` using only `xPtr`.

(g) Print out the memory address of where `xPtr` is stored.

Run this program several times; you will see that the memory addresses are not the same each time.

4. Arrays and pointers

Create a new program to do the following:

(a) Define an array of `unsigned short int` values called `intArray` and initialise its values to all the even numbers between 2 and 20. Print out a list of all the elements of `intArray`, with their associated memory addresses.

(b) Print out a list of all the elements of `intArray` and their associated memory addresses using a pointer instead of the array subscripts.

5. Functions: pass by reference

Write a function called **updateInteger** which updates the value of an integer passed to it as a parameter by adding 3 to it. The parameter will have to be a pointer to the integer in order for the value to be able to be updated.

6. Dynamic memory allocation: malloc and free

Write a program that expects an integer value to be passed as a program parameter and then creates an array to hold exactly that number of `short int` values (hint: use the `malloc` function). Populate the array with the numbers from 1 to the number entered and then print them out. For example if your program is called myProgram, then running it using the command

```
myProgram 3
```

should cause an array of size 3 to be created, with the values 1, 2 and 3. Remember to explicitly free the memory before your program closes.