ECM2433: The C Family

C Worksheet 3a: pointers and functions

Exercises

1. 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.

2. Passing parameters on the command line

The main function is supplied with any command line parameters that were specified when the program is executed. The prototype for the main function is:

```
int main(int argc, char **argv);
```

Here `argc` is the number of parameters supplied to the program, including the name of the program itself and `argv` is an array of strings, that is a pointer to pointers to `char`. It could equally well be declared as `char *argv[]` or `char (* argv)[]` which makes it clearer that it is an array of pointers to char, that is an array of strings. When the program is invoked, `argv` contains the command line parameters as an array of strings, starting with the name of the program. So, for example,

```
myprogram  5 Hello world
```

will result in `argc = 4` and `argv` being the array:

```
argv = { "myprogram",
         "5",
         "Hello",
         "World"
       };
```

Note that all the command line arguments are strings, even if you would like them to represent numbers. The functions `atoi()` and `atof()` are useful for converting strings to integers and floats respectively.

Write and test a program to print its command line arguments.

Write another program like `myprogram` above, whose first argument is a number that specifies how many times the remaining arguments will be printed. So, invoking it as `myprogram 3 Hello world` would result in the output:

Hello world
Hello world
Hello world
Hello world
Hello world

Make sure that your program behaves sensibly if the first argument is 0 or negative, or if there are no further arguments. Another edge case is if the first argument is not a string representing an integer (eg. `myprogram foo Hello`); work out and test how to do something sensible in this case.

3. Functions: pass by value

Write a function called `printInteger` that takes a single integer as a parameter, adds 3 to it and then prints out its value. The return type of the function should be the updated value of the integer. Call your function like this:

```
int x = 42;
int y = 0;

printf("in main function x = %d, y = %d\n",x,y);
y = printInteger(x);
printf("in main function x = %d, y = %d\n",x,y);
```

The value of y should be 0 from the first `printf` and 45 from the second.

Remember to include a function prototype in your program – this declares the function in the same way that variables are declared.

4. Functions: pass by reference

Write a function called `updateInteger` that 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. Call your function like this:

```
int x = 42;

printf("in main function x = %d\n",x);
updateInteger(&x);
printf("in main function x = %d\n",x);
```

Think carefully about what is going on here. The argument to the function is a pointer, so the memory address of x in the calling program or function is available to `updateInteger`, which can therefore modify it. What are the implications for arrays as arguments, which are always pointers?

5. Dynamic memory allocation: malloc and free

Write a program that expects an integer value to be passed as a command line 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 executable is called `myProgram`, then running it using the command

```
myProgram 3
```

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

On the ELE page there are instruction for running your program through a utility called `valgrind`. Do this for your completed program. Then comment out the `free` statement, recompile the program and run it through `valgrind` again. Note how `valgrind` enables you to spot memory leakages.