Exercise 6 – Arrays

Objective

The objectives of this session are to consolidate your understanding of arrays in C and to appreciate the efficiency of code produced by the compiler in the presence and absence of an optimiser.

Reference Material

This chapter is based on material in the Arrays chapter. You might also like to refer back to the Pointers chapter to remind yourself about pointer arithmetic. This practical session is located in the following directory:

	Microsoft Windows	*Linux*
Directory:	**c:\qacadv\arrays**	**~/qacadv/arrays**
Solution directory:	**c:\qacadv\arrays\Solution**	**~/qacadv/arrays/Solution**

Overview

The first question asks you to measure the relative efficiency of various techniques for initialising an array. The second question is a pencil and paper puzzle to test your understanding of arrays of pointers.

Practical Outline

1. **On Microsoft Windows** open **arraytim.sln, on Linux** change your current working directory.

   Take a look at the code fragments and comments already written in **arraytim.c**. Your task is to measure the efficiency of three different methods for initialising the large character array `bigarray`:

   a) using array/index notation (i.e. `a[i] = '\0'`)

   b) using pointer/offset notation (i.e. `*p = '\0'`)

   c) using the standard `memset()` function

   Write three separate functions to initialise the array using these three different techniques. Call these functions from `main()` - we suggest calling each function 100 times (use the preprocessor define `REP_COUNT`) in order to increase the total elapsed time and to improve the accuracy of your results.

The single function `timer()` does the timing for you. As you will see from the code, it should be called with one of the three values `Start`, `Stop` or `Show` to respectively start the timer, stop the timer or display the elapsed time.

Your code needs to go between the call to start the timer and stop the timer. Don't forget to show the results or your program will be rather unhelpful.

When you have compiled the code, write down the timings for the three functions. Check the optimisation possibilities and try various optimisations (you should have quite a few choices). Note down the timings and see what difference is made.

On Linux the supplied makefile compiles five versions of arraytim:

arraytim	no-optimisation
arraytim.1	Level 1 optimisation
arraytim.2	Level 2 optimisation
arraytim.3	Level 3 optimisation
arraytim.s	Optimised for size

2. What is printed by the following program?

```
#include <stdio.h>

char *c[] =         /* What kind of variable is c? */

{
    "Two",
     "Enter",
     "Hell",
     "Harold"
};

char **cp[] =       /* What kind of variable is cp? */
{
     c + 3,
     c,
     c + 2,
     c + 1
};

char ***cpp = cp; /* What kind of variable is cpp? */
```

```
int main(void)
{
    printf("%s",   *cpp[2]);
    printf("%s ",  **++cpp + 2);
    printf("%s",   **cpp + 1);
    printf("%s",   *cpp[2] + 4);
    printf("%s\n",  cpp[1][1] + 4);

    return 0;
}
```

Hint: if you can’t wrap your mind around, say:

```
cpp[2]
```

remember that it is equivalent to:

```
*(cpp + 2)
```

Also, the most important thing here is to draw a diagram. The program is almost impossible to solve without it.

To check your answer, **on Microsoft Windows** open **quiz.sln** and build and run the program, **on Linux** run the program **quiz**. The correct answer is displayed on the screen. Did you get it right?