Exercise 3 – The Preprocessor

Objective

The major objective of this session is to practice using some of the advanced preprocessor facilities available in the C language.

Reference Material

This practical session is based entirely on material in The Preprocessor chapter. The session is located in the following directory:

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

Overview

The first question asks you to make use of the preprocessor operators `#` and `##`. The next three questions deal with conditional compilation and using the symbols `__FILE__` and `__LINE__`. The last question looks at how to construct "type-generic" functions to overcome some of the notorious problems with macros.

Practical Outline

On Microsoft Windows open **pre1.sln**, on Linux change your current working directory.

1. Load the (empty) file called **pre1.c** into your editor. Write a preprocessor macro to print an integer expression as a string, along with its value. As an example of the use of the macro, consider the following:

```
#include <stdio.h>
/* Definition of PEXP macro goes here... */
int main(void)
{
    int i=4, j=5;

    PEXP(i);
    PEXP(i+j);
    PEXP(i*j);

    return 0;
}
```

Using the macro would produce the following output:

```
i = 4
i+j = 9
i*j = 20
```

A model solution is available in **Solution** sub-directory, called **pre1.c**

2. Working in the same workspace, enhance the macro so that it can be "turned off" at compile time using a symbolic constant (e.g. `DEBUG`). If the debug mode constant is defined, then the macro works as before. If is not defined, then the macro must do nothing. A solution is available in **Solution** sub-directory, called **pre2.c**.

   On Microsoft Windows, the symbolic constant _DEBUG is defined automatically when using the **Debug** configuration. Change configurations to **Release**, then rebuild, if you do not want this symbol defined.

   On Linux, the makefile builds two programs, **pre1** and **pre1.debug**. The symbolic constant DEBUG is defined for **pre1.debug**, but not **pre1** .

3. Enhance the macro so that it prints not only the name and value of the expression, but also the line and file name from which the macro was invoked. In this case, the example program above will print results of the form:

```
EXAMPLE.C(9): i = 4
EXAMPLE.C(10): i+j = 9
EXAMPLE.C(11): i*j = 20
```

   The solution for this question is in **Solution** sub-directory, called **pre3.c**

4. As it stands, the `PEXP` macro is excellent at printing `int`s. How could it be modified such that it could print any type, `int`, `char`, `double`, etc.? You will need to modify the number of parameters to the macro. (Hint: what is it that tells `printf` to print an integer?). A solution is given in in **Solution** sub-directory, called **pre4.c**

5. This is the final question in this practical session.

On Microsoft Windows open **pre5.sln**, on Linux just edit the file pre5.c.

The most famous preprocessor macro must be `MAX`, which could be defined as follows:

```
#define MAX(A, B) (((A) > (B)) ? (A) : (B))
```

The excessive braces avoid potential operator precedence problems when the macro is invoked, for example:

```
x = MAX(a + b, b * 100) % 16;
```

As discussed in the preceding chapter, however, there is no protection against one of the following two side effects being executed twice:

```
x = MAX(a++, b--);
```

Write a macro `DEFINE_MAX(type)` which, if invoked as `DEFINE_MAX(int)`, would create the following function:

```
int  max_int (int a, int b)
{
    return (a > b) ? a : b;
}
```

When invoked as `DEFINE_MAX(double)` the following function will be created:

```
double  max_double (double a, double b)
{
    return (a > b) ? a : b;
}
```

Write a second macro, `CALL_MAX(type, a, b)` which, when called as `CALL_MAX(int, x+1, y*2)`, will invoke the `max_int()` function with the second and third parameters.

Test your macros with a variety of types and parameters. A solution is provided in the **Solution** sub-directory called **pre5.c**