Exercise 11 – Pointers and Functions

Objective

The major objective is to practice call by reference. A second objective is to gain an appreciation of returning pointers and the implication of doing so.

Reference Material

This is based mainly on the *Pointers and Functions* chapter, but clearly uses much of the basic concepts introduced in the *Pointers* chapter. The practical session is located in the following directory:

Windows Directory:	**c:\qacprogex\ptrfunc**
Windows Solution directory:	**c:\qacprogex\ptrfunc\solution**
Linux Directory:	**/home/user1/qacprg/PTRFUNC**
Linux Solution directory:	**/home/user1/qacprg/PTRFUNC/Solution**

Overview

Questions 1, 2 and 4(optional) use the call-by-reference technique. Questions 3 and 5 cover returning by pointer, with question 5 indicating possible danger areas.

Practical Outline

1. Open the Visual Studio Solution **average.sln**, and take a look at the code template in **average.c**. The `main()` function tries to invoke a function called `average()`, which takes three parameters; the first two parameters are `ints` and the third is a pointer to a `double`.

```
int main(void)

{
     int  num1 = 0,
          num2 = 0;

     double ave = 0.0;

     printf("Please type in two integers :\t");
     scanf("%d%d", &num1, &num2);

     average(num1, num2, &ave);

     printf("\nThe average of %d and %d is %.1f\n",
                num1, num2, ave);

     return 0;
}
```

Your task is to provide the prototype and definition for the `average()` function.

2. Open the Visual Studio Solution **larger.sln**, and take a look at the code template in **larger.c**. Using this code, write a function called `larger()` that has the prototype:

```
void larger(int *, int, int);
```

`larger()` should put the larger of the values passed (the second and third parameters) into the integer pointed to by first parameter. The following code prints 3, followed by 15 (on separate lines):

```
int main(void)
{
     int  in1 = 3,
          in2 = -2,
          out;

     larger(&out, in1, in2);
     printf("%d\n", out);

     larger(&out, in1, 15);
     printf("%d\n", out);

     return 0;
}
```

Optional:

3. Open the Visual Studio Solution **p_larger.sln**, and take a look at the code template in **p_larger.c**. Using this code, write a function called `p_larger()` that has the prototype:

```
int * p_larger(int *, int *);
```

The function returns a pointer to the larger of the values pointed to by its two pointer arguments, i.e. the following code displays `12`:

```
int main(void)
{
     int  x = 9,
          y = 12;
     int * p;

     p = p_larger(&x, &y);
     printf("%d\n", *p);

     return 0;
}
```

4. Open the Visual Studio Solution **fraction.sln**, and take a look at the code template in **fraction.c**. The program includes a `main()`, which calls a function called `fraction()` with the prototype:

```
void fraction(double, int *, double *);
```

The function creates the integer and fractional part of the `double` and assigns them to the `int` and `double` pointer, respectively.

The call, `fraction(3.14159, &whole, &part)` should place the integer 3 in `whole` and the `double 0.14159` in `part`.

Complete the program.

Hint:

Use the cast operator or the standard function `floor()` from the maths library.

Note: When satisfied, qualify the pointers in the parameters as `const`.

5. In Question 3, you implemented a working (hopefully) `p_larger()` function. What is wrong with these versions of `p_larger()`?

```
int * p_larger1(int * p1, int * p2)
{
     int i;

     i = (*p1 > *p2) ? *p1 : *p2;
     return &i;
}

int * p_larger2(int * p1, int * p2)
{
     int * ans;

     *ans = (*p1 > *p2) ? *p1 : *p2;
     return ans;
}
```