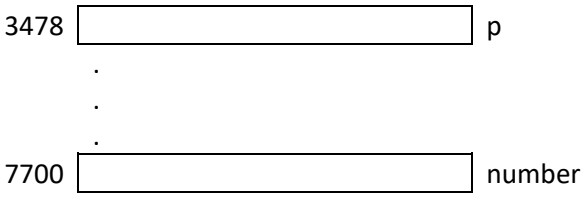
# Tutorial 2 – Functions and Pointers

1. Assume the following declaration:

```
int number;
int *p;
```

Assume also that the address of number is 7700 and the address of p is 3478. That is,




For each case below, determine the value of

(a) number (b) &number (c) p (d) &p (e) *p

All of the results are cumulative.

(i) p = 100; number = 8
(ii) number = p
(iii) p = &number
(iv) *p = 10
(v) number = &p
(vi) p = &p

2. **(digitValue)** Write a function that returns the value of the $k^{th}$ digit (k>0) from the right of a non-negative integer *num*. For example, if num is1234567 and k is 3, the function will return 5 and if num is 1234 and k is 8, the function will return 0. Write the function in two versions. The function **digitValue1()** returns the result, while **digitValue2()** passes the result through pointer parameter result. The prototypes of the function are given below:

```
int digitValue1(int num, int k);
void digitValue2(int num, int k, int *result);
```

A sample program template is given below to test the functions:

```
#include <stdio.h>
int digitValue1(int num, int k);
void digitValue2(int num, int k, int *result);
int main()
{
   int num, digit, result=-1;

   printf("Enter the number: \n");
   scanf("%d", &num);
   printf("Enter k position: \n");
   scanf("%d", &digit);
   printf("digitValue1(): %d\n",  digitValue1(num, digit));
   digitValue2(num, digit, &result);
```

```
    printf("digitValue2(): %d\n", result);
    return 0;
}
int digitValue1(int num, int k)
{
    /* Write your code here */
}
void digitValue2(int num, int k, int *result)
{
    /* Write your code here */
}
```

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Enter the number:
234567
Enter k position:
3
digitValue1(): 5
digitValue2(): 5
```

(2) Test Case 2:

```
Enter the number:
234567
Enter k position:
1
digitValue1(): 7
digitValue2(): 7
```

(3) Test Case 3:

```
Enter the number:
123
Enter k position:
8
digitValue1(): 0
digitValue2(): 0
```

3. **(extOddDigits)** Write a function that extracts the odd digits from a positive number, and combines the odd digits sequentially into a new number. The new number is returned to the calling function. If the input number does not contain any odd digits, then the function returns -1. For example, if the input number is 1234567, then 1357 will be returned; and if the input number is 24, then –1 will returned. Write the function in two versions. The function extOddDigits1() returns the result to the caller, while the function extOddDigits2() returns the result through the pointer parameter, result. The function prototypes are given as follows:

```
int extOddDigits1(int num);
void extOddDigits2(int num, int *result);
```

A sample program template is given below to test the functions:

```
#include <stdio.h>
#define INIT_VALUE 999
int extOddDigits1(int num);
```

```
void extOddDigits2(int num, int *result);
int main()
{
  int number, result = INIT_VALUE;

  printf("Enter a number: \n");
  scanf("%d", &number);
  printf("extOddDigits1(): %d\n", extOddDigits1(number));
  extOddDigits2(number, &result);
  printf("extOddDigits2(): %d\n", result);
  return 0;
}
int extOddDigits1(int num)
{
  /* Write your code here */
}
void extOddDigits2(int num, int *result)
{
  /* Write your code here */
}
```

Some sample input and output sessions are given below:

(1) Test Case 1:
Enter a number:
1234
extOddDigits1(): 13
extOddDigits2(): 13

(2) Test Case 2:
Enter a number:
2468
extOddDigits1(): -1
extOddDigits2(): -1

(3) Test Case 3:
Enter a number:
1357
extOddDigits1(): 1357
extOddDigits2(): 1357

(4) Test Case 4:
Enter a number:
5
extOddDigits1(): 5
extOddDigits2(): 5

4. **(calDistance)** Write a C program that accepts four decimal values representing the coordinates of two points, i.e. (x1, y1) and (x2, y2), on a plane, and calculates and displays the distance between the points:

$$\text{distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Your program should be implemented using functions. Provide two versions of the function for calculating the distance: (a) one uses call by value only for passing parameters; and (b) the other uses call by reference to pass the result to the calling function.

The function prototypes are given below:

```
void inputXY(double *x1, double *y1, double *x2, double *y2);
void outputResult(double dist);
double calDistance1(double x1, double y1, double x2, double y2);
void calDistance2(double x1, double y1, double x2, double y2,
                  double *dist);
```

A sample program template is given below to test the functions:

```
#include <stdio.h>
#include <math.h>
void inputXY(double *x1, double *y1, double *x2, double *y2);
void outputResult(double dist);
double calDistance1(double x1, double y1, double x2, double
y2);
void calDistance2(double x1, double y1, double x2, double y2,
double *dist);
int main()
{
   double x1, y1, x2, y2, distance=-1;

   inputXY(&x1, &y1, &x2, &y2);          // call by reference
   distance = calDistance1(x1, y1, x2, y2); // call by value
   printf("calDistance1(): ");
   outputResult(distance);
   calDistance2(x1, y1, x2, y2, &distance); // call by reference
   printf("calDistance2(): ");
   outputResult(distance);              // call by value
   return 0;
}
void inputXY(double *x1, double *y1, double *x2, double *y2)
{
   /* Write your code here */
}
void outputResult(double dist)
{
   /* Write your code here */
}
double calDistance1(double x1, double y1, double x2, double y2)
{
   /* Write your code here */
}
void calDistance2(double x1, double y1, double x2, double y2,
double *dist)
{
   /* Write your code here */
}
```

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Input x1 y1 x2 y2:
1 1 5 5
calDistance1(): 5.66
calDistance2(): 5.66
```

(2) Test Case 2:

```
Input x1 y1 x2 y2:
-1 -1 5 5
calDistance1(): 8.49
calDistance2(): 8.49
```