Functions and Pointers – Q1

(**numDigits**) Write a function that counts the number of digits for a non-negative integer. For example, 1234 has 4 digits. The function **numDigits1()** returns the result. The function prototype is given below:

```
int numDigits1(int num);
```
// call by value

Write another function **numDigits2()** that passes the result through the second parameter, *result*. The function prototype is given below:

```
void numDigits2(int num, int *result);
```
// call by reference

Write a C program to test the functions.

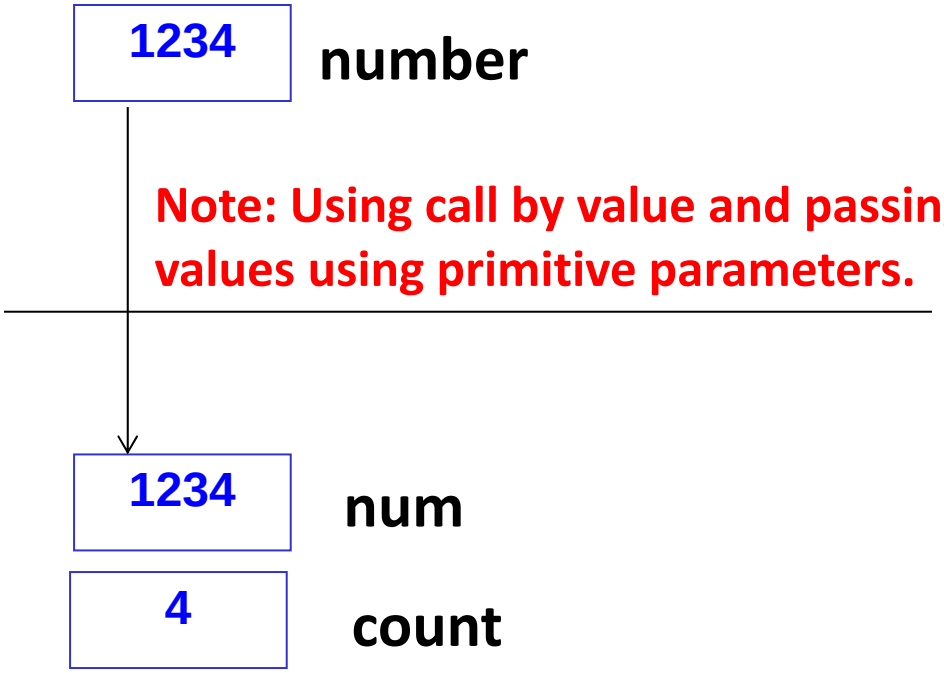
Sample input and output sessions:

```
(1)   Enter the number:
      1
      numDigits1(): 1
      numDigits2(): 1
(2)   Enter the number:
      13579
      numDigits1(): 5
      numDigits2(): 5
```

Q1 – Call by Value

```
#include <stdio.h>
int numDigits1(int num);
int main()
{
   int number;

   printf("Enter the number: \n");
   scanf("%d", &number);
   printf("numDigits1(): %d\n",
           numDigits1(number));
   return 0;
}
int numDigits1(int num)
{
   int count = 0;
   do {
      count++;
      num = num/10;
   } while (num > 0);
   return count;
}
```



In the loop:

- count=1 1234/10 -> 123
- count=2 123/10 -> 12
- count=3 12/10 -> 1
- Count=4 1/10 -> 0 then exit loop

Note:

- When programming with integer number, use % operator to get the remainder of a number, and / operator to get the quotient of the number. For example: 1234/10 -> 123; 1234%10 -> 4.

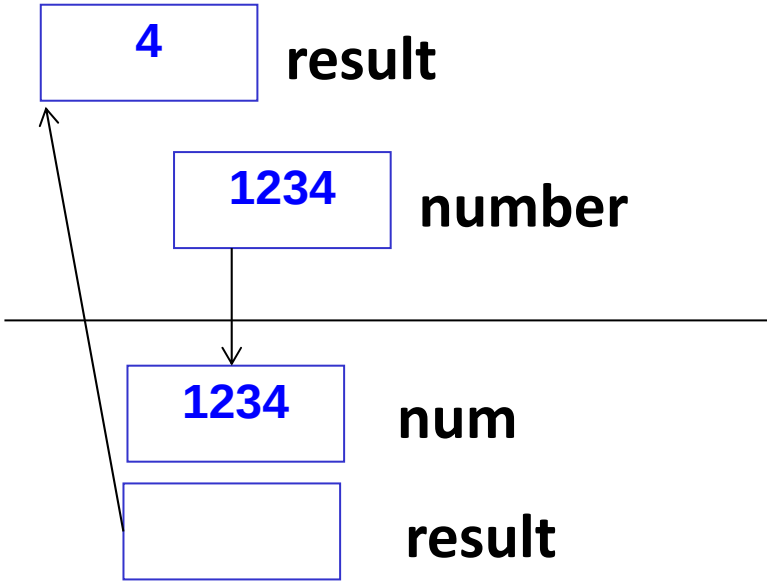
Q1 – Call by Reference

```
#include <stdio.h>
void numDigits2(int num, int *result);
int main()
{
   int number, result=0;

   printf("Enter the number: \n");
   scanf("%d", &number);
   numDigits2(number, &result);
   printf("numDigits2(): %d\n", result);

   return 0;
}
void numDigits2(int num, int *result)
{
   *result=0;
   do {
      (*result)++;
      num = num/10;
   } while (num > 0);
}
```

Note: Using call by reference and passing the address of the variable *result* to the called function.



Note: the parameter *result* in the function header is declared as a pointer parameter.

Functions and Pointers – Q2

(**digitPos**) Write the function **digitPos1()** that returns the position of the first appearance of a specified digit in a positive number. The position of the digit is counted from the right and starts from 1. If the required digit is not in the number, the function should return 0. For example, digitPos1(12315, 1) returns 2 and digitPos1(12, 3) returns 0. The function prototype is given below:

```
int digitPos1(int num, int digit);          // call by value
```

Write another function **digitPos2()** that passes the result through the third parameter, *result*. For example, if num = 12315 and digit = 1, then *result = 2 and if num=12 and digit = 3, then *result = 0. The function prototype is given below:

```
void digitPos2(int num, int digit, int *result);      // call by reference
```

Write a C program to test the functions.

Sample input and output sessions:

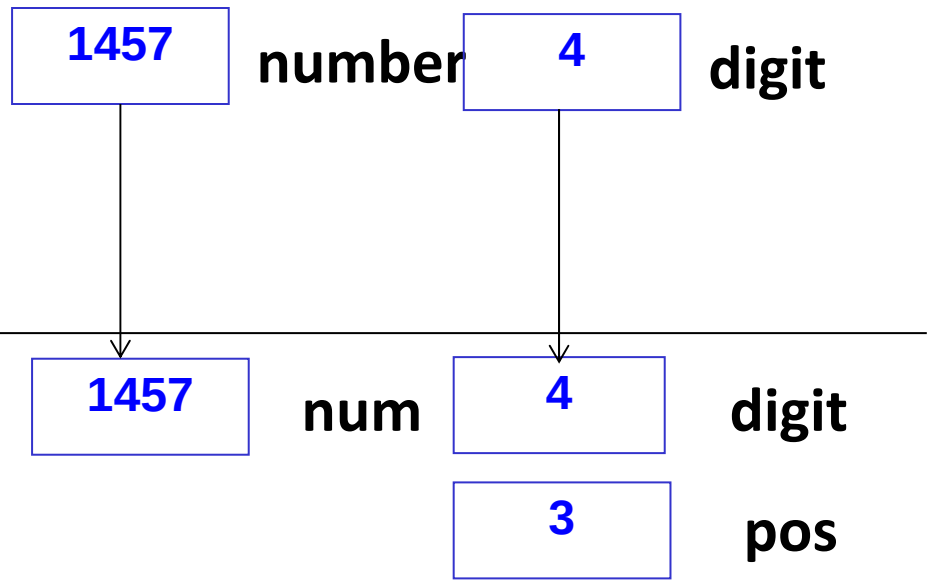
(1)
```
Enter the number:
1234567
Enter the digit:
6
digitPos1(): 2
digitPos2(): 2
```

(2)
```
Enter the number:
1234567
Enter the digit:
8
digitPos1(): 0
digitPos2(): 0
```

Q2 – Call by Value

```
#include <stdio.h>
int digitPos1(int num, int digit);
int main()
{
   int number, digit;

   printf("Enter the number: \n");
   scanf("%d", &number);
   printf("Enter the digit: \n");
   scanf("%d", &digit);
   printf("digitPos1(): %d\n",
      digitPos1(number, digit));
   return 0;
}
int digitPos1(int num, int digit)
{
   int pos=0;
   do {
      pos++;
      if (num % 10 == digit)
         return pos;
      num = num / 10;
   } while (num > 0);
   return 0;
}
```

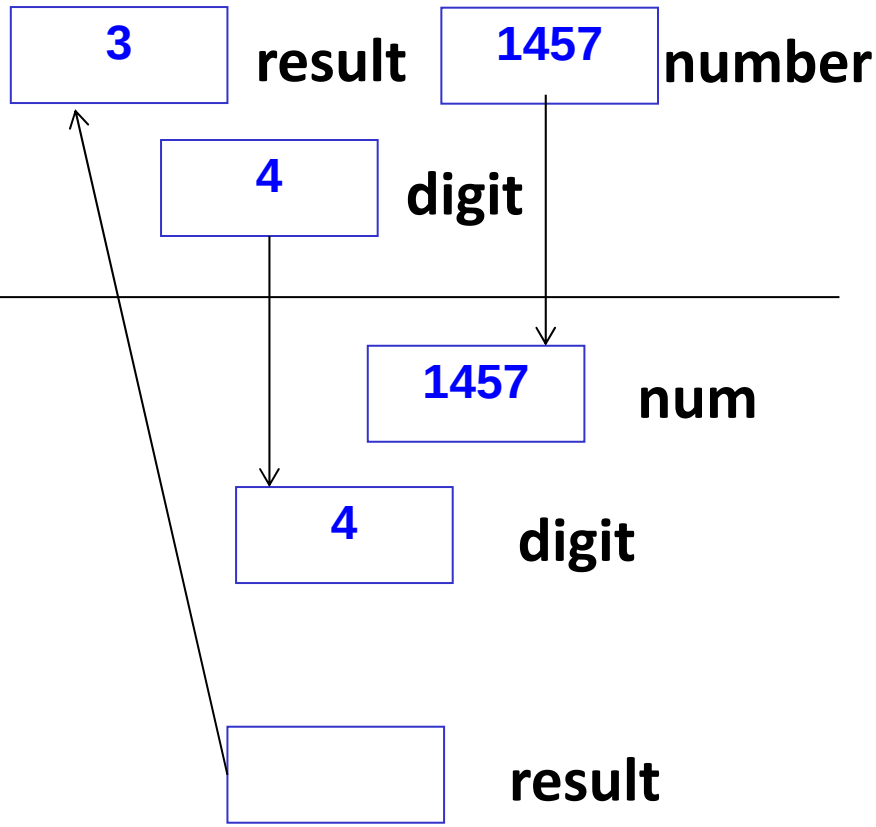


In the loop:

- pos=1 , 1457%10 -> 7 (not equal to digit 4)
  num=1457/10 = 145
- pos=2, 145%10 -> 5 (not equal to 4)
  num=145/10 = 14
- pos=3, 14%10 -> 4 (equal to 4)
  return 3, then exit function

Q2 – Call by Reference

```
#include <stdio.h>
void digitPos2(int num,int digit,int *result);
int main()
{
   int number, digit, result=0;
   printf("Enter the number: \n");
   scanf("%d", &number);
   printf("Enter the digit: \n");
   scanf("%d", &digit);
   digitPos2(number, digit, &result);
   printf("digitPos2(): %d\n", result);
   return 0;
}
void digitPos2(int num, int digit,
int *result)
{
   int pos=0;
   *result=0;
   do {
     pos++;
      if (num % 10 == digit){
         *result = pos;
         break;
      }
      num = num / 10;
   } while (num > 0);
}
```



Functions and Pointers – Q3

(**square**) Write a function **square1()** that returns the square of a positive integer number *num*, by computing the sum of odd integers starting with 1 as shown in the example below. The result is returned to the calling function. For example, if *num* = 4, then $4^2 = 1 + 3 + 5 + 7 = 16$ is returned; if *num* = 5, then $5^2 = 1 + 3 + 5 + 7 + 9 = 25$ is returned. The function prototype is:

```
int square1(int num);
```
// call by value

Write another function **square2()** that passes the result through the third parameter, *result*. The function prototype is:

```
void square2(int num, int *result);
```
// call by reference

Sample input and output sessions:

(1)
```
Enter the number:
4
square1(): 16
square2(): 16
```

(2)
```
Enter the number:
5
square1(): 25
square2(): 25
```

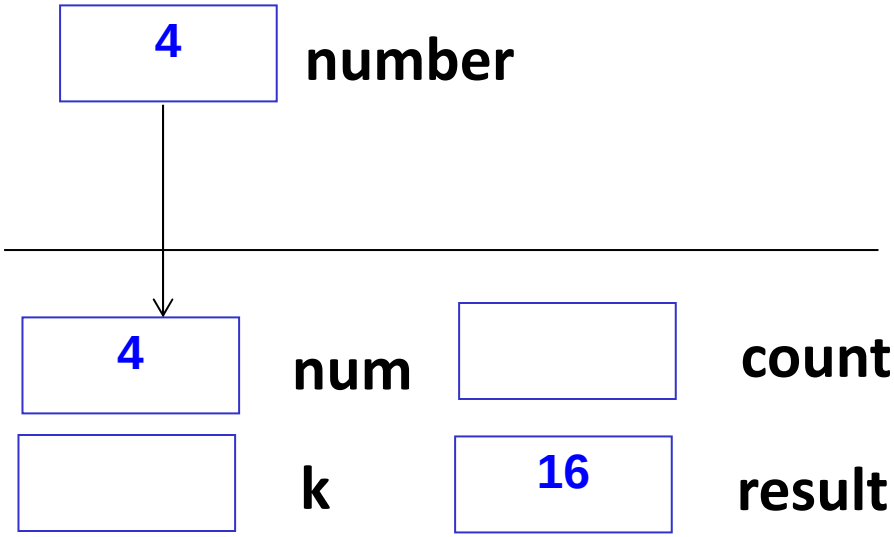
Q3 – Call by Value

```
#include <stdio.h>
int square1(int num);
int main()
{
   int number;

   printf("Enter the number: \n");
   scanf("%d", &number);
   printf("square1(): %d\n",
           square1(number));
   return 0;
}
```

```
int square1(int num)
{
    int count=0, k=1, result=0;

    while (count < num)
    {
        result += k;
        k += 2;
        count++;
    }
    return result;
}
```



In the loop:

- result=0+1=1 , k=3, count=1
- result=1+3=4, k=5, count=2
- result=4+5=9, k=7, count=3
- result=9+7=16, k=9, count=4
- Exit loop

Q3 – Call by Reference

```c
#include <stdio.h>
void square2(int num, int *result);
int main()
{
   int number, result=0;

   printf("Enter the number: \n");
   scanf("%d", &number);
   square2(number, &result);
   printf("square2(): %d\n", result);
   return 0;
}
```

```c
void square2(int num, int *result)
{
   int count=0, k=1;

   *result=0;
   while (count < num)
   {
      *result += k;
      k += 2;
      count++;
   }
}
```