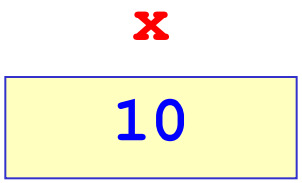
Tutorial 2 (Week 3)
Functions and Pointers

Variables in C

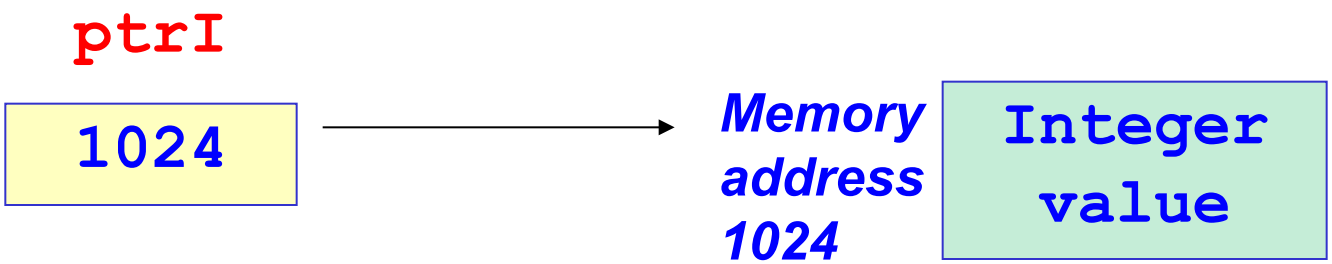
Primitive Variables – Variables which store data declared under basic data types, e.g. integer (int), floating point (float, double), character (char).

int x = 10;



Pointer Variables – Variables which store the **address** of the memory locations of a data object.

int *ptrI;



Primitive Variables

```
int num;
```

(1) num

– it is a variable of data type int

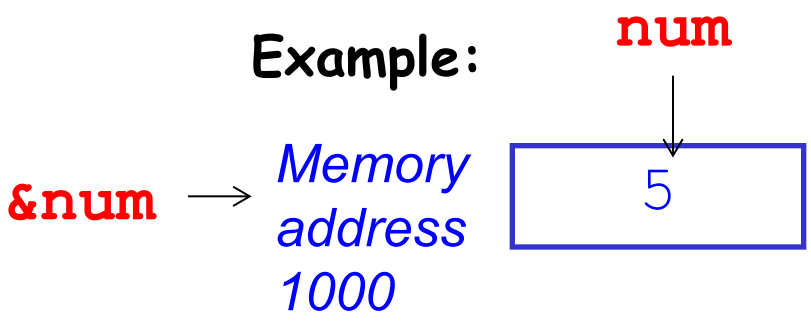
– its memory location (4 byes) stores the int value of the variable

(2) &num

– it refers to the memory address of the variable

– the memory location is used to store the int value of the variable

Example:



Note: You may also print the address of the variable using the printf() statement.

Pointer Variables

```
int * ptrI;
```

You need to understand the following 2 concepts:

(1) ptrI

– pointer variable

– the value of the variable (i.e. stored in the variable) is an **address**

(2) *ptrI

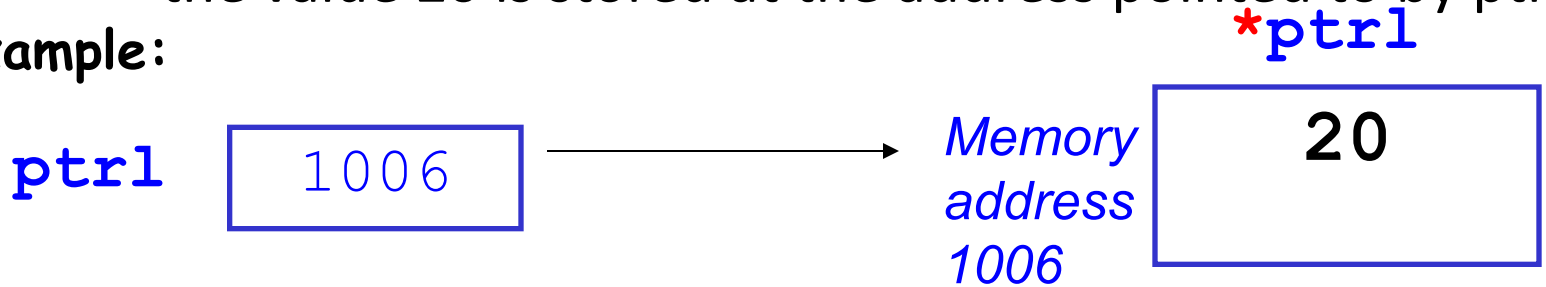
– contains the **content (or value)** of the memory location pointed to by the pointer variable ptrI

– referred to by using the **indirection operator (*), i.e.** *ptrI, *ptrF, *ptrC.

– For example: we can assign `*ptrI = 20;`

=> the value 20 is stored at the address pointed to by ptrI.

Example:



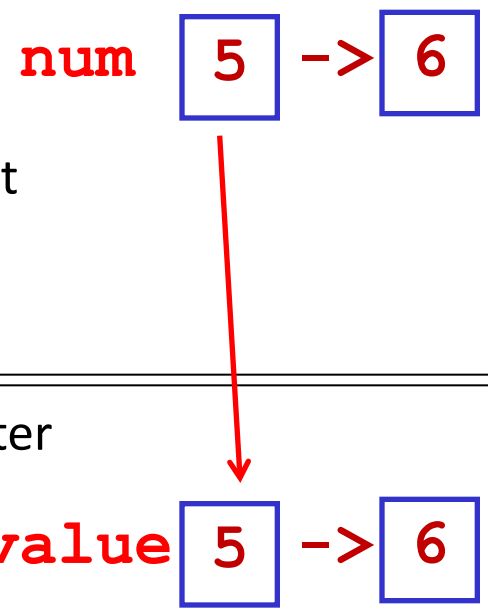
Function Communications: (1) Call by Value

- **Call by Value -** Communications between a function and the calling body is done through **arguments** and the **return value** of a function.

```
#include <stdio.h>
int add1(int);

int main( )
{
    int num = 5;
    num = add1(num);        // num – called argument
    printf("The value of num is: %d", num);
    return 0;
}
int add1(int value)         // value – called parameter
{
    value++;
    return value;
}
```

Output
The value of num is: 6

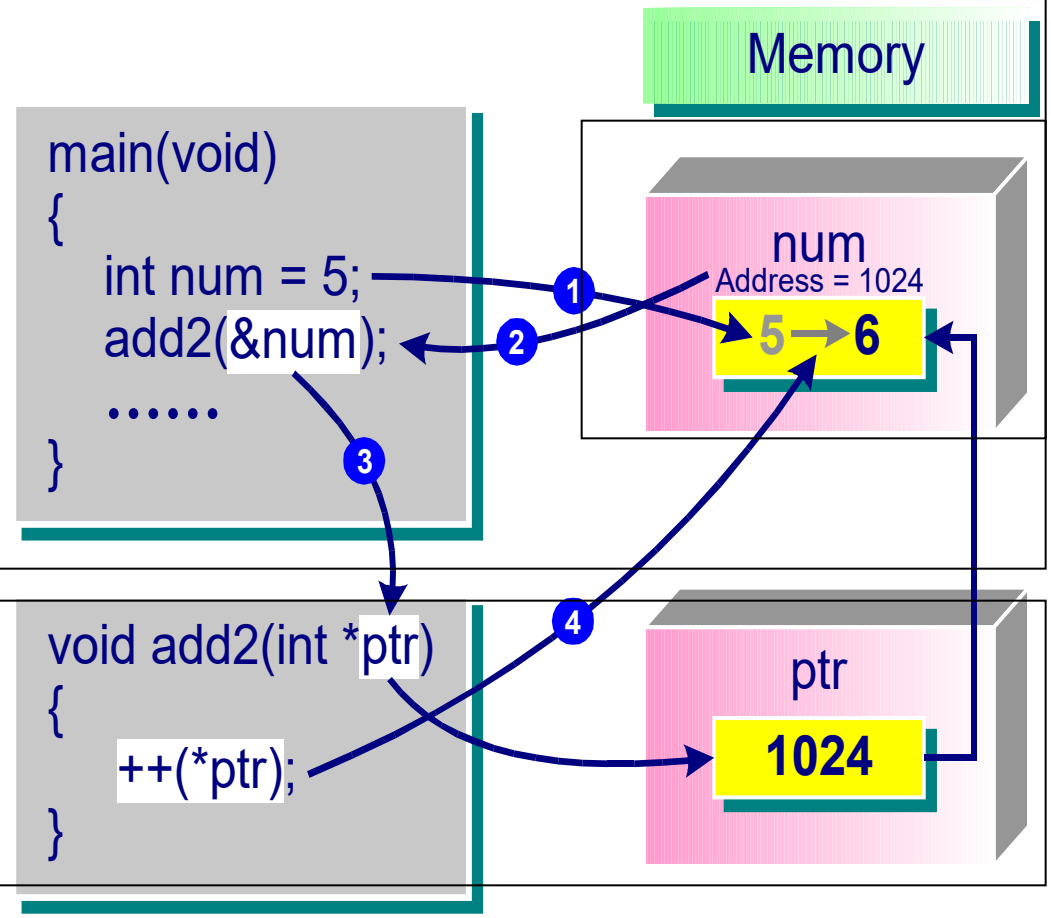


Function Communications: (2) Call by Reference

```
#include <stdio.h>
void add2(int *ptr);
 int main()
{
    int num = 5;
    /*passing the address of num*/
    add2(&num);
    printf("Value of num is: %d",
            num);
    return 0;
}
```

```
void add2(int *ptr)
{
    ++(*ptr);
}
```

pointer



Output
Value of num is 6

- **Any change to the value pointed to by the parameter ptr will change the argument value num (instantly).**

Function Communications: (2) Call by Reference

1. In the **function definition**, the parameter must be prefixed by **indirection operator ***:

add2( ) function header: **void add2(int *ptr) { …}**

2. In the **calling function**. the arguments must be pointers (or using **address** operator as the prefix):

main/other calling function: **add2(&num) ;**

Q1 (Pointers)

Assume the following declaration:

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

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

For each case below, determine the values of

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

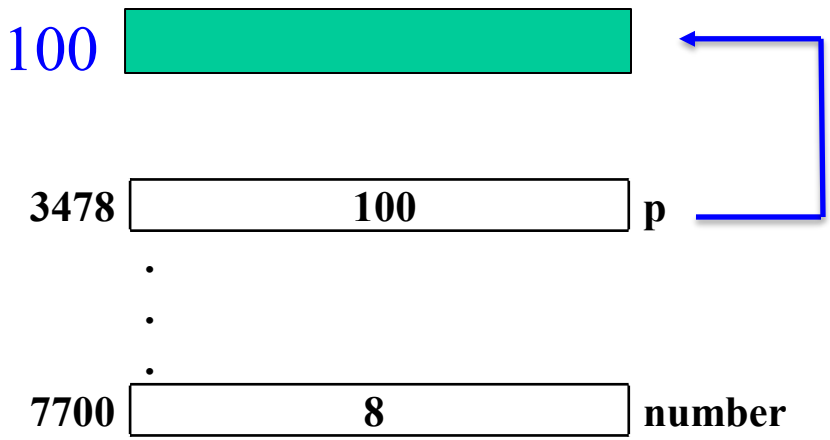
All of the results are cumulative.

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

(i) p = 100; number = 8

p = 100

number = 8



That is,

(a) number is 8

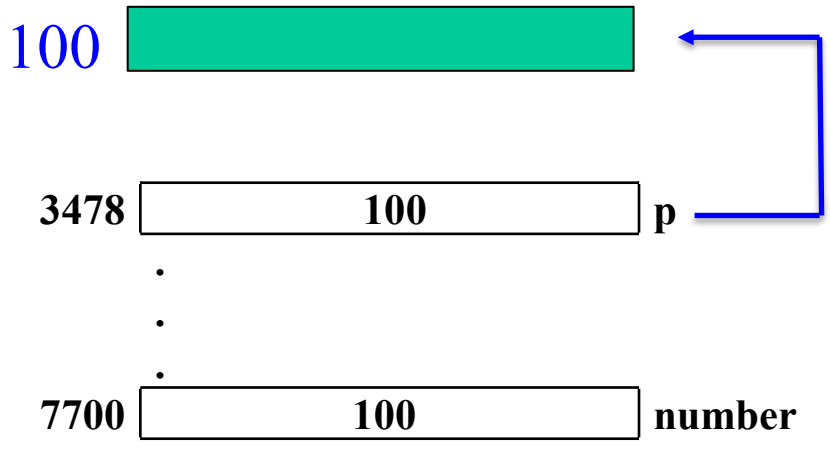
(b) &number is 7700

(c) p is 100

(d) &p is 3478

(e) *p is the content of the memory location 100.

(ii) number = p



number = p

That is,

(a) number is 100

(b) &number is 7700

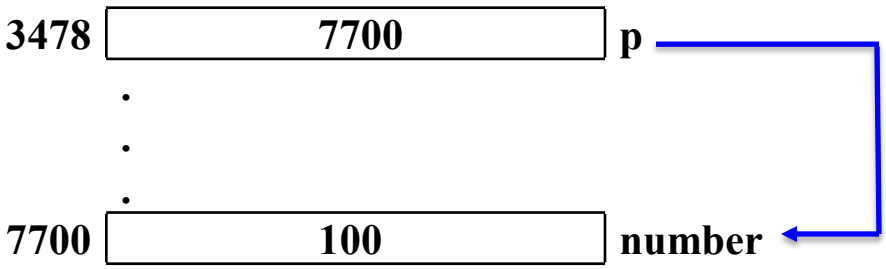
(c) p is 100

(d) &p is 3478

(e) *p is the content of the memory location 100

(iii) p = &number

p = &number



That is,

(a) number is 100

(b) &number is 7700

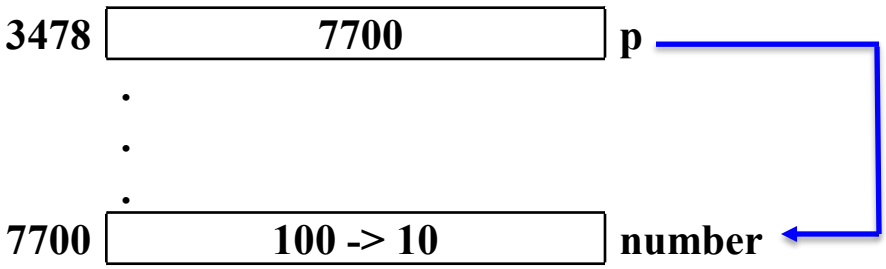
(c) p is 7700

(d) &p is 3478

(e) *p is 100

(iv) *p = 10

*p = 10



That is,

(a) number is 10

(b) &number is 7700

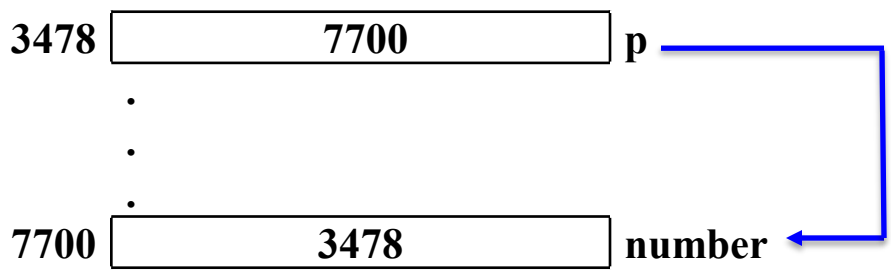
(c) p is 7700

(d) &p is 3478

(e) *p is 10

(v) number = &p

number = &p



That is,

(a) number is 3478

(b) &number is 7700

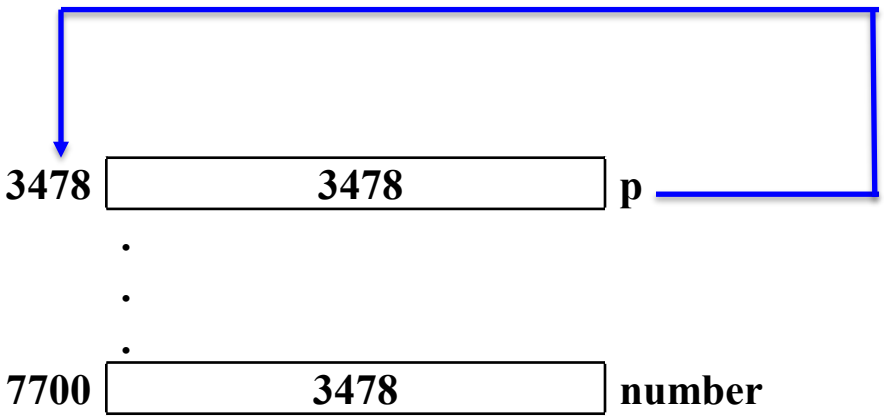
(c) p is 7700

(d) &p is 3478

(e) *p is 3478

(vi) p = &p

p = &p



That is,

(a) number is 3478

(b) &number is 7700

(c) p is 3478

(d) &p is 3478

(e) *p is 3478

Q2 (digitValue)

Write a function that returns the value of the kth digit (k>0) from the **right** of a non-negative integer *n*. For example, if n=1234567 and k=3, the function will return 5 and if n=1234 and k=8, the function will return 0. Write the function in two versions.

(1)The function **digitValue1()** returns the result:

```
int digitValue1(int n, int k);                    // call by value
```

(2) The function **digitValue2()** passes the result through the parameter *result*:

```
void digitValue2(int n, int k, int *result);   // call by reference
```

Some sample input and output sessions are given below:

```
Enter a number: 1284567
Enter the digit position: 3
digitValue1(): 5

Enter a number: 1234
Enter the digit position: 8
digitValue2(): 0
```

Q2 (digitValue)

```
#include <stdio.h>
int digitValue1(int num, int k);

int main(){
   int num, digit;

   printf("Enter the number: ");
   scanf("%d", &num);
   printf("Enter the digit pos:");
   scanf("%d", &digit);
   printf("digitValue1(): %d\n",
       digitValue1(num, digit));

   return 0;
}
```

1284567	num
3	digit

```
int digitValue1(int num, int k)
{
   int i, r;
   for (i=0; i<k; i++)
   {
      r = num%10;
      num /= 10;
   }
   return r;
}
```

num → 1284567

k → 3

In the loop:

- i=0, r=1284567%10 = 7
  num=1284567/10=128456
- i=1, r=128456%10 = 6
  num=128456/10=12845
- i=2, r=12845%10 = 5
  num=12845/10=1284
- Exit loop

Return 5

Note:

- **When dealing with number, use % operator to get the remainder of a number, and / operator to get the quotient of the number.**

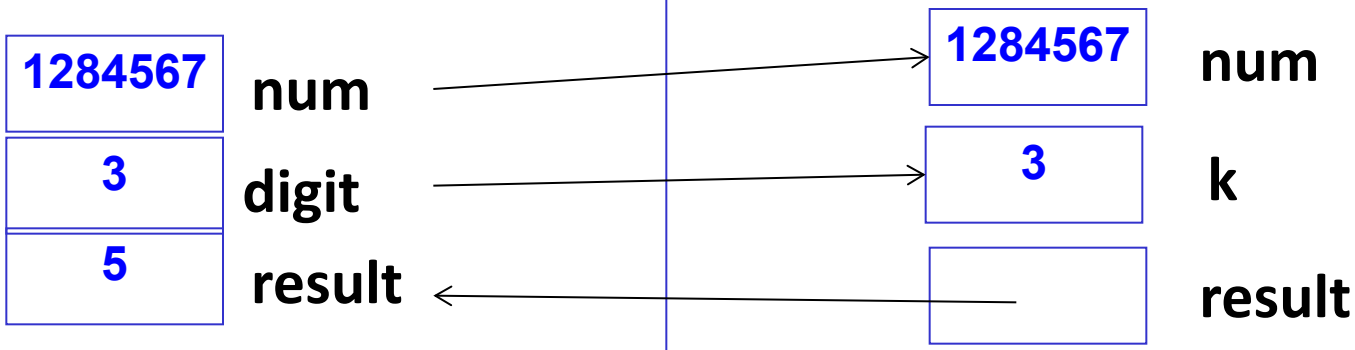
Q2 (digitValue)

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

   printf("Enter the number: ");
   scanf("%d", &num);
   printf("Enter the digit pos:");
   scanf("%d", &digit);
   digitValue2(num, digit, &result);
   printf("digitValue2():%d\n", result);
   return 0;
}
```

```
void digitValue2(int num,
int k, int *result)
{
   int i, r;

   for (i=0; i<k; i++)
   {
      r = num%10;
      num /= 10;
   }
   *result = r;
}
```



Q3 (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 be 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);
```

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
```

Q3 (extOddDigits)

```
#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;
}
```

Q3 (extOddDigits)

```
int extOddDigits1(int num)
{
    int power = 1;
    int total = 0;
    int digit;

    while (num > 0) {
        digit = num % 10;
        num /= 10;
        if ((digit % 2) == 1) {
            total += digit * power;
            power *= 10;
        }
    }
    return (power==1) ? -1 : total;
}
```

Example: num=123

In the loop:

- num=123, digit=3,num=12
  digit%2==1
  TRUE =>total=0+3*1=3;
  power=1*10
- num=12, digit=2,num=1
  digit%2==0
  FALSE
- num=1, digit=1,num=0
  digit%2==1
  TRUE =>total=3+1*10=13;
  power=10*10=100
- Exit loop

power==100 => return 13

Q3 (extOddDigits)

```
void extOddDigits2(int num, int *result)
{
    int power  = 1;
    int total = 0;
    int digit;

    while (num > 0) {
        digit = num % 10;
        num /= 10;
        if ((digit % 2) == 1) {
            total += digit * power;
            power *= 10;
        }
    }
    if (power == 1)
        *result = -1;
    else
        *result = total;
}
```

Q4 (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 back.

A sample input and output session is given below:

```
Input x1 y1 x2 y2: 1 1 5 5
calDistance1()
Distance: 5.656854
calDistance2()
Distance: 5.656854
```

Q4 (calDistance)

```
#include <stdio.h>
#include <math.h>

void  inputXY(double *, double *, double *, double *);
double   calDistance1(double, double, double, double);
void   calDistance2(double, double, double, double, double*);
void  outputResult(double);

int main()
{
    double   x1, y1, x2, y2, distance;

    inputXY(&x1, &y1, &x2, &y2);   // call by reference

    distance = calDistance1(x1, y1, x2, y2);    // call by value
    printf("calDistance1()\n");
    outputResult(distance);     // call by value

    calDistance2(x1, y1, x2, y2, &distance);    // call by reference
    printf("calDistance2()\n");
    outputResult(distance);     // call by value

    return 0;
 }
```

```
void inputXY(double *x1, double *y1, double *x2, double *y2)
{
    printf("Input x1 y1 x2 y2: ");
    scanf("%lf %lf %lf %lf", x1, y1, x2, y2);
}
```

Using Call by Reference

/* with call by reference, the function inputXY() will be able to pass the values of 4 variables to the calling function */

User Input:

```
Input x1, y1, x2, y2: 5 10 15 20
```

Note: more than 1 input to be returned

inputXY – you may return more than one value to the calling function via the pointer variables

```
double calDistance1(double x1, double y1, double x2, double y2)
{
    x1 = x1 - x2;      x1 = x1 * x1;
    y1 = y1 - y2;      y1 = y1 * y1;
    return (sqrt(x1 + y1));
 }
```

Using Call by Value

```
void calDistance2(double x1, double y1, double x2, double y2, double *dist)
{
    x1 = x1 - x2;      x1 = x1 * x1;
    y1 = y1 - y2;      y1 = y1 * y1;
    *dist = sqrt(x1 + y1);
 }
```

Using Call by Reference

```
void outputResult(double dist2)
{
    printf("Distance: %f\n", dist2);
 }
```