Example 1 : Pass by Value

- A function that should swap two values

```
int main(void)
{
   int num1=3,
   num2=9;
   swap(num1,num2);
}
```

```
void swap( int first,int second)
{
   int temp;
   temp = first;
   first = second;
   second = temp;
    return;
}
```

Actually, it does not! As copies of num1 and num2 are passed to the function, all changes of first and second have no effect on num1 and num2.

...Correction- Pass by Reference

- Pass the addresses of the variables from the main()

```
int main(void)
{
    int num1=3, num2=9;

    swap(&num1, &num2);
}
```

```
*first = &num1
*second = &num2
```

```
void swap( int *first,int *second)
{
    int temp;

    temp = *first;

    *first = *second;

    *second = temp;

    return;
}
```

As *first* and *second* point to *num1* and *num2*, any changes in *first* and *second* affect *num1* and *num2*

Example 2 :Program to Sort three Numbers

Write a program that gets three data values, num1, num2, and num3, and rearrange the data so that they are in increasing sequence.

```
Enter three numbers separated by blanks> 7.5 9.6 5.5
The numbers in ascending order are: 5.50 7.50 9.60
```

...Continued

```
1.  /*
2.   * Tests function order by ordering three numbers
3.   */
4.  #include <stdio.h>
5.
6.  void order(double *smp, double *lgp);
7.
8.  int
9.  main(void)
10. {
11.         double num1, num2, num3; /* three numbers to put in order        */
12.
13.         /* Gets test data                                                 */
14.         printf("Enter three numbers separated by blanks> ");
15.         scanf("%lf%lf%lf", &num1, &num2, &num3);
16.
17.         /* Orders the three numbers                                       */
18.         order(&num1, &num2);
19.         order(&num1, &num3);
20.         order(&num2, &num3);
21.
22.         /* Displays results                                               */
23.         printf("The numbers in ascending order are: %.2f %.2f %.2f\n",
24.                 num1, num2, num3);
25.
26.         return (0);
27. }
```

...Continued

```
28.
29. /*
30.  * Arranges arguments in ascending order.
31.  * Pre:   smp and lgp are addresses of defined type double variables
32.  * Post:  variable pointed to by smp contains the smaller of the type
33.  *        double values; variable pointed to by lgp contains the larger
34.  */
35. void
36. order(double *smp, double *lgp)     /* input/output */
37. {
38.         double temp; /* temporary variable to hold one number during swap    */
39.         /* Compares values pointed to by smp and lgp and switches if necessary    */
40.         if (*smp > *lgp) {
41.                 temp = *smp;
42.
43.                 *smp = *lgp;
44.                 *lgp = temp;
45.         }
46. }
```

Example 3 : Printing a string

```
2  // Printing a string one character at a time using
3  // a non-constant pointer to constant data.
4
5  #include <stdio.h>
6
7  void printCharacters(const char *sPtr);
8
9  int main(void)
10 {
11    // initialize char array
12    char string[] = "print characters of a string";
13
14    puts("The string is:");
15    printCharacters(string);
16    puts("");
17 }
19 // sPtr cannot be used to modify the character to which it points,
20 // i.e., sPtr is a "read-only" pointer
21 void printCharacters(const char *sPtr)
22 {
23    // loop through entire string
24    for (; *sPtr != '\0'; ++sPtr) { // no initialization
25       printf("%c", *sPtr);
26    }
27 }
```

```
The string is:
print characters of a string
```

Example 4 : Converting a string case

```
2  // Converting a string to uppercase using a
3  // non-constant pointer to non-constant data.
4  #include <stdio.h>
5  #include <ctype.h>
6
7  void convertToUppercase(char *sPtr); // prototype
8
9  int main(void)
10 {
11    char string[] = "cHaRaCters and $32.98"; // initialize char array
12
13    printf("The string before conversion is: %s", string);
14    convertToUppercase(string);
15    printf("\nThe string after conversion is: %s\n", string);
16 }
18 // convert string to uppercase letters
19 void convertToUppercase(char *sPtr)
20 {
21    while (*sPtr != '\0') { // current character is not '\0'
22       *sPtr = toupper(*sPtr); // convert to uppercase
23       ++sPtr; // make sPtr point to the next character
24    }
25 }
```

```
The string before conversion is: cHaRaCters and $32.98
The string after conversion is: CHARACTERS AND $32.98
```

Example 5 : **C sizeof with pointers**

```
float *ptrF;    /* can only point at float variables */
char  *ptr1;    /* can only point at char variables */
int   *ptr2;    /* can only point at int variables */

printf("char*  pointer requires %d bytes\n", sizeof(ptr1)   );
printf("char*  points at %d byte data", sizeof(*ptr1) );
```

Output:

```
char* pointer requires 4 bytes
char* points at 1 byte data
```

- As all pointers have a type associated with them, you cannot assign a pointer of one type to a pointer of another type.

Example 5: Using Array Elements as Function Arguments

The function prototype below shows one type double input parameter (`arg_1`) and two type double * output parameters (`arg2_p and arg3_p`).

```
void do_it (double arg_1, double *arg2_p, double *arg3_p);
```

If `x` is declared as an array of type double elements in the calling module, the statement:

```
do_it(x[0], &x[1], &x[2]);
```

uses the first three elements of array `x` as actual arguments. Array element `x[0]` is an input argument and `x[1]` and `x[2]` are output arguments.