Lecture 11

Interchanging Data Between Two Functions in C language. Passing and Receiving Data. Passing arrays

When a function calls another they communicate to each other by interchanging data between them. Interchanging data between two functions is provided by two ways: **a) passing (sending) data** from calling function to called function and **b) receiving data** by calling function from called function.

Passing data into function can be performed in 2 ways:
1) by global variables of the program (is not recommended);
2) by arguments of the function (is recommended).

Using global variables for passing data is not recommended, because they are declared out of texts of functions and it is difficult to manage them. Also, if your function has local variables with the same names as global variables than in the body of the function only local variables are valid and in this case global variables cannot be used for data interchanging.

Passing (sending) data by using arguments of the function is performed **by copying values of arguments of a function call into locations of parameter variables, which are local variables** of the function **(passing data by values).** What is way, when the function changes the values of parameter variables these changes do not reflect to the argument variables of the function call. If you want to change values of variables of the function it is necessary to pass (send) to it addresses of these variables which can be used by called function for changing these variables by means of dereferencing operator. In this case corresponding parameter variables of called function have to be pointers **(passing data by address or by reference - but steel it is passing addresses by values).**

Passing arrays into function. In C/C++ languages arrays cannot be passed into function. We can pass only starting address of an array by using pointer parameter of the function

Receiving data from function can be performed in 3 ways:
1) by global variables of the program (is not recommended for the same reasons as for passing data);
2) by pointer arguments (is recommended) and
3) by returning value (is recommended).
Note, that by using returning value we can receive from function only one value of corresponding type. If we want to determine more than a value we'll have to use pointer parameters (arguments).

To understand better using of pointer parameters and arguments for communication between two functions let's consider two versions of the function **swap( )** and corresponding program for swapping two integer values. The first version of function and program does not swap the values, because in this function we use regular variable parameters and this version cannot be used for swapping two integer values whereas the second version of function and program does swap the values, because in this function we use pointer variable parameters.

The **first version** of the function **swap( )** and corresponding program for swapping two integer values:

```
#include<stdio.h>
void  swap(int a , int b);
int  main( )
{ int  a, b ;
  clrscr( );
  printf("\n Enter two integer values a and b:  ");
  scanf("%d%d", &a, &b);
  printf ( "\n Initial value a = %d, Initial value b = %d\n", a, b);
  swap(a, b); // function call
  puts("\n Results of swapping for the first version of function swap( ) : ");
  printf ("\n Final value  a = %d, Final value  b = %d\n", a, b);
  return 0;
}
void  swap( int a, int b)
  { int t;
    t = a;
    a = b;
    b = t;
    return;
 }
```

The **second version** of the function **swap( )** and corresponding program for swapping two integer values:

```
#include<stdio.h>
void  swap(int *a, int *b);
int  main( )
{ int  a,b;
  clrscr( );
  printf("\n Enter two integer values a and b:  ");
  scanf("%d%d", &a, &b);
  printf ( "\n Initial value a = %d, Initial value b = %d\n", a, b);
  swap(&a,&b);
  puts("\n Results of swapping for the second version of function swap( ) : ");
  printf ("\n Final value  a = %d, Final value  b = %d\n", a, b);
  return 0;
}
 void  swap( int *a, int *b)
  {  int t;
     t =*a;
   *a =*b;
   *b = t;
     return;
   }
```