**Labsheet 7**

1. We have two arrays A and B, each with 10 integers. Write a function that takes the two arrays and their size as parameters and tests if every element of array A is equal to its corresponding element in array B. The function is to return 1 if all elements are equal, and 0 if at least one element is not equal. Test using a main( ) function.
2. Your friend sends an encrypted secret message to you. You know that the message is a single word and to read the original message you need to add a key to each character. Both of you know that the key is the ASCII difference of the first and last character of the message sent to you. Write a function to decrypt the message. Test using a main( ) function.
3. Let marks[5][4] be a two dimensional array of integers. Write the following functions.

1. *Read\_array( )* to read values into the array
2. *Display\_array( )* to print the array
3. *Inc\_even( )* to increment the values in the EVEN columns by 1 for each ODD row in the array
4. *main( )* function to test your program.
5. Write a C Program to rotate all the elements in the array cyclically to the left by one position. If the original array is {41, 12, 23, 34, 15} after rotation the array will look like {12, 23, 34, 15, 41}. Write functions to read, display and rotate the array to the left.
6. Write a C program to read a sentence from the user and count the number of words in the sentence. The prototype of the count function is as follows:

***int count( char s[ ]);***

1. Write a C Program to reverse an array. If the original array is {41, 12, 23, 34,15} after reversal the array will look like { 15, 34, 23, 12, 41}. The prototype of reverse function is given as follows: ***void reverse(int a[ ], int n);***
2. Write a program to find the frequency of a particular element in an array. Read the element from the user in the main function. For example if the array contains the following elements : {1, 4, 3, 4, 7, 4, 3}, and the user entered 4, then the output would be:  
   ***4 is repeated 3 times.***

**Pointers**

1. Run the following program and understand the concept of pointers:

*#include<stdio.h>*

*main()*

*{*

*int a=10;*

*int \* b;*

*b= &a;*

*printf(" \n a=%d\t b=%d\t \*b=%d\n",a,&a,\*b);*

*}*

1. Write a program which contains one regular integer variable **x** and one pointer to integer **ip.** Try out the following basic uses of pointers:

* Assign **ip** the address of **x**.
* Using the pointer as the argument to **scanf()**, read a value into the location **ip** points at ( **i.e. x**) and then print out the value of **x.**
* Assign a new value to **\*ip**, and print out **x**.
* Assign a value to **x** and print out **\*ip**.

1. Write and test the following function:

**void CharSwap( char \*cp1, char \*cp2 );**

The function must swap the characters in the memory locations stored by its two pointer arguments.

1. Write the following function:

**void input( int \*small, int \*medium, int \*large);**

The function must read in 3 integers from the user, assign the smallest of the three to **\*small**, the middle one to \***medium**, and the largest to \***large.** Write a main program which tests the function.

1. Write a function which finds both the minimum and maximum elements in an array of **doubles.** The function will need four arguments: the array, the length of the array, and two pointers to **double** through which the function can return the minimum and maximum values. Here is the prototype for the function:

**void MinMax( double \*data, int length, double \*retmin, double \*retmax);**

Write a main( ) to test the function.

1. Declare an array of 5 integers. Print out the address of each element of the array. To print the address you can use **%p printf()** format which usually prints pointers in hexadecimal. Use the **%u (unsigned) printf()** format if you would rather not deal with hexadecimal notation. Notice how the difference between each successive address is **sizeof(int)**.
2. Create an array of characters and fill it with a string. Define a character pointer and assign it the base address of the array. Use pointer arithmetic to traverse the array and print out the characters one at a time.
3. Write a function which takes a **char\*** as its only argument and prints out each character in the array one at a time. Use pointer arithmetic to step through the array. Recall that the pointer is passed by value so it can be modified without affecting the actual parameter.
4. Declare an array of **ints** and a pointer to **int.** Assign the pointer the base address of the array.Print out the address of the array using **printf()**, increment the pointer using **++** and print out the address again. The value will have increased by **sizeof(int).**

Try the same excersise with other types of arrays such as **char, float, double,** and **long double.** Notice how the value of the pointer is automatically increased by the appropriate amount.

1. It is the programmer’s responsibility to keep a pointer into an array inside the array bounds. Compilers do not catch the error if the pointer is moved outside the array. Using an invalid pointer may write into memory being used for other variables. To demonstrate this, try the following. Define the variables:

**Char s1[5] = “AAA”, s2[5], \*s;**

Make **s** point just beyond the end of **s2** and write into the location **s** points at. Print out **s1.**

1. Write your own version of the library function **strcat(). strcat()** takes two arguments, **dest**  and **source**. It copies **source**  onto **dest.** Use pointer arithmetic to manipulate the arrays. Declare the function like this:

**void StringCopy( char \*dest, const char \*source);**

1. Write your own version of the library function **strcat().** Recall that **strcat()** takes two arguments, **dest and src**, and appends **src** onto the end of **dest**. **strcat()** must first find the end of the **dest**  string, and then copy **src** onto the end of **dest.**

Declare the function like this:

**void StringCat( char \*dest, char \*src);**

1. Write a program to read and print an array of n numbers, then find out the smallest number. Also print the position of the smallest number.

Use the following functions in the program.

**void read\_array(int \*array, int n);**

**void print\_array(int \*array , int n);**

**void find\_small(int \*array, int n, int \*small, int \*pos);**

1. Write a program to read and display a matrix using pointers.
2. Write a program to read and display a 3D array using pointers.
3. Write a program to read and display values of an integer array. Allocate space dynamically for the array.

**Command Line Arguments**

1. Write a program which prints out the argument count and loops through its command line arguments printing each one. Run the program with different number of arguments.
2. Write a program which uses **strcmp()**  to determine if any two of its command line arguments are the same. You will need to use nested loops to do this. Do not include the program name in your comparison (i.e. skip **argv[0]**).
3. Write a program which prints out all its command line arguments ( including the program name) ***one character at a time***. Use either **printf()** or **putchar()**  to print the characters.

**Program Exit Status**

1. Write a simple program in which **main()** returns a value using a **return**  statement. You can run the command

**echo $status**

to verify that the value you returned from your program was sent to the operating system.

Note: **status** will hold the exit status of the last command, so if you run anything else after your program but before the **echo**  command your program return value will be gone.

1. Instead of using **return** to set the exit status, have **main()** call the exit function. Be sure to include **stdlib.h** to get the prototype for the **exit()**  function.
2. Move the call to **exit()** to a function other than **main().**  Print out a message just before and after the call to **exit()** and verify that the second message is never printed.

\*\*\*\*\*\*\*\*\*\*\*\*\*