**Labsheet 9**

**Pointer Arithmetic**

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 programmers 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);**

**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.