# EEE2007: Computer Systems and Microprocessors
*Lab 2: Introduction to Pointers and Call by References*
*Module Instructor: Dr Rishad Shafik*

## Exercise I: Introduction to C++ Pointers, and Call by Reference

**Recommended Time:** 25 Mins Maximum

**Aims:**
a. To understand the difference between C++ pointer variables and non-pointer variables
b. To understand the difference between C++ call by value and call by reference

Follow the instructions below and try to do accordingly-

1.  **DOWNLOAD** the source code of pointer_example.cpp.

2.  **REVIEW** the source code of pointer_example.cpp using Notepad++ (Start->type "Notepad++")

    Go through each line to understand how the code is organized. Check the following:
    - Two non-pointer integer variables *B_local* and *C_local* are used together with two pointer integer variables *B_pointer* and *C_pointer*
    - The *B_pointer* and *C_pointer* are used to point to the addresses of B_local and C_local
    - The example uses to functions to add the *B_local* and *C_local* values: *add_values_cbv*() to add values through call by value, *add_values_cbr*() to add values through call by reference.
    - The *add_values_cbv*() function copies the values of *B_local* and *C_Local* to the parameters *param_B* and *param_C*, while the *add_values_cbr*() function copies the memory references of *B_local* and *C_Local* through the pointer-type parameters *param_B* and *param_C*.

3.  **COMPILE** the source code of pointer_example.cpp:
    a.  Start Cygwin command shell through Start->All Programs->Cygwin->Cygwin Bash Shell
    b.  In the Cygwin shell type: *g++ –Wall pointer_example.cpp –o pointer_example*

        The –Wall option enables all the warnings, and the –o option enables specification of the output executable

        Your compilation should generate an executable called *pointer_example*

4.  **EXECUTE** the *pointer_example* executable by typing the following in the Cygwin shell
    *./pointer_example*

5.  **OBSERVE** the outputs:
    a.  see how *B_pointer* and *C_pointer* can copy the memory references of *B_local* and *C_local*
    b.  see how values (not reference addresses) are copied by non-pointer variables within the *add_values_cbv*() function

c. see how memory reference addresses are copied by pointer variables within the *add_values_cbr*() function

6. Now **UNCOMMENT** the lines 80-81 and 95-96 to enable local value modification within the two functions.
7. **RECOMPILE, EXECUTE** and **OBSERVE** the following:
    d. see how *B_local* and *C_local* values are <u>unaffected</u> by even after modification within the *add_values_cbv*() function
    e. see how *B_local* and *C_local* values are <u>affected</u> by the modification within the *add_values_cbr*() function

**QUESTION**: Why did the *B_local* and *C_local* values change after the *add_values_cbr*() function? Why did the values not change after the *add_values_cbv*() function?


## Exercise II: Call by Reference Using Arrays

**Recommended Time:** 25 Mins Maximum

**Aims:**
a. To understand how arrays can be used for modular operations through call by reference

Follow the instructions below and try to do accordingly-

9. **DOWNLOAD** the source code of arrayreference.cpp.

10. **REVIEW** the source code of arrayreference.cpp using Notepad++ (Start->type "Notepad++")

    Go through each comment embedded and try to understand the program. Check the following:
    • An double type array x is declared in the main function
    • When x is passed as an argument in functions, the first element address is passed as a call by reference
    • Hence, both fill_array() and change_array() functions can make amendments in the array elements, the first function fills the array with random double numbers and the second changes the values based on certain conditions (see the codes)

11. **COMPILE** the source code of arrayreference.cpp by
    a. Start Cygwin command shell through Start->All Programs->Cygwin->Cygwin Bash Shell
    b. In the Cygwin shell type: *g++ –Wall arrayreference.cpp –o arrayreference*

    Your compilation should generate an executable called *arrayreference*
12. **EXECUTE** the *arrayreference* executable by
    *./arrayreference*
13. **OBSERVE** the output and see how x elements are filled and changed using call by reference using x array as parameters in the functions

**QUESTION**: TRY to print the address of x's first element within main() and also x_param address within the local function.

## Exercise III: Dynamic Memory Allocation for Arrays

**Recommended Tim:** 45 Mins Maximum

**Aims:**
a. To be able to write a C++ program using dynamic memory allocation for arrays

14. **DOWNLOAD** the source code of dynamic_allocation.cpp.

15. **REVIEW** the source code of arrayreference.cpp using Notepad++ (Start->type "Notepad++")

     Go through each comment embedded and try to understand the program. Check the following:
     - How arrays are declared using a pointer
     - How new operator is used to allocate memory required
     - How delete operator is used to de-allocate or release the memory allocated.

16. **COMPILE** the source code of dynamic_allocation.cpp by
     c. Start Cygwin command shell through Start->All Programs->Cygwin->Cygwin Bash Shell
     d. In the Cygwin shell type: *gcc –Wall dynamic_allocation.cpp –o dynamic_allocation*


     Your compilation should generate an executable called *dynamic_allocation*
17. **EXECUTE** the *dynamic_allocation* executable by
     *./dynamic_allocation*
18. **OBSERVE** the input and output; execute a few times with high i values and low i values.


**SOLVE:** write a program that asks the user the number of random characters he/she wants to generate, and generate them accordingly, followed by their display on the output console.


## Exercise IV: Review of Array based Call by Reference

**Recommended Tim:** 45 Mins Maximum

**Aims:**
a. To be able to write a C++ program using array based parameters
b. To understand the call by reference using arrays in C

19. WRITE a program (encrypt.cpp) to encrypt a message, which is an array of characters: `char msg[25]="Very Important Message"`, declared and initialized in the main function. The message will be encrypted as follows:

     Update each character's ASCII code (i.e. `(unsigned int) msg[c]`, where *c* is the iterator) as
     Current ASCII code = ((previous ASCII code * 9) % 96) + 31;

     Implement the encryption through a function called *do_encrypt(char [])*. Before and also after this encrypt operation, show the encrypted message on the standard console output through another function, called `display_message(char [])`. Declare all the associated variables required.

**Exercise V: File operations in C++**

**Recommended Tim:** 25 Mins Maximum

20.