Academic Year: <u>2020</u>

Course Title: Object Oriented Programming Lab

Session:	2019
Semester:	2nd
Course Code:	CS-241L

CS-241L Object Oriented Programming Lab 02

Type of Lab: Open + Close Ended **Weightage: 5%**

CLO 1: Implement abstraction and encapsulation to develop reusable classes for objects of real world problems

Student understand the conceptual and practical concept of pointers and functions for implementing OOP concepts	**Cognitive/Understanding**	CLO1	Rubric A

Rubric A: Cognitive Domain

Evaluation Method: GA shall evaluate the students for Question according to following rubrics.

CLO	**0**	**1**	**2**	**3**	**4**	**5**
CLO1	Student is unable to understand concept of pointers and functions	Student understand pointers conceptually	Student implements concept of pointers	Student understand and implement the concept of functions	Problem set 1 and 2 is completed	Problem set 3 and 4 is completed

Lab 2

BS-Computer Science

Object Oriented Programming

Target: Revision (Pointers, Functions)

Pointers: A pointer is a variable that contains the address of a variable.

What's this about addresses, though? In C, as in many computer languages, variables refer to particular locations in memory. This location is known as the address of the variable. Numerically, the address is the number of the memory location in bytes. It is normally represented using a hexadecimal number (e.g. 0x00124AF4; recall that hexadecimal numbers in C are entered with a leading "0x"). In fact, you can directly get at the address in C by using the & operator e.g. &foo is the address of the variable foo. (Note: the & operator has another meaning ("bitwise AND") that the compiler determines from the context.) This address can be stored in a pointer. However, pointers have specific types (e.g. "pointer to integer"), so, in general, you can only store the address of an integer in a pointer to an integer. Pointers are declared as follows:

```
   int *p;  /* declares 'p' to be a pointer to an integer */
```

The declaration uses the * operator. Confusingly, this operator also stands for multiplication. However, it's usually easy to see which meaning is which depending on the context.

What can you do with a pointer? The most basic thing is that you can store a memory address there:

```
   int *p;

   int q = 79;

 p = &q;  /* now 'p' stores the address of the integer variable 'q'. */
```

Once you have the address of a variable stored in a pointer, you can retrieve the contents of the variable by dereferencing the pointer:

```
int *p;

int q = 79;

int r = 21;

int sum;

p = &q;

sum = *p + r;  /* = 100 */
```

The expression *p means to fetch the value at the address stored in the pointer p. This unary use of the * operator is called the pointer dereferencing operator. If there is any chance that it might be confused with the multiplication operator, you should surround it with parentheses e.g.

```
sum = (*p) + r;
```

These two operations, storing addresses into a pointer and dereferencing the pointer, are the two most basic pointer operations.

Problem Set:

- What will be the output of the following block of code? Show your calculations on paper and explain them.

```
#include <iostream>
using namespace std;

int main()
{
   int a = 32, *ptr = &a;
   char ch = 'A', &cho = ch;

   cho += a;
   *ptr += ch;
```

```
    cout << a << ", " << ch << endl;
    return 0;
}
```

- What will be the output of the following program? Show your calculations on paper and explain them.

```
#include <iostream>
using namespace std;
int main()
{
   int num[5];
   int* p;
   p = num;
   *p = 10;
   p++;
   *p = 20;
   p = &num[2];
   *p = 30;
   p = num + 3;
   *p = 40;
   p = num;
   *(p + 4) = 50;
   for (int i = 0; i < 5; i++)
      cout << num[i] << ", ";
   return 0;
      }
```

Functions: A function is a set of statements that take inputs, do some specific computation and produces output.

The idea is to put some commonly or repeatedly done task together and make a function, so that instead of writing the same code again and again for different inputs, we can call the function.

```
#include <stdio.h>

// An example function that takes two parameters 'x' and 'y'
// as input and returns max of two input numbers
```

```
int max(int x, int y)
{
   if (x > y)
     return x;
   else
     return y;
}

// main function that doesn't receive any parameter and
// returns integer.
int main(void)
{
   int a = 10, b = 20;

   // Calling above function to find max of 'a' and 'b'
   int m = max(a, b);

   printf("m is %d", m);
   return 0;
}
```

Parameter Passing to functions:

The parameters passed to function are called ***actual parameters***. For example, in the above program 10 and 20 are actual parameters.
The parameters received by function are called ***formal parameters***. For example, in the above program x and y are formal parameters.
There are two most popular ways to pass parameters.

Pass by Value: In this parameter passing method, values of actual parameters are copied to function’s formal parameters and the two types of parameters are stored in different memory locations. So any changes made inside functions are not reflected in actual parameters of caller.

Pass by Reference: Both actual and formal parameters refer to same locations, so any changes made inside the function are actually reflected in actual parameters of caller.

```
# include <stdio.h>
void fun(int *ptr)
{
   *ptr = 30;
}

int main()
{
  int x = 20;
  fun(&x);
  printf("x = %d", x);

  return 0;
}
```

Problem Set:

1. **Write a program in C++ to demonstrate the use of &(address of) and *(value at address) operator.**

2. **Write a C++ program to find the max of an integral data set (using pointer).** The program will ask the user to input the number of data values in the set and each value. Then the program will show the max number of the data set. In this C++ program you have to use a function that accepts the array of data values as pointer and its size. The return from the function is the pointer that points to the max value.

3. **Write a C++ function to sort an array of ten integer values in ascending order.** The function will accept two arguments-- a pointer that points to the array and the array size. The function returns a pointer that points to the sorted array.

4. **Write a C++ program that will display the calculator menu.** The program will prompt the user to choose the operation choice (from 1 to 5). Then it asks the user to input two integer vales for the calculation. Program uses a function that receives 2 inputs and choice of calculation through pointers from the menu below and performs the calculation upon input and displays result. MENU is as follows.

    1. Add
    2. Subtract
    3. Multiply
    4. Divide
    5. Modulus