Experiment No. 01
Pointers

OBJECTIVE:

Things that will be covered in today's lab:

- Pointers
- Pointer Arithmetic
- Dynamic memory allocation and deallocation

THEORY:

Pointer variable: A variable whose content is an address (i.e., a memory address) In C++, you declare a pointer variable by using the asterisk symbol (*) between the datatype and the variable name. The general syntax to declare a pointer variable is as follows:

```
datatype * identifier;
```

In C++, the *ampersand* (&), address of the operator, is a unary operator that returns the address of its operand. Similarly, "*" is a dereferencing operator, refers to the object to which its operand (pointer) points. For example, given the statements:

```
int *p, x;
p = &x;                 // assigns address of x to p
cout<< *p <<endl;       // pointer p points to x
```

The arrays studied in the prerequisite Programming Fundamentals course were **static arrays** because their size was fixed at compile time. The name of the static array is a constant pointer. One of the limitations of a static array is that every time you execute the program, the size of the array is fixed. One way to handle this limitation is to declare an array that is large enough to process a variety of data sets. However, if the array is very big and the data set is small, such a declaration would result in memory waste. On the other hand, it would be helpful if, during program execution, you could prompt the user to enter the size of the array and then create an array of the appropriate size.

Dynamic Array: An array created during the execution of a program**.** To create a dynamic array, we use ***new*** operator.

```
int size;
int *p;
p = new int [size];
```

If you are not in need of dynamically allocated memory anymore, you can use ***delete*** operator, which de-allocates memory previously allocated by new operator.

```
delete [] p;
```

Exercise 1: (10 points)

Give the output of the following C++ codes without running the code in Visual Studio.

a)

```
int x;
int y;
int *p=&x;
int *q=&y;
x=35;
y=46;
p=q;
*p=78;
cout<<x<<"  "<<y<<"  ";
cout<<*p<<"  "<<*q;
```

Output

35 78 78 78

b)

```
int x[3]={0,4,6};
int *p,t1,t2;
p=x;
(*p)++;
cout<<*p;
cout<<*(p+1);
```

Output

1 4

c)

```
int x,*p,*q;
int arr[3]={0};
p=arr;
q=p;
*p=4;
for (int j=0;j<2;j++)
{
      x=*p;
      p++;
      *p=(x+j);
}
for (int k=0;k<3;k++)
{
        cout<<*q;
        q++;
}
```

Output

1st ietreation
x = 4.
p++ mean it increment index
*p = 4 + 0 = 4.

2nd itreation
x=4.
p++ it is 1 on 1st itreation
and now it is 2 .
*p = 4 +1 = 5.

elements of array
4 4 5

Exercise 2: (10 points)

The following code snippets all have a problem. In the answer textbox, please mention what is wrong with the code and what is the fix the issue. **For part (d) only** after fixing the code please give the output as well after putting it inside a main function in a C++ file, compiling and running it in Visual Studio.

a)

```
float piVal =3.14;
float * const pi= &piVal;

float a= 2.1;
  pi = &a;

const float * pi2= &piVal;
*pi2=3.2;
```

Ans:

b)

```
int *p= new int;
int *q=new int;
*p=4;
*q=7;
p=q;
//hint can we access the 4?
cout<<"p is carrying"<<*p<<endl;
cout<<"q is carrying"<<*q<<endl;
```

Ans:

7
7

c)

```
int *p= new int;
int *q=new int;
*p=4;
*q=7;
p=q;
delete p;
//hint what sort of pointer q is
now
 *q= 10;
cout<<"q is carrying"<<*q<<endl;
```

Ans:

10

q is dangling pointer , it works some times then code crashes .

d)

```
int x,*p;
int arr[3]={0};
p=&arr;//is this correct?
p=&arr[0];
//why is arr++ not possible?
arr++;
//why is p++ possible?
p++;
for (int j=0;j<2;j++)
{
     arr[j]=j+1;
}
p--;
for (int k=0;k<3;k++)
{         cout<<*p;
        p++;}
```

Ans:

1 2 0

Exercise 3: (10 points)

Write a code in C++ that asks the user for size of an array, then allocates the required memory for a dynamic array. Using pointer arithmetic (and not the [] operator) it should then populate the array with data (if it is an integer array do add your rollnumber as well), display the data and upon completion it should deallocate the memory assigned to the array so no memory leak happens.

In the space below please give your code as well as the screenshot of the output you get after running your code