# DS
# Lab 2

Mubashra Fayaz & Aqsa Zahid

## Google Classroom

- poqor62

- Email:

  mubashra.fayyaz@nu.edu.pk

## Today’s Agenda:

- Debugging(on dev)
- OOP Programming
- Dynamic Memory
- Dynamic Arrays
- Safe Array
- Jagged Array
- Rule of three

- Exercises

# Debugging

- *Using the debugger:*
- The various features of the debugger are pretty obvious. Click the "Debug" icon to run your program and pause at the current source code cursor location; Click "Next Line" to step through the code; Click "Add Watch" to monitor variables.
- Setting breakpoints is as easy as clicking in the blank space (Line Number) next to the line in the source code.

# OOP programming

- **Using Header Files**
- Header files are used for declaration. In OOP you should use header files to declare classes and functions. It will make your program look more clean and professional.

- Header file for a class will include:
  - Include guards.
  - Class definition.
    - Member variables
    - Function declarations (only prototype)

- Implementation File will include:
    - Include directive for “header.h”
    - Necessary include directives.
    - Function definitions for all the functions of the class.

## Example

myclass.h

```
// my_class.h
#ifndef MY_CLASS_H // include guard
#define MY_CLASS_H

namespace N
{
    class my_class
    {
    public:
    void do_something();
    };

}
```

myclass.cpp

```
// my_class.cpp
#include "my_class.h" // header in
local directory
#include <iostream> // header in
standard library

using namespace N;
using namespace std;

void my_class::do_something()
{
    cout << "Doing something!" <<
endl;
}
```

my_program.cpp

```
// my_program.cpp
#include "my_class.h"

using namespace N;

int main()
{
    my_class mc;
    mc.do_something();
    return 0;
}
```

# Dynamic memory Allocation

- **Static memory allocation** happens for static and global variables. Memory for these types of variables is allocated once when your program is run and persists throughout the life of your program.
- **Automatic memory allocation** happens for function parameters and local variables. Memory for these types of variables is allocated when the relevant block is entered, and freed when the block is exited, as many times as necessary.
- **Dynamic memory allocation** is a way for running programs to request memory from the operating system when needed.

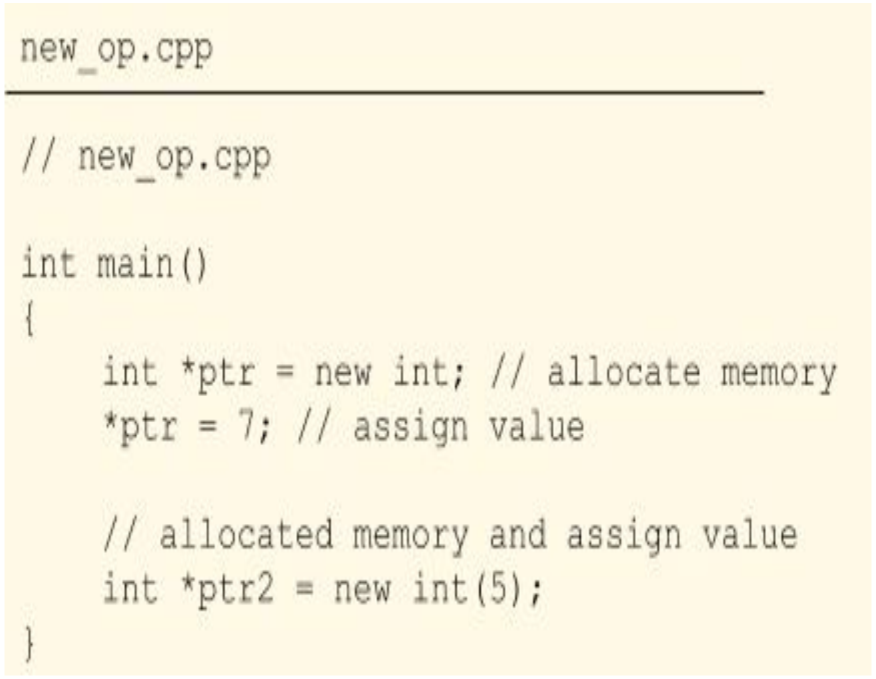
## New and Delete

**new Operator**

- This operator is used to allocate a memory of a particular type.
- This creates an object using the memory and **returns a pointer** containing the memory address.
- The return value is mostly stored in a **pointer** variable.

**delete Operator**

- When we allocate memory dynamically, we need to explicitly tell C++ to deallocate this memory.
- **delete** Operator is used to release / deallocate the memory.

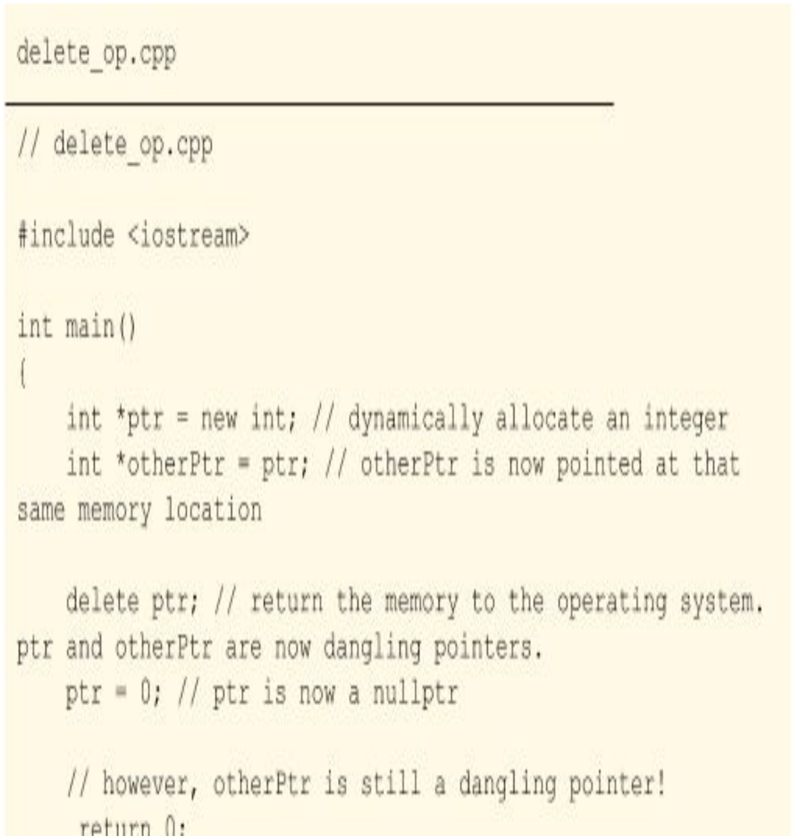


new_op.cpp

```
// new_op.cpp

int main()
{
    int *ptr = new int; // allocate memory
    *ptr = 7; // assign value

    // allocated memory and assign value
    int *ptr2 = new int(5);
}
```



delete_op.cpp

```
// delete_op.cpp

#include <iostream>

int main()
{
    int *ptr = new int; // dynamically allocate an integer
    int *otherPtr = ptr; // otherPtr is now pointed at that
same memory location

    delete ptr; // return the memory to the operating system.
ptr and otherPtr are now dangling pointers.
    ptr = 0; // ptr is now a nullptr

    // however, otherPtr is still a dangling pointer!
     return 0;
```

# Dynamic Arrays

- To allocate an array dynamically we use array form of **new** and **delete**
- (new[ ] , delete[ ])

dynamic_array.cpp

```
// dynamic_array.cpp

#include<iostream>

using namespace std;

int main()
{
    int array[] = {1,2,3};
    cout << array[0];
    cout << endl;

//    int* dArray = new int[] {1,2,3};
    int* dArray = new int[3] {1,2,3};
    cout << *dArray+1;
    cout << endl;
    cout << dArray[2];

    delete[] dArray;
}
```

## Safe Array

- This is a type of the array that ensures to handle the error of **index out of bounds.** This is done by **overloading the subscript [ ]** operator.

```
// Implementation of [] operator.  This function must
return a
// reference as array element can be put on left side
int &Array::operator[](int index)
{
    if (index >= size)
    {
        cout << "Array index out of bound, exiting";
        exit(0);
    }
    return ptr[index];
}
```

## Jagged Array

- This is also referred to as an Array of pointers or Array of arrays. This is a multidimensional with different sizes.

```
int *weekArray[7]
weekArray[0] = new int[3]; // expenditures for Sunday
weekArray[1] = new int[5]; // expenditures for Monday
etc.
weekArray[0][0] = 7; weekArray[0][1] = 17; weekArray[0][2] = 23; //
Sunday
weekArray[1][0] = 5; weekArray[1][1] = 10; etc. // Monday
delete [] weekArray[0];
delete [] weekArray[1];
etc.
```

## Rule of Three

- If you need to explicitly declare either the **destructor**, **copy constructor** or **copy assignment operator** yourself, you probably need to explicitly declare all three of them.

```
struct Node {
    char *name;
    int age;
    Node(char *n = "", int a
    = 0) {
    name = strdup(n);
    age = a; }
};
Node node1("Roger",20),
node2(node1);
strcpy(node2.name,"Wendy");
node2.age = 30;
cout<<node1.name<<'
'<<node1.age<<'
'<<node2.name<<'
'<<node2.age;
```

```
Node(char *n = 0, int a =
0) {
name = strdup(n);
age = a;
}
Node(const Node& n) { //
copy constructor;
name = strdup(n.name);
age = n.age;
}
```

```
Node& operator=(const
Node& n) {
if (this != &n) { // no
assignment to itself;
if (name != 0)
free(name);
name = strdup(n.name);
age = n.age;
}
return *this;
}
```

# Task:1

- Create a Program to solve Quadratic Equation and Calculate the roots.
  - Your program must implement a class Quadratic Equation.
  - You must use header files for class implementation.

## Task:2

- Create a program containing information for a **Student**.
  - A Student can have the following information
    - ID
    - Batch
    - Discipline
    - Expected Graduation Year
    - Current Courses (this can be an array of strings)
  - Use Dynamic Safe Arrays to store the information of multiple students.

## Task:3

- Implement a class for a **Car**. Implement Rule of Three for this class.
  - A car can have properties of another car. (same category cars).
    - Using copy constructor.
    - Using Assignment Operator.
  - A car object should be destroyed properly. Using destructor.