Object Oriented Programming Lab # 02

Course: Object Oriented Programming (CL-1004) **Semester**: Fall 2025
Instructor: **Mr. Abdullah Yaqoob**

Note:
- Maintain discipline during the lab.
- Listen and follow the instructions as they are given.
- Just raise hand if you have any problem.
- Completing all tasks of each lab is compulsory.
- Get your lab checked at the end of the session.

Pointers in C++

A Pointer is a variable whose content is a memory address. In C++, a pointer is a variable that stores the memory address of another variable. Pointers are one of the fundamental concepts in C++ that allow for dynamic memory allocation, passing variables by reference.

Single Pointer

A single pointer refers to a pointer variable that holds the address of a single variable of a specific type. It points to a single memory location. To declare a single pointer variable, you need to specify the data type, an asterisk symbol (*) and the name of the pointer variable.

```
dataType *ptrName;
```

Following is an example of declaration of a Pointer variable:

```
int *ptr;
```

Pointer variable holds the memory address of the variable which is of same data type (integer in this case). To assign the memory address of any variable to the pointer variable we use Address of Operator (&):

```
int intVar = 5;
int *ptr = &intVar;
```

In this statement ptr now holds the memory address of an integer variable **‘intVar’**. To access the value at the memory address (currently stored) in the variable we use Dereferencing Operator (*). Do not confuse this with the symbol used for the declaration of a pointer.

```
int intVar2 = *ptr;
```

In this statement another integer variable ‘intVar2’ is now initialized with the value at the memory address which is stored in ptr (that is the value of intVar).

2D Pointer

In C++, a 2D array can be represented using pointers. A 2D pointer is essentially a pointer to an array of pointers, where each pointer points to a separate array representing a row in the 2D array.

Following is an example of declaration of a 2D pointer:

int matrix;**

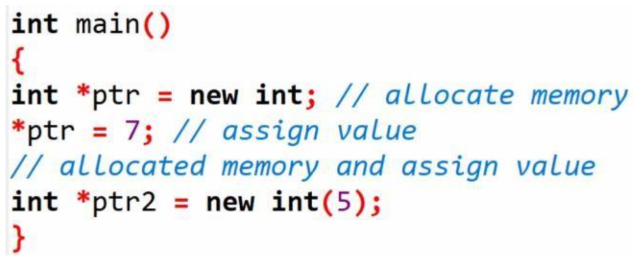
Dynamic Memory Allocation

Variables created during the program execution are called dynamic variables. To create a dynamic variable, we use new operator. C++ supports three types of memory allocation:

1. 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.
2. 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.
3. Dynamic memory allocation is a way for running programs to request memory from the operating system when needed.

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.



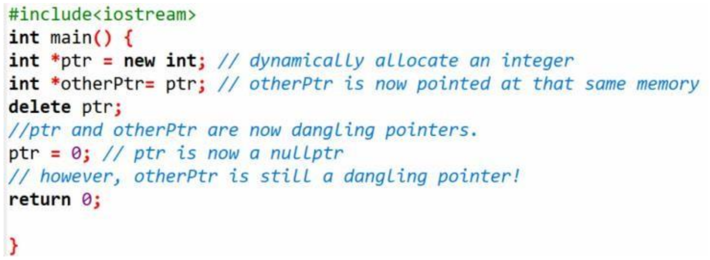
```
int main()
{
int *ptr = new int; // allocate memory
*ptr = 7; // assign value
// allocated memory and assign value
int *ptr2 = new int(5);
}
```

Consider another example below:

new dataType [size]; // to allocate an array of variables.

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. Consider the below example:

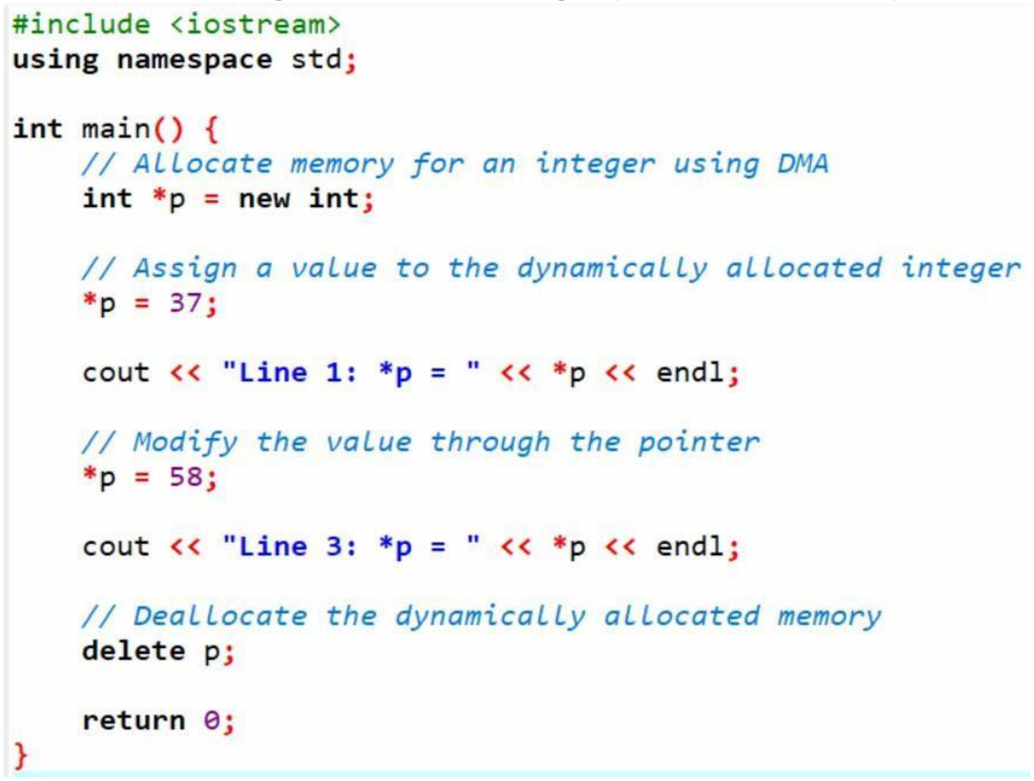


```
#include<iostream>
int main() {
int *ptr = new int; // dynamically allocate an integer
int *otherPtr= ptr; // otherPtr is now pointed at that same memory
delete ptr;
//ptr and otherPtr are now dangling pointers.
ptr = 0; // ptr is now a nullptr
// however, otherPtr is still a dangling pointer!
return 0;

}
```

To delete the dynamically allocated memory we use delete operator. delete operator is used to free the memory which is dynamically allocated using new operator.

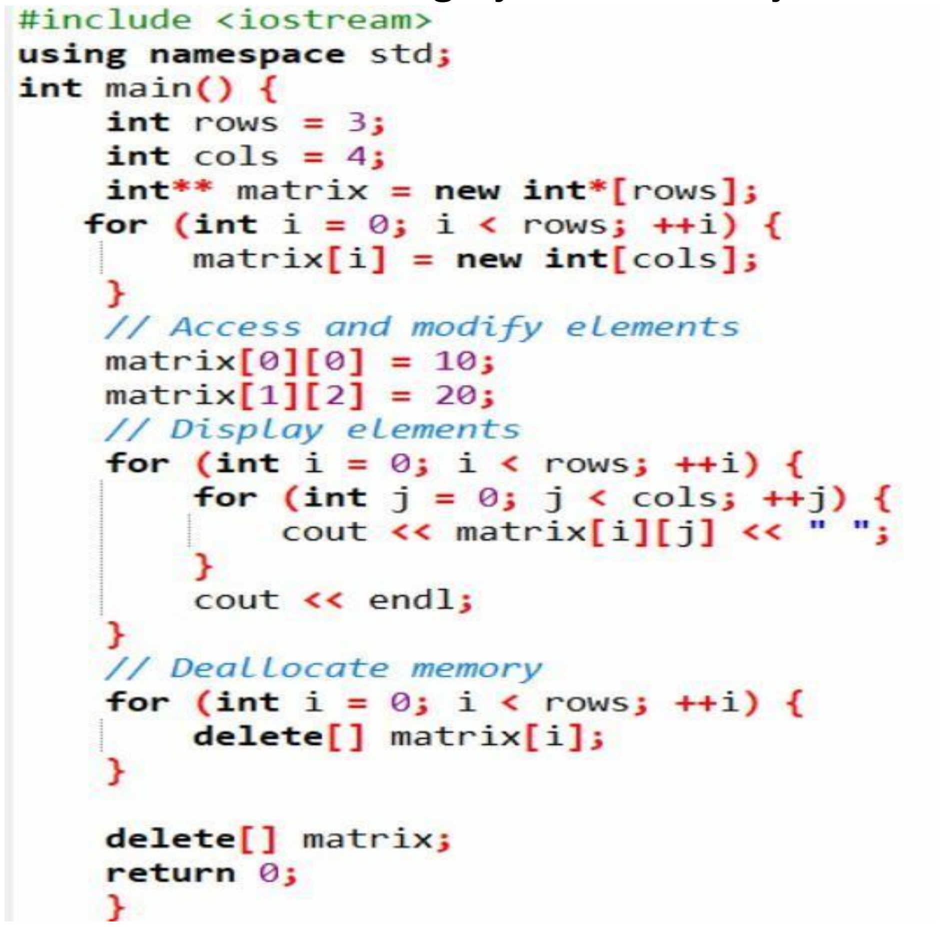
delete ptrVar; //to deallocate single dynamic variable
delete [] ptrArray; //to deallocate dynamically created array.

Example: Code for Single Pointer using Dynamic Memory Allocation



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

int main() {
    // Allocate memory for an integer using DMA
    int *p = new int;

    // Assign a value to the dynamically allocated integer
    *p = 37;

    cout << "Line 1: *p = " << *p << endl;

    // Modify the value through the pointer
    *p = 58;

    cout << "Line 3: *p = " << *p << endl;

    // Deallocate the dynamically allocated memory
    delete p;

    return 0;
}
```

Example: Code for 2D Pointer using Dynamic Memory Allocation



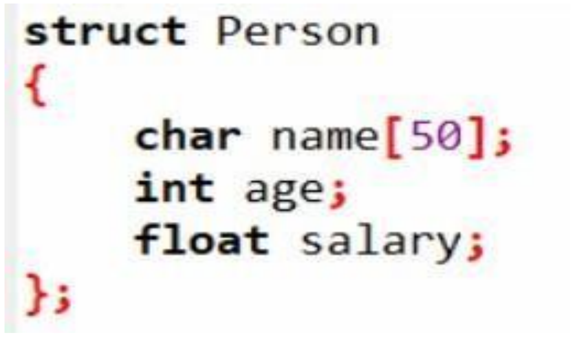
```
#include <iostream>
using namespace std;
int main() {
    int rows = 3;
    int cols = 4;
    int** matrix = new int*[rows];
   for (int i = 0; i < rows; ++i) {
        matrix[i] = new int[cols];
    }
    // Access and modify elements
    matrix[0][0] = 10;
    matrix[1][2] = 20;
    // Display elements
    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < cols; ++j) {
            cout << matrix[i][j] << " ";
        }
        cout << endl;
    }
    // Deallocate memory
    for (int i = 0; i < rows; ++i) {
        delete[] matrix[i];
    }

    delete[] matrix;
    return 0;
    }
```

Struct in C++

Structure is a collection of variables of different data types under a single name. It is similar to a class in that, both holds a collection of data of different data types. For example: You want to store some information about a person: his/her name, citizenship number and salary. You can easily create different variables name, citNo, salary to store this information separately.
However, in the future, you would want to store information about multiple people. Now, you'd need to create different variables for each information per person: name1, citNo1, salary1, name2, citNo2, salary2.
You can easily visualize how big and messy the code would look. Also, since no relation between the variables (information) would exist, it's going to be a daunting task. A better approach will be to have a collection of all related information under a single name Person, and use it for every person. Now, the code looks much cleaner, readable and efficient as well. This collection of all related information under a single name Person is a structure.

Declaring a struct in C++

The struct keyword defines a structure type followed by an identifier (name of the structure). Then inside the curly braces, you can declare one or more members (declare variables inside curly braces) of that structure. For example:



```
struct Person
{
    char name[50];
    int age;
    float salary;
};
```

Here a structure person is defined which has three members: name, age and salary.
When a structure is created, no memory is allocated. The structure definition is only the blueprint for the creating of variables. You can imagine it as a datatype like when you define an integer. The int specifies that a variable can hold integer element only. Similarly, structure definition only specifies that, what property a structure variable holds when it is defined.
Note: Remember to end the declaration with a semicolon (;).

Defining a struct variable in C++

Once you declare a structure person as above. You can define a structure variable as:

Person Ali;

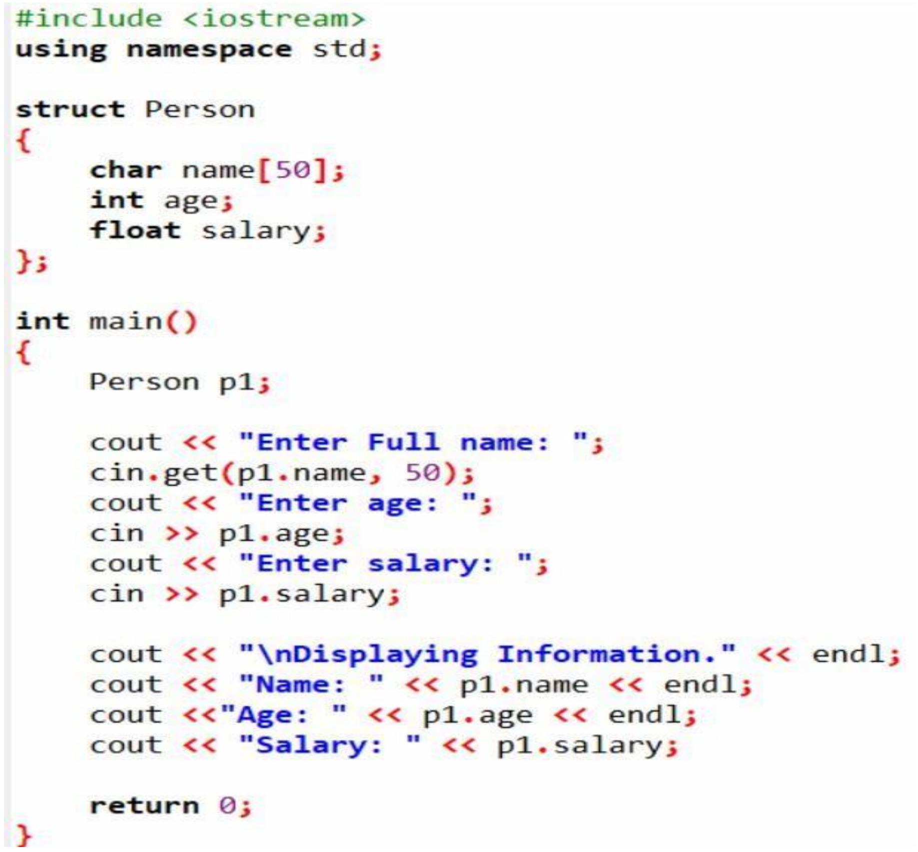
Here, a structure variable bill is defined which is of type structure Person. When structure variable is defined, only then the required memory is allocated by the compiler. Considering you have either 32-bit or 64-bit system, the memory of float is 4 bytes, memory of int is 4 bytes and memory of char is 1 byte. Hence, 58 bytes of memory is allocated for structure variable Ali.

Accessing members of struct in C++

The members of structure variables are accessed using a dot (.) operator. Suppose, you want to access age of structure variable Ali and assign it 50 to it. You can perform this task by using following code below:

Ali.age = 50;

Example: Code for struct in C++

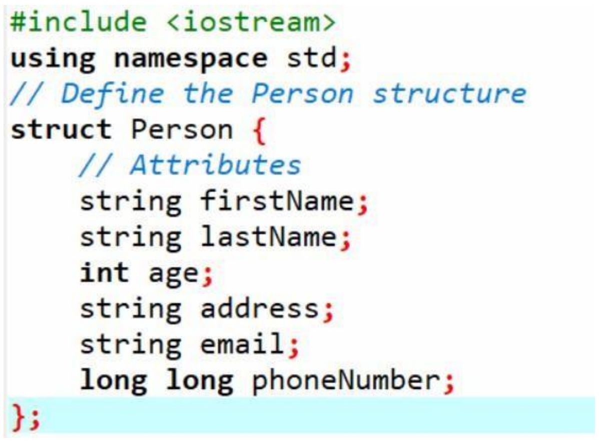


```
#include <iostream>
using namespace std;

struct Person
{
    char name[50];
    int age;
    float salary;
};

int main()
{
    Person p1;

    cout << "Enter Full name: ";
    cin.get(p1.name, 50);
    cout << "Enter age: ";
    cin >> p1.age;
    cout << "Enter salary: ";
    cin >> p1.salary;

    cout << "\nDisplaying Information." << endl;
    cout << "Name: " << p1.name << endl;
    cout <<"Age: " << p1.age << endl;
    cout << "Salary: " << p1.salary;

    return 0;
}
```

Here a structure Person is declared which has three members name, age and salary. Inside main() function, a structure variable p1 is defined. Then, the user is asked to enter information and data entered by user is displayed.

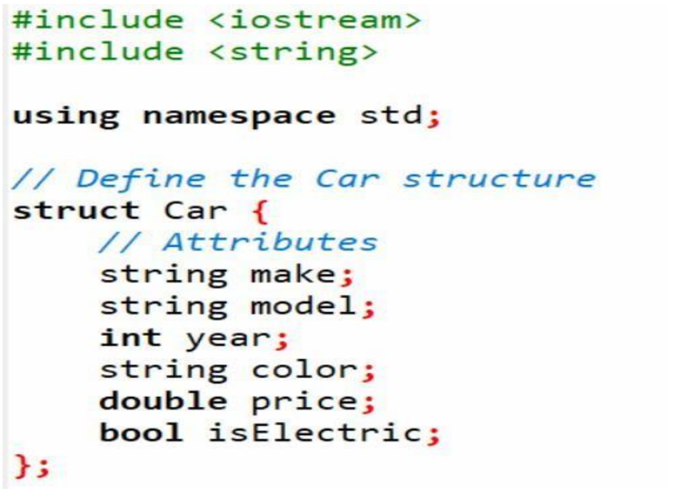
Real World Use Case of struct in C++

Consider a real-world scenario of a "Person" structure in C++ with attributes that might be used in a personal information management system.



```
#include <iostream>
using namespace std;
// Define the Person structure
struct Person {
    // Attributes
    string firstName;
    string lastName;
    int age;
    string address;
    string email;
    long long phoneNumber;
};
```

Consider a different example with a "Car" structure in C++. This structure represents real-world attributes of a car.



```
#include <iostream>
#include <string>

using namespace std;

// Define the Car structure
struct Car {
    // Attributes
    string make;
    string model;
    int year;
    string color;
    double price;
    bool isElectric;
};
```

LAB TASKS:

The Lab Tasks are provided in a separate PDF.