**Object Oriented Programming Lab 02**

|  |  |
| --- | --- |
| **Course**: Object Oriented Programming (CL1004) | **Semester**: Spring 2025 |
| **Instructor**: Shafique Rehman |  |
| 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).

**Void Pointers**

#include <iostream>

using namespace std;

int main()

{

int intvar; //integer variable

float flovar; //float variable

int\* ptrint; //define pointer to int

float\* ptrflo; //define pointer to float

void\* ptrvoid; //define pointer to void

ptrint = &intvar; //ok, int\* to int\*

// ptrint = &flovar; //error, float\* to int\*

// ptrflo = &intvar; //error, int\* to float\*

ptrflo = &flovar; //ok, float\* to float\*

ptrvoid = &intvar; //ok, int\* to void\*

ptrvoid = &flovar; //ok, float\* to void\*

return 0;

}

You can assign the address of intvar to ptrint because they are both type int\*, but you can’t

assign the address of flovar to ptrint because the first is type float\* and the second is type

int\*. However, ptrvoid can be given any pointer value, such as int\*, because it is a pointer to

void

**Pointers to String Constants** Here’s an example, in which two strings are defined, one using array notation, and one using pointer notation:

// strings defined using array and pointer notation

#include using namespace std;

int main() {

char str1[] = “Defined as an array”;

char\* str2 = “Defined as a pointer”;

cout << str1 << endl; // display both strings

cout << str2 << endl; // str1++; // can’t do this; str1 is a constant

str2++; // this is OK, str2 is a pointer

cout << str2 << endl; // now str2 starts “efined...”

return 0; }

**Example: Get even numbers from array to main function**

**#include <iostream>**

**int\* getEvenNumbers(int\* arr, int size, int& resultSize) {**

**static int evenNumbers[100]; // Assuming the result will not exceed 100 even numbers.**

**resultSize = 0;**

**for (int i = 0; i < size; ++i) {**

**if (arr[i] % 2 == 0) {**

**evenNumbers[resultSize++] = arr[i];**

**}**

**}**

**return evenNumbers;**

**}**

**int main() {**

**int arr[] = {1, 2, 3, 4, 5, 6, 7, 8};**

**int size = sizeof(arr) / sizeof(arr[0]);**

**int resultSize = 0;**

**int\* evenNumbers = getEvenNumbers(arr, size, resultSize);**

**std::cout << "Even numbers in the array: ";**

**for (int i = 0; i < resultSize; ++i) {**

**std::cout << evenNumbers[i] << " ";**

**}**

**return 0;**

**}**

**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;**

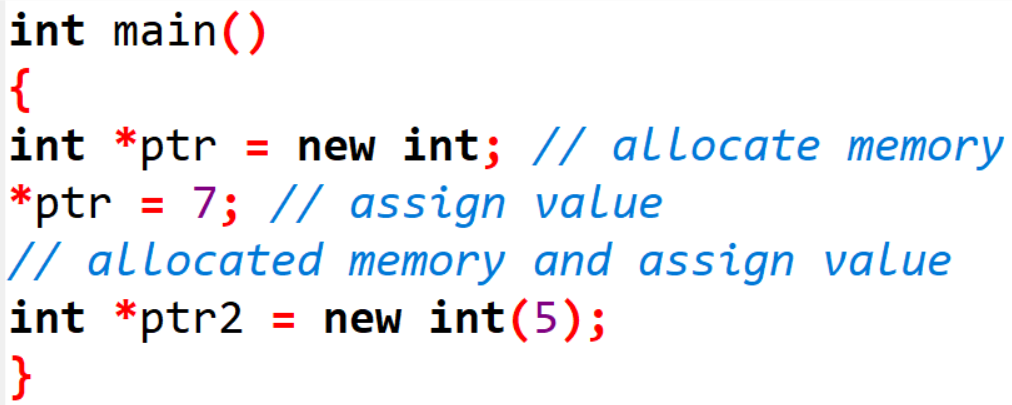
**Dynamic Memory Allocation**

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.

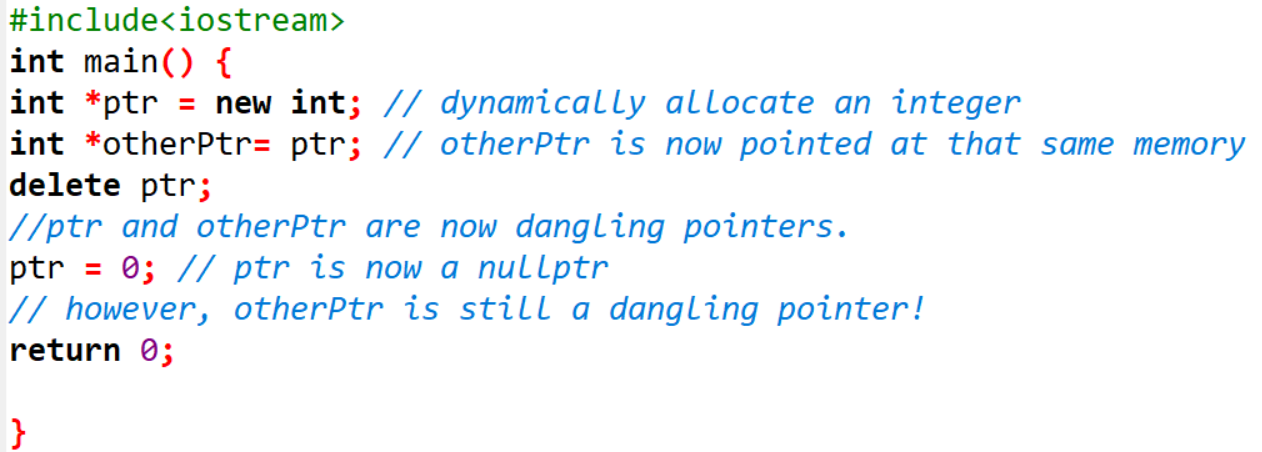


Consider another example below:

**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:

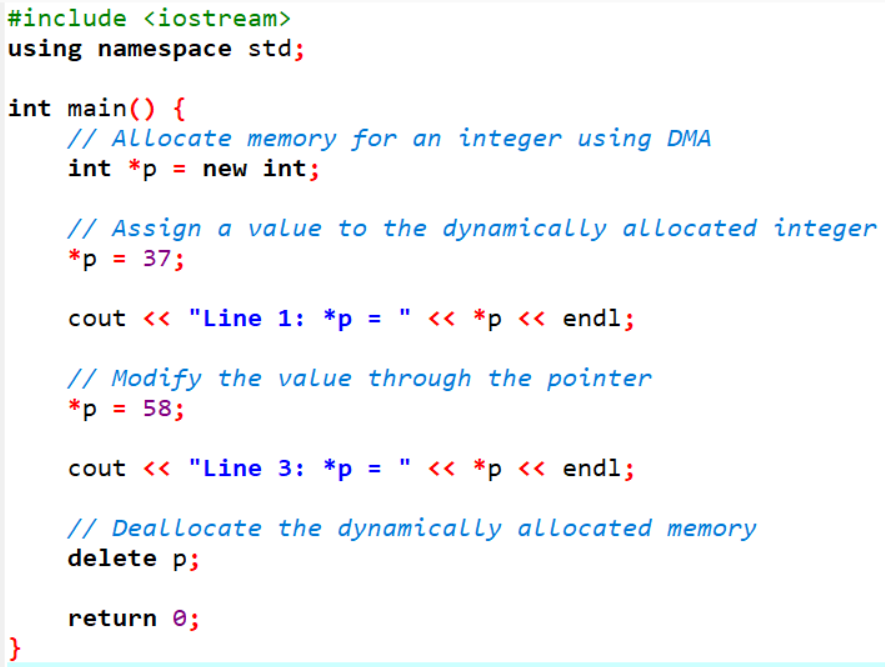


To delete the dynamically allocated memory we use delete operator. delete operator is used to free the memory which is dynamically allocated using new operator.

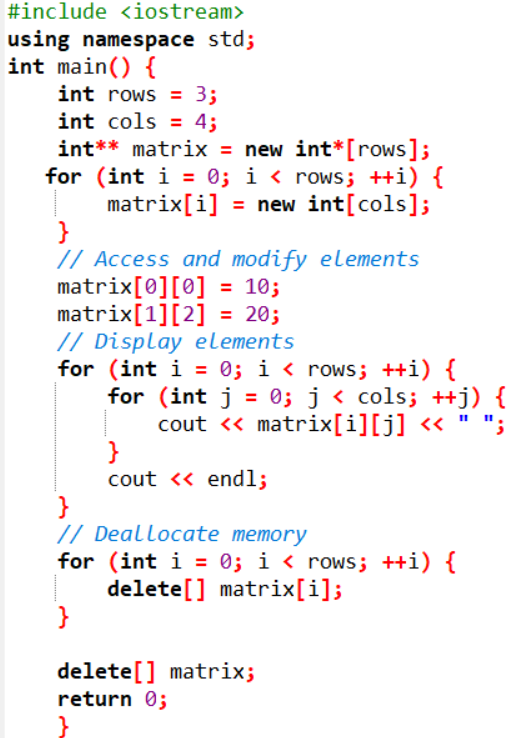
**delete ptrVar; //to deallocate single dynamic variable**

**delete [] ptrArray; //to deallocate dynamically created array.**

**Example: Code for Single Pointer using Dynamic Memory Allocation**



**Example: Code for 2D Pointer using Dynamic Memory Allocation**



**Struct in C++**

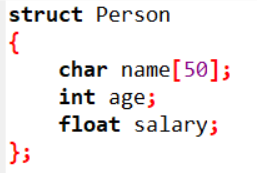
Structure is a collection of variables of different data types under a single name. It is similar to a class in that, both holds a collection of data of different data types. For example: You want to store some information about a person: his/her name, citizenship number and salary. You can easily create 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:

****

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;**

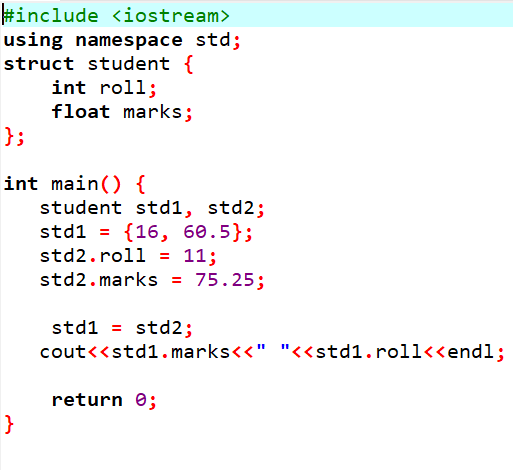
Here, a structure variable bill is defined which is of type structure Person. When structure variable is defined, only then the required memory is allocated by the compiler. Considering you have either 32-bit or 64-bit system, the memory of float is 4 bytes, memory of int is 4 bytes and memory of char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

**Accessing members of struct in C++**

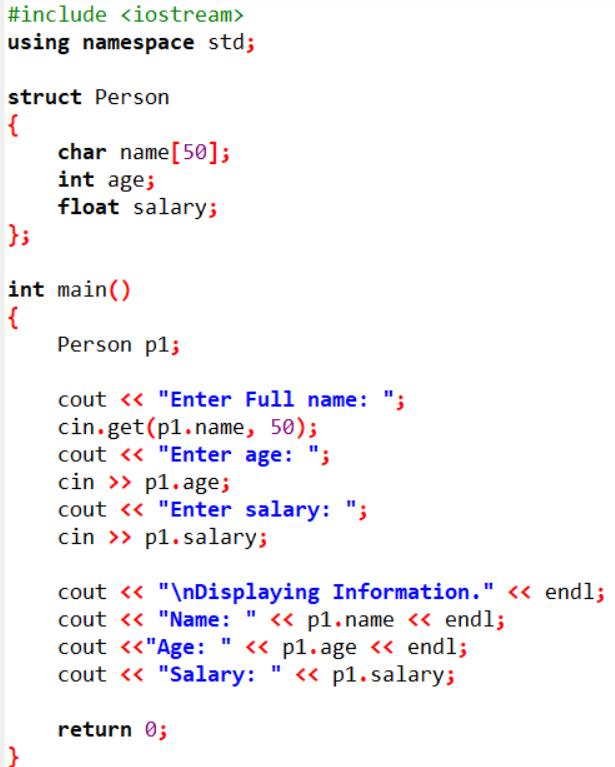
The members of structure variables are accessed using a dot (.) operator. Suppose, you want to access age of structure variable Ali and assign it 50 to it. You can perform this task by using following code below:

**Ali.age = 50;**

**Example:**



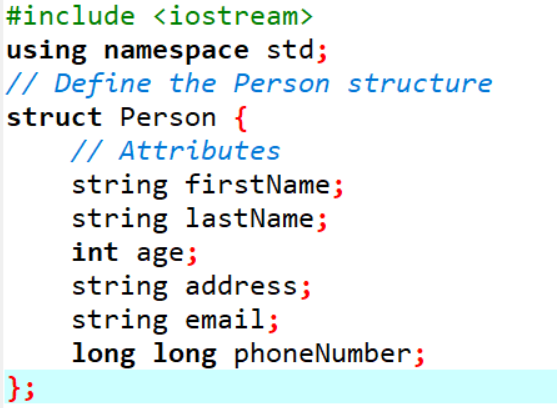
**Example: Code for struct in C++**



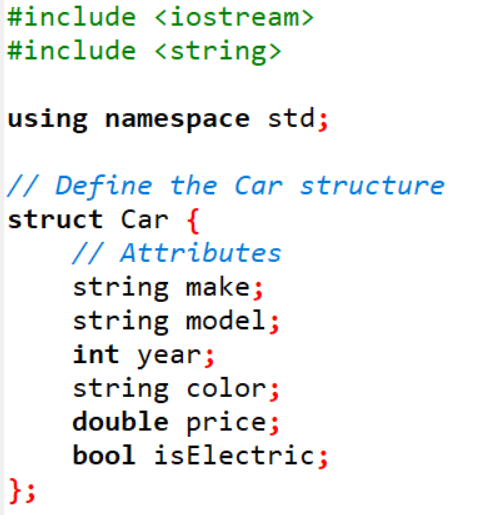
Here a structure Person is declared which has three members name, age and salary. Inside main() function, a structure variable p1 is defined. Then, the user is asked to enter information and data entered by user is displayed.

**Real World Use Case of struct in C++**

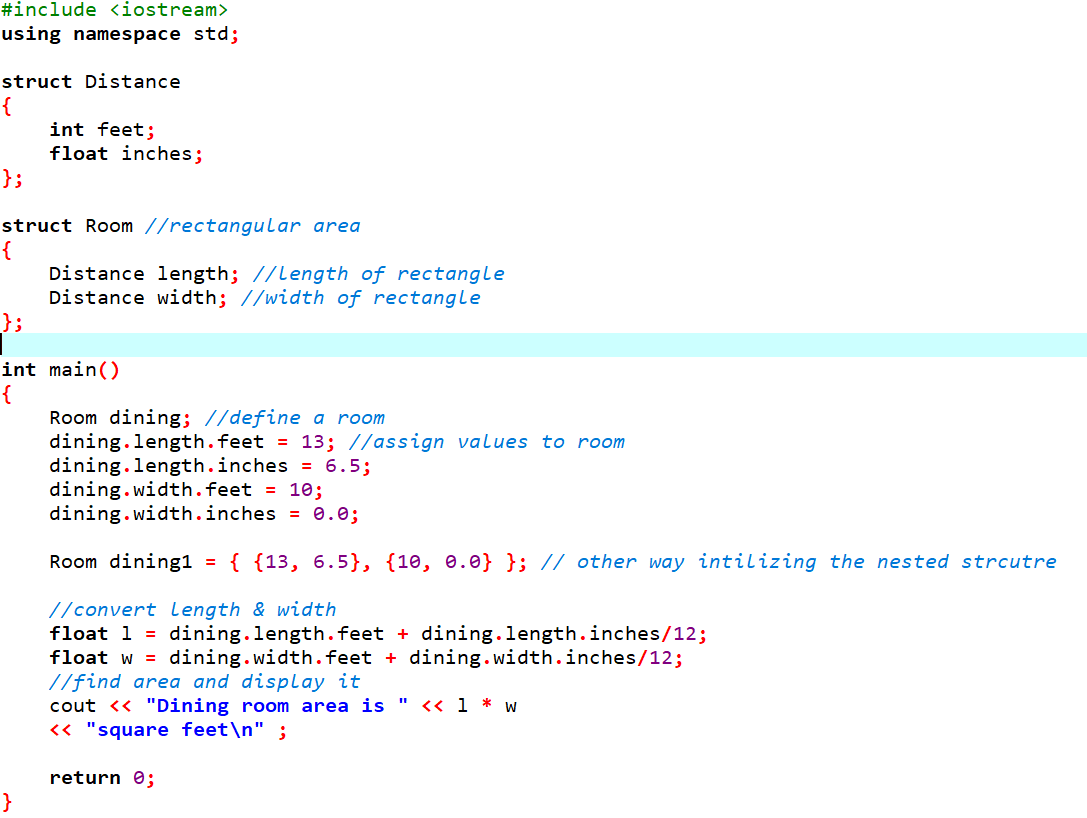
Consider a real-world scenario of a "Person" structure in C++ with attributes that might be used in a personal information management system.



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



**Nested Structure:**



**Lab 02 Exercise**

**Note: Vectors are not allowed in any question**

**Task 1**

Write a C++ program that reads a group of n numbers from the user and stores them in a dynamically allocated array of type float. Then, the program should:

* Calculate the average of the numbers.
* Find the number closest to the average.
* Print the average and the number closest to it.
* Use pointer notation wherever possible.

**Task 2**

Write a C++ program that:

* Reads n strings from the user and stores them in a dynamically allocated array of char\*.
* Prints the strings in reverse order using pointer arithmetic.
* Finds and prints the string with the most vowels (a, e, i, o, u).
* Calculates and prints the average length of all the strings.

Note: Use pointer notation wherever possible.

If there are multiple strings with the same number of vowels, print the first one encountered

**Task 3**

Write a C++ program that:

* Dynamically allocates a 2D array using pointers (not using vector or standard containers).
* Fills the array with random integers between 1 and 100.
* Pass the 2D array to function to perform these tasks:
* Calculates and prints The sum of all elements in the array.
* Calculates and prints The sum of each row and each column.
* Calculates and prints The row and column with the highest sum.
* Pass the 2D array to a function to transpose the matrix and print the resulting matrix.

Free the dynamically allocated memory.

**Note:**

Use functions to perform the calculations and matrix operations (do not write all code inside main()). Handle edge cases, such as when the array has no elements or is improperly allocated.

**Sample Output:**

Original Matrix:

[ 12 35 56 ]

[ 8 45 67 ]

[ 23 54 34 ]

Sum of all elements: 434

Row sums: 103, 120, 111

Column sums: 43, 134, 157

Row with highest sum: Row 2

Column with highest sum: Column 3

Transposed Matrix:

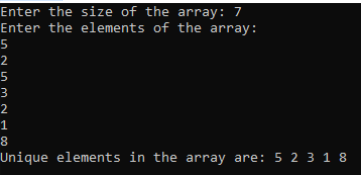
[ 12 8 23 ]

[ 35 45 54 ]

[ 56 67 34 ]

**Task 4**

You are required to write a C++ program that will creates a function named unique that will take array as input . the array may contains the duplicates values but you have to process on the array and have to return the array which must contains only unique values not duplicates.

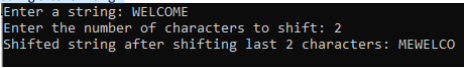


**Task 5**

You are required to write a c++ function swap\_string that shifts the last n characters of a string to the front n times. It will take str and int as parameters. And will return the new string after shifting.

**Note: You have to work with pointers.**

**Output:**



**Task 6**

You are tasked with implementing a simple Student Registration System in C++. Define two structures, Register and Student, where Register contains attributes courseId and courseName, and Student inherits from Register while having additional attributes such as studentId, firstName, lastName, cellNo, and email. Your goal is to create an array of Student structures to store information for five students. Write a **C++ program that accomplishes the following tasks:**

* Implement the Register and Student structures.
* Inherit the Register structure in the Student structure.
* Create an array of Student structures to store information for 5 students.
* Take input for each student, including their courseId, courseName, studentId, firstName, lastName, cellNo, and email.
* Display the information for all 5 students.

**Task 7**

You are tasked with building a simple product management system for an online store.

1. Create a function that allows the addition of a new product to the system. The function should take parameters such as product name, price, quantity, and any other relevant details.
2. Implement a function that takes a product ID as input and displays detailed information about the product, including its name, price, quantity in stock, and any other relevant details.
3. Design a function that enables the update of product information. It should take a product ID as well as the new details (e.g., updated price, quantity, etc.) and modify the existing product's information accordingly.
4. Create a function that removes a product from the system based on its product ID. Ensure that the inventory is updated after the removal.