Object Oriented Programming

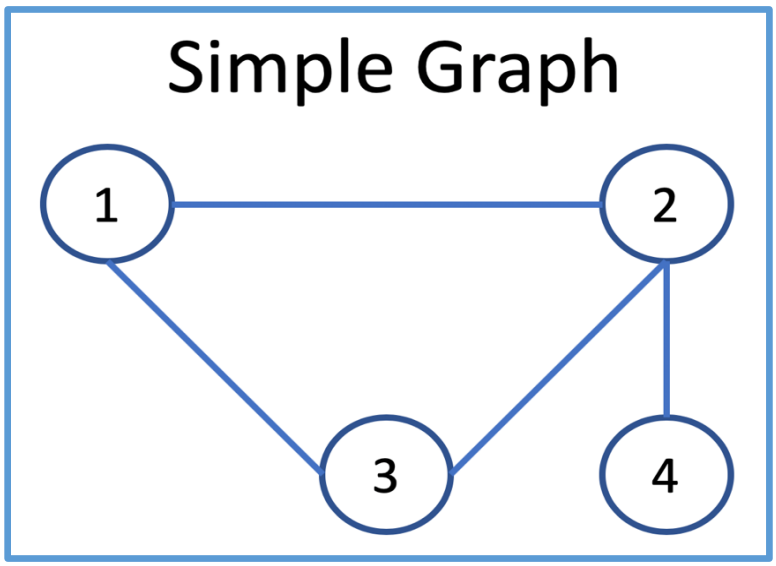
Project

Due Date/Time	4th June, 2023 11:30 PM
Files to be submitted	Documentation along with code file.
File Naming	Roll_No_Group_Member1_ Roll_No _Group_Member2_project.pdf Roll_No_Group_Member1_ Roll_No _Group_Member2_project.pdf .cpp Note: Any assignment that is not according to format will be marked as zero.
Coding Guides	**1.** Use of proper variable declaration/initialization according to the naming conventions (**camelCase, snake_case, PascalCase**) **2.** Use of proper function for each question. Note: Marks will be deducted if not following the above guide line.
Submission Guide line	**1.** Code along with documentations should be submitted on teams by due date/time.
Plagiarism	Any kind of plagiarism will result in F grade in course
Weightage	This assignment will be marked on CLOs, also it will be graded for lab course.

Background

Social networks (Facebook, Twitter, WhatsApp etc) are very common these days and people from all walks of life are using them for variety of purposes. Behind the scene, the data of any social network is modelled as a graph (see example below), where each person serves as a node (node id may be roll number of a student) and relationship between any two persons is represented as an edge.

In this assignment, your job is to apply the concepts of OOP learnt in the class to solve following problems.



Classes to be created

	SimpleGraph.cpp	SimpleNode.cpp
Member Data	"numNodes" of type int to store total number nodes	(i) "nodeId" of type int to store id of a node object. (ii)"NeighborCount" of type int to store count neighbors of that node. (iii)An Array, named "arrNeighbors", of type "SimpleNode" to store the neighbors of each a node.
	"numEdges" of type int to store total number of edges	
	An array of type "SimpleNode" to store all the created nodes	
Member Functions	**Overloaded Constructor** to initialize the member data. A message should be printed for this action.	**Constructor** to initialize the member data. A message should be printed for this action. **Overloaded Constructor:** to initialize the private members
	Destructor to drop the graph object. A message should be printed for this action.	**Destructor** to drop the node object. A message should be printed for this action.
	Name: addNode() **Parameter**: nodeId **Return Type**: void **Purpose**: creates an object of type SimpleNode and assign nodeId to object	**Name**: addEdge() **Parameter**: const SimpleNode& n **Return Type**: void **Purpose**: adds an edge between caller and a node passed as parameter. An edge is stored in "arrNeighbors"
	Name: addEdge() **Parameter**: nodeid1, nodeid2 **Return Type**: void **Purpose**: To get the objects of An edge against Node id's from SimpleNode array, and call addEdge() of SimpleNode class.	**Name**: getneighborcount() **Parameter**: None **Return Type**: int **Purpose:** to get the count of neighbor
	Name: printNeighbors() **Parameter**: Node id **Return Type**: none **Purpose**: prints all the neighbors of an input nodes.	**Name**: getneighbor() **Parameter**: None **Return Type**: SimpleNod type **Purpose:** to get the arrneighbor
	Name: printGraphData() **Parameter**: none **Return Type**: none **Purpose**: prints all the nodes along with their neighbors.[***Hint:** you can use print neighbor functionality]	Create setter and getter functions for the private data memebers

Question No.1:

Implement the above mentioned classes and their member functions. Make separate class.h, class .cpp, and **one main.cpp file for all questions**. [Also ensure to use correct access modifier/access specifier for each class]

Create an object of type SimpleGraph in main function

Execute a loop which asks the user to enter node id as an integer value like "cin>>nodeId1"

a. Create an object of type Node using constructor and **addNode**() for nodeId1

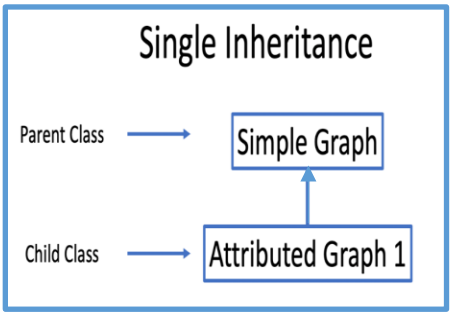
b. Create an object of type Node using constructor and **addNode**() for nodeId2 and so on..
Call **addEdge** function in **main.cpp** multiple times to create an edge between any 2 of created nodes.

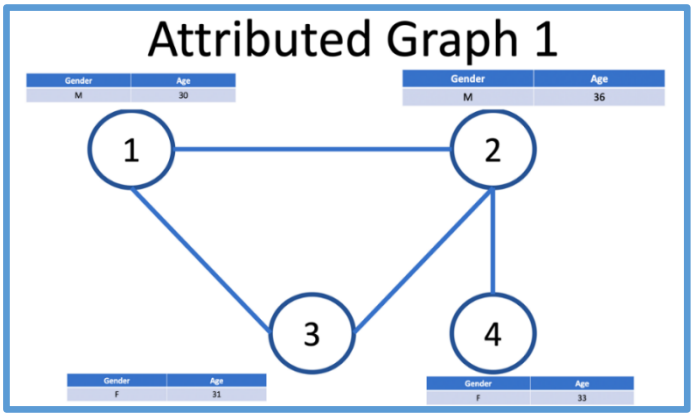
Call **printNeighbors** function in **main.cpp** with different Nodeid's to print its neighbors.

Call **printGraphData** function in **main.cpp** to print all the created nodes of class SimpleNode along with their neighbors.

[You can use this link for initializing of array of object with overloaded constructor]

*https://www.includehelp.com/code-snippets/initialization-of-array-of-objects-with-parameterized-constructor-in-cpp-program.aspx



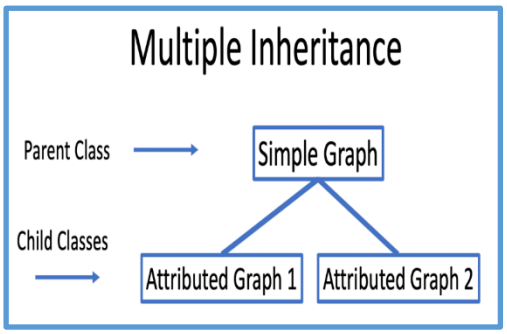


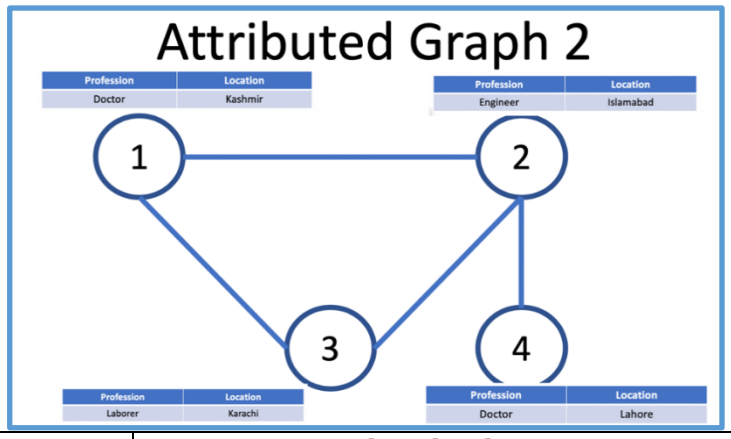
	AttributedGraph1.cpp	**AttributedNode1.cpp**
Purpose	Create a class AttributeGraph1 that extends the SimpleGraph	Create a class named AttributedNode1.cpp
Member Data	An array of type "AttributedNode1" to store all the created nodes	Char variable to store Gender Int variable to store Age
Member Functions	**Constructor** to initialize the member data. A message should be printed for this action.	**Constructor** to initialize the member data. A message should be printed for this action.
	Destructor to drop the graph object. A message should be printed for this action.	**Destructor** to drop the node object. A message should be printed for this action.
	Name: appendAttributes () **Parameter**: None **Return Type**: void **Purpose**: iterate a loop over array of SimpleNode to read the NodeID's and ask user to add attributes (Gender, Age) into "AttributeNode1" array on same index as of SimpleNode array.	Create **setter** and **getter** functions for the private data memebers
	Name: printGraphData() **Parameter**: none **Return Type**: none **Purpose**: prints data of all nodes of AttributedGraph1 along with their neighbors.	

Question No.2:

1. Implement the above mentioned classes and their member functions. Make separate class.h, class .cpp, for each class. [* Also ensure to use correct access modifier/access specifier for each class]
2. Create **setter** and **getter** functions of data members.
3. Create an object of type AttributedGraph1 (child class) in main function and **check the order of constructors and destructors** in 1 level inheritance.
4. Call **appendAttribute**() function in **main.cpp** to add the age and gender to already created NodeID's.
5. Call **printGraphData** function in **main.cpp** to print the data of all nodes of **AttributeGraph1** along with their neighbors.[* Apply runtime polymorphism for printGraphData() which override the SimpleGraph's function]

Note: [If the object that the pointer is pointing to is deleted, and the destructor is not set to virtual, then the base class destructor will be called instead of the derived class destructor. This can lead to a memory leak. So you are required to deallocate the memory correctly by making virtual destructor in both classes i.e. base and derived clasess]





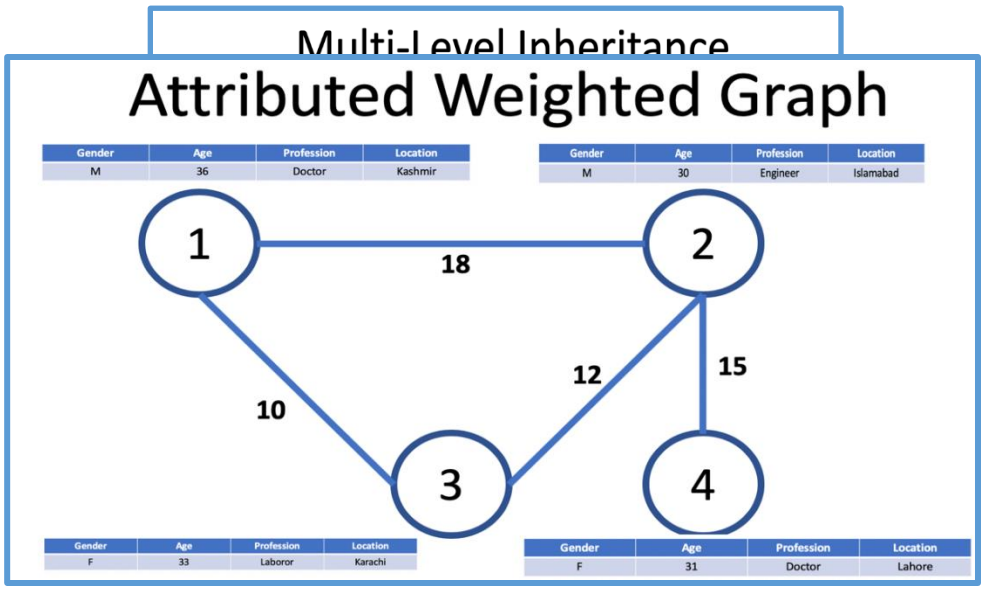
		AttributedNode2.cpp
Purpose	Create a class AttributeGraph2 that extends the SimpleGraph	Create a class named AttributedNode2.cpp
Member Data	An array of type "AttributedNode2" to store all the created nodes	String variable to store Profession String variable to store CityName
	Create a class AttributeGraph2 that extends the SimpleGraph	
Member Functions	**Constructor** to initialize the member data. A message should be printed for this action. **Overloaded Constructor** to initialize the member data. A message should be printed for this action **[*** you have to explicitly mention to call the Base class's parameterized constructor when Derived class's parameterized constructor is called to maintain order **]**	**Constructor** to initialize the member data. A message should be printed for this action.
	Destructor to drop the graph object. A message should be printed for this action.	**Destructor** to drop the node object. A message should be printed for this action.

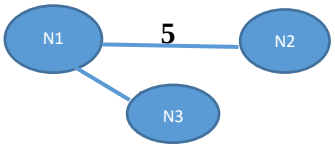
	Name: appendAttributes () **Parameter**: nodeId **Return Type**: void **Purpose**: add the attributes (Profession, Location) to already created objects of class SimpleGraph	Create **setter** and **getter** functions for the private data memebers
	Name: printData() **Parameter**: none **Return Type**: none **Purpose**: prints all the nodes along with their neighbors and attributes **Note**: Avoid code repetition by using inheritance	
	Name: printNeighbors() **Parameter**: Node id **Return Type**: none **Purpose**: prints all the neighbors of an input node and attributes	

Question No.3:

Implement the above mentioned classes and their member functions. Make separate class.h, class .cpp, main.cpp file for each class. [* Also ensure to use correct access modifier/access specifier for each class]

1. Create **setter** and **getter** functions of data members.
2. Create an object of type AttributedGraph2 in main function.
3. Call **appendAttribute**() function in **main.cpp** to add the profession and location to already created NodeID's.
4. Call **printGraphData()** function in **main.cpp** to print the data of all nodes of **AttributeGraph2** along with their neighbors.[* Apply runtime polymorphism for printGraphData() which override the SimpleGraph's function]



<table>
<tr><td></td><td>AttributedWeightedGraph.cpp</td></tr>
<tr><td>Purpose</td><td>Create a class named AttributedWeightedGraph.cpp which extends AttributedGraph1.cpp , AttributeGraph2.cpp</td></tr>
<tr><td>Member Data</td><td>A 2D-array of type int to store all the edge weights of NodeID</td></tr>
<tr><td>Member Functions</td><td>Constructor to initialize the member data. A message should be printed for this action.</td></tr>
<tr><td></td><td>Destructor to drop the node object. A message should be printed for this action.</td></tr>
<tr><td></td><td>Name: appendWeight ()
Parameter: nodeId1,nodeId2,Weight
Return Type: void
Purpose: adds a weight to an edge between given 2 nodes
 

<table>
<tr><td></td><td>IndexOfN1</td><td>IndexOfN2</td><td>IndexOfN3</td></tr>
<tr><td>IndexOfN1</td><td>0</td><td>5</td><td>2</td></tr>
<tr><td>IndexOfN2</td><td>5</td><td>0</td><td>0</td></tr>
<tr><td>IndexOfN3</td><td>2</td><td>0</td><td>0</td></tr>
</table>
*
Undirected Graph weight for N1-N2 or N2-N1 will be same
* No weight will be assign to N2-N3 as they not edge in-between.
* Value zero depict there is no edge between the involving Nodes at indexes.</td></tr>
<tr><td></td><td>Name: printNeighbors()
Parameter: Node id
Return Type: none
Purpose: prints all the neighbors along with weight of an input node and attributes
Note: Avoid code repetition by using inheritance</td></tr>
<tr><td></td><td>Name: printData()
Parameter: none
Return Type: none
Purpose: prints all the nodes along with their neighbors, their weights, and attributes
Note: Avoid code repetition by using inheritance</td></tr>
</table>

Question No.4:

Implement the above mentioned class and their member functions. Make separate class.h, class.cpp file for the class. [* Also ensure to use correct access modifier/access specifier for each class]

1. Create **setter** and **getter** functions of data members.
2. Create an object of type **AttributedWeightedGraph** in main function:
3. Call **appendWeight**() function in **main.cpp** to add weight on an edge between two nodes. For this purpose find the indexes of Nodes and fill 2D with the weights. Execute this function multiple times to appendweights on all created edges of graph.
4. Call **printGraphData** function in **main.cpp** to print the data of all nodes.

Question No.5:

A class is made abstract by declaring at least one of its functions as **pure virtual** function. A pure virtual function is specified by placing "= 0" in its declaration as follows.

1. You are required to add the **pure virtual function** of **sendMessage** () in the base class (simpleGraph) and implementations of this pure virtual function provided in the derived classes.
2. Call **Print**() function in main.cpp to print the data of the function through the object of the derived class.

Bonus Task:

Bonus Task 1: Add a function to find path between any 2 nodes, received as input from the user
Bonus Task 2: Add a function to find path between any 2 nodes, received as input from the user, where sum of weights of all the member edges is higher
Bonus Task 3: You can use any graph visualization tool (like Cytoscape which is free to download and easy to play with) to visualize the graphs, paths found, the communities found and so on.

END