NATIONAL UNIVERSITY OF COMPUTER & EMERGING SCIENCES

ISLAMABAD CAMPUS

OBJECT ORIENTED PROGRAMMING (CS217) - SPRING 2020

ASSIGNMENT-5

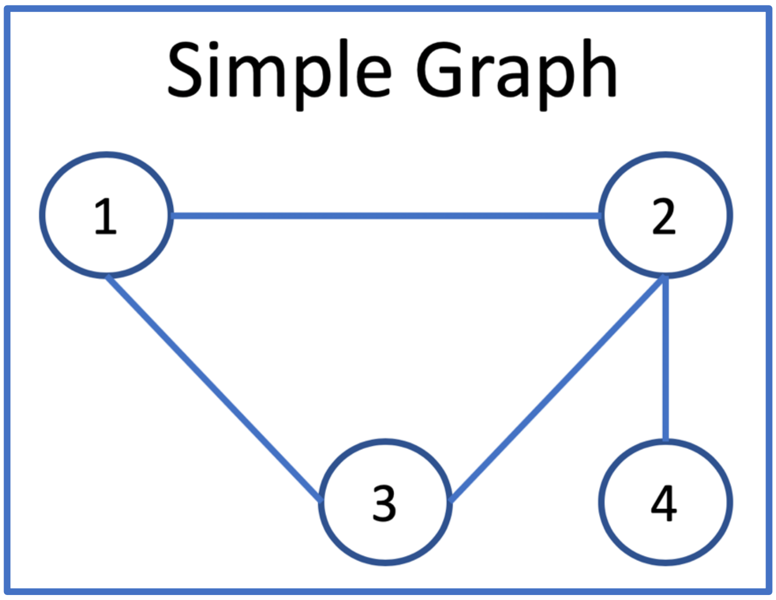
Due Date: April 23, 2020 (05:00 pm) Instructions:

1. Make sure that you read and understand each and every instruction. If you have any questions or comments you are encouraged to discuss with your colleagues and instructors on piazza.
2. Create each problem solution in a separate .cpp file, i.e. you must name the file containing solution of Q1 as 'q1.cpp', Q2 as 'q2.cpp' and Q3 as 'q3.cpp'. Combine all your work in one .zip file.

   – Name the .zip file as ROLL-NUM SECTION.zip (e.g. 19i-0001 B.zip).
   – Submit the .zip file on Google Classroom within the deadline.
   Start early otherwise you will struggle with the assignment. You must follow the submission instructions to the letter, as failing to do so will get you a zero in the assignment.
3. Plagiarism is strongly forbidden and will be very strongly punished. If we find that you have copied from someone else or someone else has copied from you (with or without your knowledge) both of you will be punished. You will be awarded straight zero in this assignment or all assignments.

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.



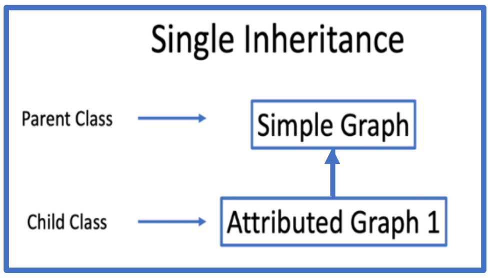
Classes to be created

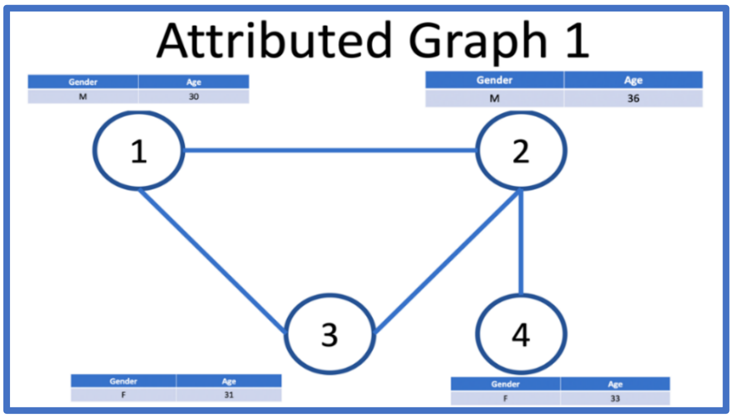
	SimpleGraph.cpp	SimpleNode.cpp
Member Data	“numNodes” of type int to store total number nodes “numEdges” of type int to store total number of edges “allnodes” An array of type “SimpleNode” to store all the created nodes	(i) “nodeId” of type int to store id of a node object. (ii)”NeighborCount” of type int to store count neighbors of that node. (iii)An Array, named “arrNeighbors”, of type “SimpleNode” to store the neighbors of each a node.
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 main.cpp file for the question. [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



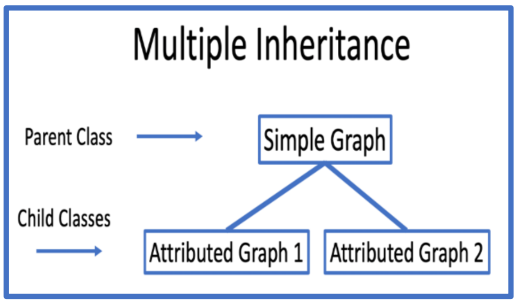


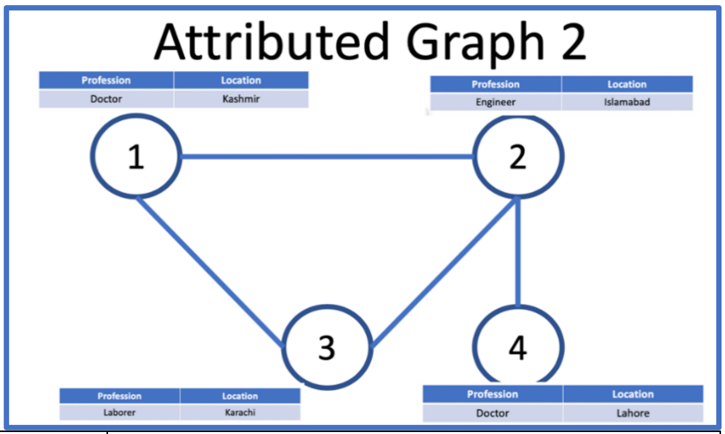
	AttributedGraph1.cpp	AttributedNode1.cpp
Purpose	Create a class AttributeGraph1 that extends the SimpleGraph	Create a class named AttributedNode1.cpp
Member Data	An array of type "AttributedNode1" to store all the created nodes	Char variable to store Gender Int variable to store Age
Member Functions	**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.
	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:
Implement the above mentioned classes and their member functions. Make separate class.h, class .cpp and main.cpp for the question. [* 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 AttributedGraph1 (child class) in main function and **check the order of constructors and destructors** in 1 level inheritance. Also use this object to add nodes & edges using SimpleGraph's methods.
3. Call **appendAttribute**()function in **main.cpp** to add the age and gender to already created Nodeid's.
4. 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: [Identify the issue and resolve it accordingly when the object that the pointer is pointing to is deleted, it calls base class destructor instead of the derived class destructor]

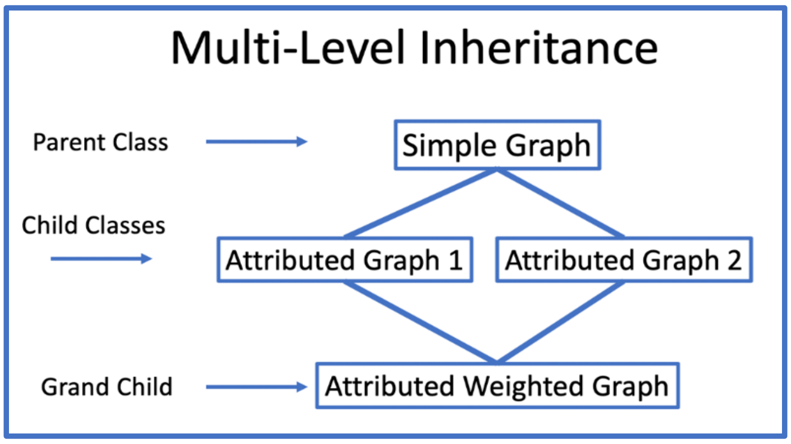


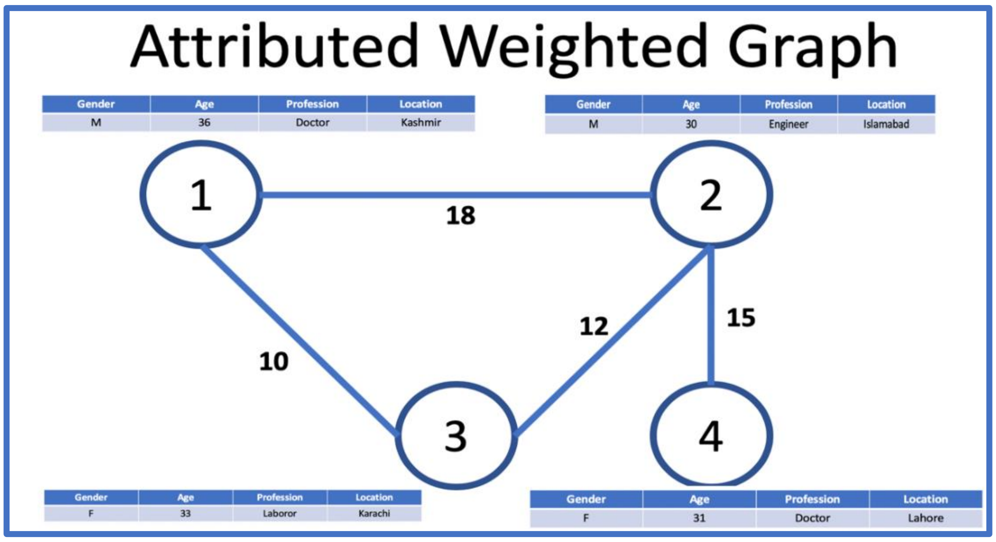


	AttributeGraph2.cpp	**AttributedNode2.cpp**
Purpose	Create a class AttributeGraph2 that extends the SimpleGraph	Create a class named AttributedNode2.cpp
Member Data	An array of type "AttributedNode2" to store all the created nodes	String variable to store Profession String variable to store CityName
	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 maintain order of overloaded constructors for both the based and derived class **]**	**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 (Profession, location) into "AttributeNode2" array on same index as of SimpleNode array.	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 the question. [* 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. Also use this object to add nodes & edges using SimpleGraph's methods.
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]
5 Call **printNeighbors()** function in **main.cpp** to print all the neighbors of an input node and attributes





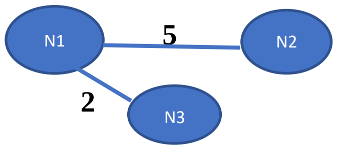
Gender	Age	Profession	Location
M	36	Doctor	Kashmir

Gender	Age	Profession	Location
M	30	Engineer	Islamabad

Gender	Age	Profession	Location
F	33	Laboror	Karachi

Gender	Age	Profession	Location
F	31	Doctor	Lahore

	AttributedWeightedGraph.cpp
Purpose	Create a class named AttributedWeightedGraph.cpp which extends AttributedGraph1.cpp , AttributeGraph2.cpp
Member Data	A 2D-array of type **int** to store all the edge weights of **NodeID**
Member Functions	**Constructor** to initialize the member data. A message should be printed for this action.

	Destructor to drop the node object. A message should be printed for this action.
	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 there is no edge in-between.** *** Value zero depict there is no edge between the involving Nodes at indexes.**
	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
	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

Question No.4:
Implement the above mentioned class and their member functions. Make separate class.h, class.cpp file main.cpp for the question. [* 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. Also use this object to add nodes & edges using SimpleGraph's methods.
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 **printNeighbors**() function in **main.cpp** to print the neighbor of a given id.
5. Call **printGraphData** function in **main.cpp** to print the data of all nodes.

Self-Learning Tasks:

Task 1: Add a function to find path between any 2 nodes, received as input from the user
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
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.

***********Good Luck *************