

**CC-211L**

**Object Oriented Programming**

**Laboratory 02**

**Introduction to Classes Objects and Member Functions - I**

**Version: 1.0.0**

**Release Date: 03-01-2023**

**Department of Information Technology**

**University of the Punjab**

**Lahore, Pakistan**

**Contents:**

- Learning Objectives
- Required Resources
- General Instructions
- Background and Overview
  - Class
  - Object
  - Concept of Encapsulation
  - Concept of Abstraction
  - Comparison between Class and Object
- Activities
  - Pre-Lab Activity
    - Introduction to Getter/Setter Member Functions
    - Example code on Getter and Setter Member Functions
    - Explanation of Public-private access specifiers
    - Demonstration of Encapsulation and Abstraction using Getter/Setter functions
    - Application of Access Specifiers in a sample program
  - In-Lab Activity
    - Constructor
    - Default Constructor
    - Example Code on Default Constructor
    - Parameterized Constructor
    - Example Code on Parameterized Constructor
    - Copy Constructor
    - Overloaded Constructor
    - Example Code on Overloaded Constructor
    - Destructor
    - Example code on Destructor
    - Task 1: Class Definition and Implementation
    - Task 2: Constructor Implementation
    - Task 3: Constructor Implementation
    - Task 4: Constructor Implementation
  - Post-Lab Activity
    - Task 01: String Formatting
- Submissions
- Evaluations Metric
- References and Additional Material
- Lab Time and Activity Simulation Log

**Learning Objectives:**

- Concept of Encapsulation and Abstraction
- Getter/Setter Member Functions
- Public private access specifiers
- Constructors
- Overloaded Constructor
- Default Constructor
- Destructor

**Resources Required:**

- Desktop Computer or Laptop
- Microsoft ® Visual Studio 2022

**General Instructions:**

- In this Lab, you are **NOT** allowed to discuss your solution with your colleagues, even not allowed to ask how is s/he doing, this may result in negative marking. You can **ONLY** discuss with your Teaching Assistants (TAs) or Lab Instructor.
- Your TAs will be available in the Lab for your help. Alternatively, you can send your queries via email to one of the followings.

Teachers:		
Course Instructor	Prof. Dr. Syed Waqar ul Qounain	<a href="mailto:swjaffry@pucit.edu.pk">swjaffry@pucit.edu.pk</a>
Teacher Assistants	Saad Rahman	<a href="mailto:bsef19m021@pucit.edu.pk">bsef19m021@pucit.edu.pk</a>
	Zain Ali Shan	<a href="mailto:bcsf19a022@pucit.edu.pk">bcsf19a022@pucit.edu.pk</a>

## **Background and Overview:**

### **Abstraction:**

Abstraction in C++ is the process of hiding the implementation details from the user. It allows the user to focus on the functionality of an object without worrying about its internal implementation. Abstraction is achieved in C++ through the use of classes and objects, which can be used to define data and functions that can be used to manipulate the data. By hiding the implementation details, the user is able to use the object without having to worry about its internal working. This helps make code easier to understand and maintain.

### **Encapsulation:**

Encapsulation in C++ is the process of combining data and functions into a single unit, called a class. This process allows data and functions to be accessed and used together while keeping the data hidden from outside users. Encapsulation is a powerful concept that helps keep code organized, secure, and manageable. By using encapsulation, developers can define the interactions between different parts of a program in a clear and concise way. This helps to reduce potential errors and maintain the program's functionality over time.

### **Class:**

A class is an important concept in Object Oriented Programming (OOP). It is a blueprint that is used to create objects, which can hold data and perform functions. A class is made up of two parts: attributes and methods. Attributes are the properties of the class that describe its characteristics, such as its name, size, and color. Methods are the functions that the class can perform, such as adding, deleting, or editing data. Classes are reusable and allow for code to be written more efficiently. They are also a key tool for creating complex applications.

### **Object:**

Objects are instances of classes, which are templates for creating objects. Objects have properties and methods that define their state and behavior, respectively. It is a self-contained entity that contains both data and instructions for manipulating that data. Objects are used in object-oriented programming (OOP) to represent real-world entities such as people, places, and things. Objects contain data and methods that describe the object's characteristics and behavior. The data is stored in the form of properties, while the methods are functions that can access and modify the object's data.

### **Comparison Between class and object:**

Objects are often referred to as the "nouns" of programming, while classes are the "verbs". Objects are tangible, physical objects in the real world, while classes are abstract concepts that define how objects should behave.

## Activities:

### Pre-Lab Activities:

#### Getter/Setter Member Functions:

Getter/Setter member functions are used to control access to class data members in C++. Getters are used to retrieve data from the class, and setters are used to assign values to the class. They allow for the encapsulation of data within a class, which makes it easier to manage class data and ensures that data is used correctly.

#### Example:

For example, if a class has a data member called "age", it is best to create a getter and setter function for that data member. The getter function would return the age of the class object, and the setter function would allow us to set the age of the class object. This way, we can ensure that the data member is being properly used and make sure it is not being passed invalid data.

#### Syntax of Getters and Setters in C++:

##### Getter:

The syntax of Getter method is as follows

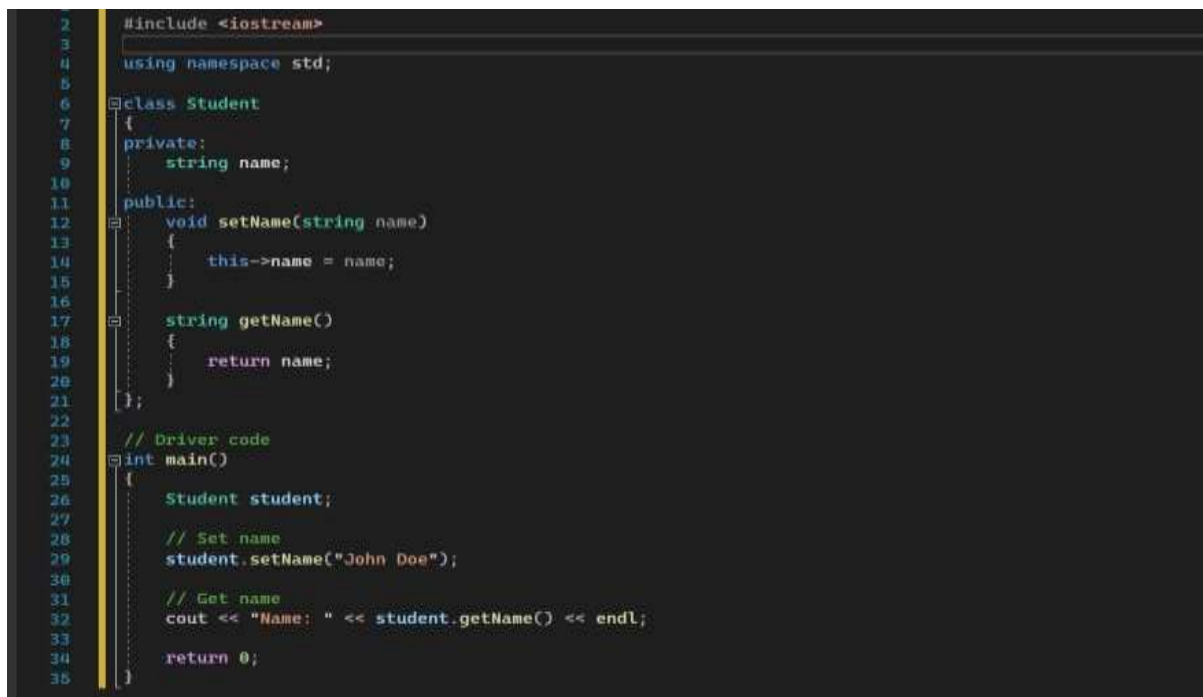
```
Type ClassName::getPropertyName(){
    return propertyName;
}
```

##### Setter:

The syntax of Setter method is as follows

```
void ClassName::setPropertyName(Type propertyName){
    this->propertyName = propertyName;
}
```

#### Example Code:



```
1  #include <iostream>
2
3  using namespace std;
4
5  class Student
6  {
7  private:
8      string name;
9
10 public:
11     void setName(string name)
12     {
13         this->name = name;
14     }
15
16     string getName()
17     {
18         return name;
19     }
20 };
21
22 // Driver code
23 int main()
24 {
25     Student student;
26
27     // Set name
28     student.setName("John Doe");
29
30     // Get name
31     cout << "Name: " << student.getName() << endl;
32
33     return 0;
34 }
```

Fig. 01 (Getter and Setter Demonstration)

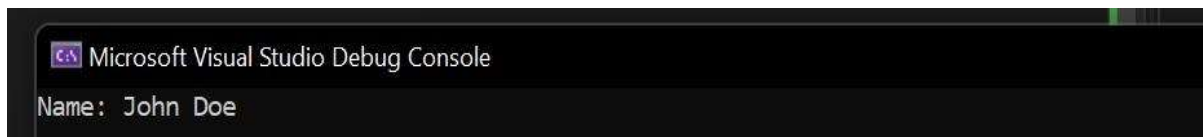
**Output:**

Fig. 02 (Output with respect to figure 1)

**Some Key points about getter and setter methods:**

- Getters and setters are used to access private member variables of a class from outside the class.
- Getters and setters should be used to ensure data integrity in the class by validating the input data and restricting access to certain areas of the class.
- Getters and setters should not be used to perform business logic or complex operations on private member variables.
- Getters should always return a copy of the private member variable and not the variable itself.
- Setters should always be used to modify the private member variable and not perform any other operations on the variable.
- Getters and setters should have meaningful names that clearly describe the purpose of the methods.
- Getters and setters should be declared as public methods of the class.

**When to Use Getters and Setters in C++:**

Getters and setters should be used when you want to maintain control over how users access and set the data members of a class. This allows you to control how the data is accessed and can be used to validate user input before setting a value or to perform calculations based on user input before setting a value. Getters and setters also make it easier to maintain the code, as the logic for how the data is accessed and set is self-contained.

**Public/Private Access Specifiers:****Access Specifiers:**

Access specifiers in C++ can be used to control the access level of the data members of a class. The three access specifiers used in C++ are public, protected, and private. In this introduction, we will discuss the purpose of public and private access specifiers and how to use them in C++.

**Public Access Specifiers:**

Public data members can be accessed and modified from anywhere within the program. To make a member of a class public, use the keyword “public” before the member declaration.

**Private Access Specifiers:**

Private data members can only be accessed and modified by the member functions of the class. They cannot be accessed or modified from outside the class. To make a member of a class private, use the keyword “private” before the member declaration.

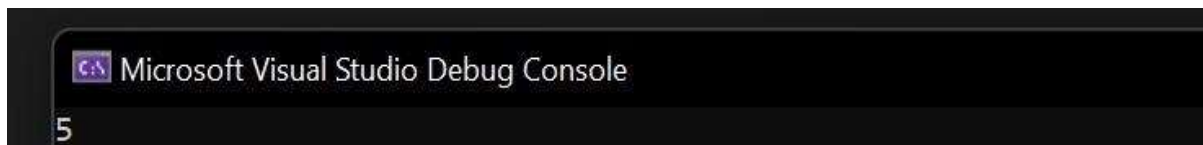
## How to Use Public and Private Access Specifiers:

### Example Code for Public and Private Access Specifiers:

```
1  #include <iostream>
2  using namespace std;
3
4  // Public access specifier
5  class PublicClass
6  {
7  public:
8      int publicVariable;
9      void setPublicVariable(int variableValue)
10     {
11         publicVariable = variableValue;
12     }
13     int getPublicVariable()
14     {
15         return publicVariable;
16     }
17 };
18
19 // Private access specifier
20 class PrivateClass
21 {
22 private:
23     int privateVariable;
24     void setPrivateVariable(int variableValue)
25     {
26         privateVariable = variableValue;
27     }
28     int getPrivateVariable()
29     {
30         return privateVariable;
31     }
32 };
33
34 int main()
35 {
36     PublicClass publicClassObject;
37     publicClassObject.setPublicVariable(5);
38     cout << publicClassObject.getPublicVariable() << endl;
39
40     PrivateClass privateClassObject;
41     // privateClassObject.setPrivateVariable(10); // Error: setPrivateVariable is private
42     // cout << privateClassObject.getPrivateVariable() << endl; // Error: getPrivateVariable is private
43     return 0;
44 }
```

Fig. 03 (Public and Private Access Specifiers)

### Output:



Microsoft Visual Studio Debug Console

5

Fig. 04 (Output for figure 3)

## Demonstration of Encapsulation and Abstraction using Getter/Setter functions

The demonstration of the encapsulation and abstraction using getter and setter functions is given in the code below:

```
1  #include<iostream>
2  using namespace std;
3
4  // Class Student
5  class Student
6  {
7  private:
8      string name;
9      int age;
10
11 public:
12     // Getter functions
13     string getName()
14     {
15         return name;
16     }
17     int getAge()
18     {
19         return age;
20     }
21
22     // Setter functions
23     void setName(string x)
24     {
25         name = x;
26     }
27     void setAge(int y)
28     {
29         age = y;
30     }
31 };
32
33 int main()
34 {
35     Student student1;
36     student1.setName("John");
37     student1.setAge(15);
38
39     cout << "Name of student1: " << student1.getName() << endl;
40     cout << "Age of student1: " << student1.getAge() << endl;
41
42     return 0;
43 }
```

Fig. 05 (Demonstration of Encapsulation and Abstraction using Getter/Setter functions)

### Output:

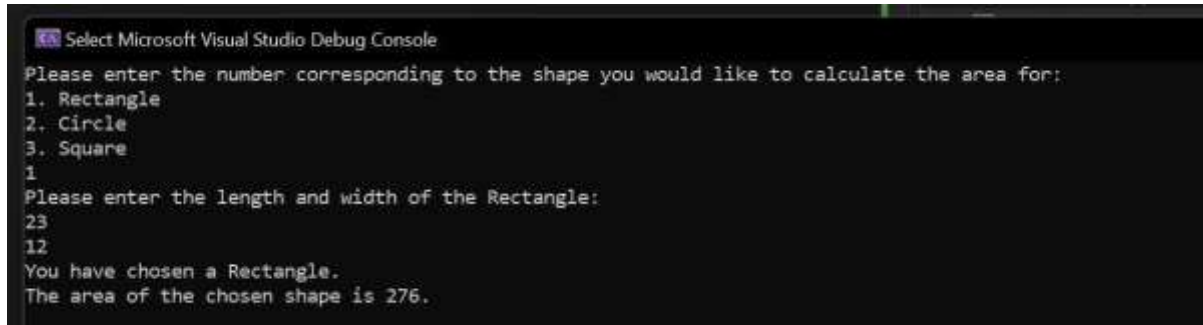
```
Microsoft Visual Studio Debug Console
Name of student1: John
Age of student1: 15
```

Fig. 06 (Output for figure 5)



**Task 01: Creating A Program with Classes and Objects** [Estimated 20 minutes / 20 marks]

Write a c++ program that uses classes and objects to calculate the area of a rectangle, circle, or square based on the user's choice and the dimensions they provide.

**Expected Output:**

```
Select Microsoft Visual Studio Debug Console
Please enter the number corresponding to the shape you would like to calculate the area for:
1. Rectangle
2. Circle
3. Square
1
Please enter the length and width of the Rectangle:
23
12
You have chosen a Rectangle.
The area of the chosen shape is 276.
```

Fig. 07 (Expected Output for Task 1)

**In-Lab Activities:****Constructors:**

Constructors in C++ are special class functions that are used to initialize objects. They are automatically called when an object is created. There are three types of constructors in C++:

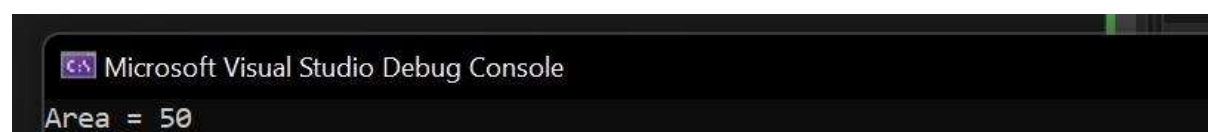
- **Default Constructor:**

The default constructor is a constructor that takes no parameters, or default values may be provided. It is automatically called when an object is created.

**Example Code:**

```
1  #include <iostream>
2
3  class Rectangle {
4      int width, height;
5  public:
6      Rectangle() { // default constructor
7          width = 10;
8          height = 5;
9      }
10     int area() {
11         return width * height;
12     }
13 };
14
15 int main() {
16     Rectangle rect;
17     int area = rect.area();
18     std::cout << "Area = " << area << std::endl;
19     return 0;
20 }
```

Fig. 08 (Default Constructor)

**Output:**

Microsoft Visual Studio Debug Console

Area = 50

Fig. 09 (Output with respect to figure 8)

- **Parameterized Constructor:**

The parameterized constructor is a constructor that takes parameters and initializes the object with the given values.

**Example Code:**

```
1  #include <iostream>
2
3  using namespace std;
4
5  class Student
6  {
7  private:
8      int age;
9      int id;
10
11  public:
12      // Parameterized Constructor
13      Student(int a, int b)
14      {
15          age = a;
16          id = b;
17      }
18
19      void showDetails()
20      {
21          cout << "Age = " << age << endl;
22          cout << "Id = " << id << endl;
23      }
24  };
25
26  int main()
27  {
28      // Creating an object using Parameterized Constructor
29      Student s1(18, 1234);
30      s1.showDetails();
31      return 0;
32  }
```

Fig. 10 (Parameterized Constructor)

**Output:**

```
Microsoft Visual Studio Debug Console
Age = 18
Id = 1234
```

Fig. 11 (Output with respect to figure 10)

- **Copy Constructor:**

The copy constructor is a constructor that creates a new object by copying the values of an existing object. It is used to create a new object based on an existing object. The details are out of scope for this laboratory manual.

**Overloaded Constructors:**

Overloaded constructors in C++ provide developers with the ability to create multiple constructors with different parameters and different implementations. This makes it easier to create objects with different initial states.

**Example:**

For example, consider a Car class with the following two constructors:

```
Car()
```

```
Car(int numberOfDoors)
```

The first constructor is the default constructor which initializes the car with the default values. The second constructor takes a parameter and initializes the car with the provided number of doors. This allows a developer to create a car with the default values or one with the number of doors specified as a parameter.

Another use of overloaded constructors is to create objects with different numbers of parameters. This allows for more flexibility when creating objects. For example, a Car class could have the following three constructors:

```
Car()
```

```
Car(int numberOfDoors)
```

```
Car(int numberOfDoors, string color)
```

With these three constructors, developers can create cars with the default values, with a specified number of doors, or with a specified number of doors and color.

Overall, overloaded constructors in C++ provide developers with the ability to create multiple constructors with different parameters and different implementations. This makes it easier to create objects with different initial states, and can be a powerful tool when creating classes.

**Example Code:**

```
1  #include <iostream>
2
3  using namespace std;
4
5  //Defining class
6  class Rectangle
7  {
8      int length;
9      int breadth;
10
11  public:
12      //Default Constructor
13      Rectangle()
14      {
15          length = 0;
16          breadth = 0;
17          cout << "Default Constructor called" << endl;
18          cout << "Length: " << length << endl;
19          cout << "Breadth: " << breadth << endl;
20      }
21
22      //Overloaded Constructor
23      Rectangle(int l, int b)
24      {
25          length = l;
26          breadth = b;
27          cout << "Overloaded Constructor called" << endl;
28          cout << "Length: " << length << endl;
29          cout << "Breadth: " << breadth << endl;
30      }
31  };
32
33  int main()
34  {
35      //Default Constructor called
36      Rectangle r1;
37      cout << endl;
38
39      //Overloaded Constructor called
40      Rectangle r2(10, 5);
41
42      return 0;
43  }
```

Fig. 11 (Overloaded Constructor)

**Output:**

```
Microsoft Visual Studio Debug Console
Default Constructor called
Length: 0
Breadth: 0

Overloaded Constructor called
Length: 10
Breadth: 5
```

Fig. 12 (Output with respect to figure 11)

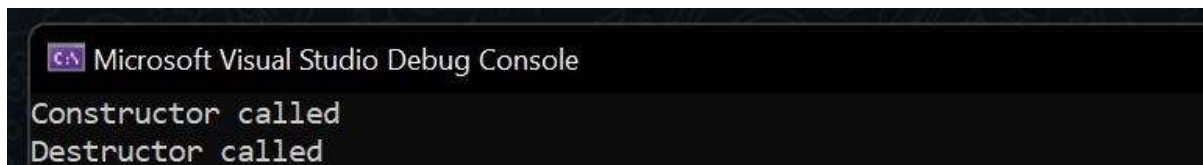
**Destructor:**

A destructor is a special member function of a class that is used to release the memory held by an object of that class, just before it is destroyed. This usually happens when the object goes out of scope, or when the delete operator is used to explicitly delete the object. The destructor has the same name as the class, preceded by a tilde (~). The destructor has no return type and no parameters. Because it is not called explicitly, it is important for the destructor to be declared in the public section of the class.

**Example Code:**

```
1  #include <iostream>
2
3  using namespace std;
4
5  class A
6  {
7  public:
8      A() {
9          cout << "Constructor called" << endl;
10     }
11     ~A() {
12         cout << "Destructor called" << endl;
13     }
14 };
15
16 int main()
17 {
18     A a; //Constructor called
19     return 0;
20 }
```

Fig. 13 (Destructor)

**Output:**

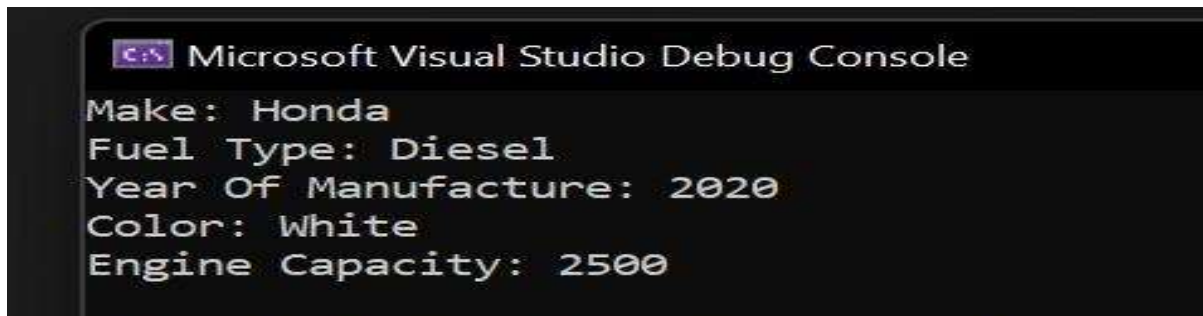
```
Microsoft Visual Studio Debug Console
Constructor called
Destructor called
```

Fig. 14 (Output with respect to figure 13)

When the object `a` goes out of scope, the destructor `~A()` is automatically called and the statement "Destructor called" is printed.

**Task 01: Class Definition and Implementation****[Estimated 15 minutes / 10 marks]**

Create a class called `MotorVehicle` that represents a motor vehicle using: `make` (type string), `fuelType` (type string), `yearOfManufacture` (type int), `color` (type string) and `engineCapacity` (type int). Your class should have a constructor that initializes the three data members. Provide a set and a get function for each data member. Add a member function called `displayCarDetails` that displays the five data members in five separate lines in the form "member name: member value". Write a test program to demonstrate `MotorVehicle`'s capabilities, that takes required values as input from the user.

**Output:**

```
Microsoft Visual Studio Debug Console
Make: Honda
Fuel Type: Diesel
Year Of Manufacture: 2020
Color: White
Engine Capacity: 2500
```

Fig. 15 (Required Output of Task 1)

**Task 2: Constructor Implementation****[Estimated 20 minutes / 20 marks]**

Assume you have to write a program for a cricket game. There are two teams and each team has a value for score and number of wickets. You are required to design a class `Team` that has relevant member functions for setting the score and wicket's value and for getting these values too.

(a). The score and wicket of both the teams should be taken as an input from the user in `main()`. The main function should interact with the class `Team` to set and get values. Write a function in the class `Team` to update the score and wickets of a particular team.

**Expected output (console):**

```
****Enter data for Team1****
Enter score: 24
Enter the number of wickets: 4
Team 1 has score: 24 and number of wickets: 4
****Enter data for Team2****
Enter score: 67
Enter the number of wickets: 9
Team 1 has score: 67 and number of wickets: 9
```

**Main function (driver function) should do the following tasks:**

```
//declare objects to interact with the class
//take input from user
//call set methods of class
//call getter methods and print to the console
```

(b). Declare two more objects and initialize one of the objects using the default constructor and the other using a parameterized constructor. You may send hardcoded values while calling the parametrized constructor.

**Expected output (console):**

```
**** Team3 created (default)****
Team 3 has score: 0 and number of wickets: 10
**** Team4 created (parameterized)****
Team 4 has a score: 11 and number of wickets: 2
```

**Main function (driver function) should do the following tasks:**

//declare objects to interact with the class

//call getter methods and print to the console

(c). In **main()**, write a logic that finds the team with 1) minimum score, 2) maximum score, and 3) greater than 6 wickets. The main function should interact with the class **Team** to set and get values. First, display the scores and wickets for all teams and then report 1,2, and 3.

**Expected output (console):**

```
Team 1 has score: 24 and the number of wickets: 4
Team 2 has score: 12 and the number of wickets: 9
Team 3 has score: 120 and the number of wickets: 3
Team 4 has score: 11 and number of wickets: 7
```

```
Team 3 has maximum score
Team 4 has minimum score
Team 2 and 4 have greater than 6 wickets
```

**Main function (driver function) should do the following tasks:**

//continuing the existing program...

//display values for teams using getter methods

//find maximum etc..as asked..

**Task 3: Constructor Implementation****[Estimated 40 minutes / 30 marks]**

Design a class of Circle with at least two data members: radius, originx, and originy. Origin is a central point of a circle and radius is the distance between the central point and the boundary of a circle. Since the origin is a point therefore it can be modeled with x and y coordinates.

(a). Write a program that finds the area, diameter, and circumference of a circle (create one sample object and find the following) by creating an appropriate class. The area, circumference, and diameter can be calculated using the following formulas, respectively:

$$\text{Area} = \text{pie} * \text{radius} * \text{radius}$$

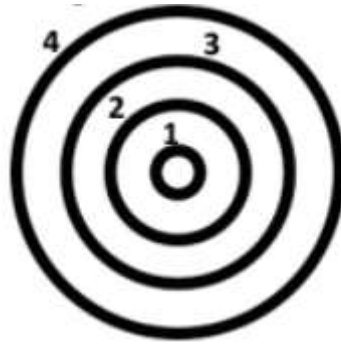
$$\text{Circumference} = 2 * \text{pie} * \text{radius}$$

$$\text{Diameter} = \text{radius} / 2$$

All the inputs should be taken from the user in the main function to set data members of the class (like done in Task 1) and then relevant member functions of the class should be called get values and find area, circumference, diameter. The functions for finding them must be a part of the class. Main function should only call relevant methods.

(b) Create 4 circles and initialize them using the parameterized constructor (you can hardcode values while calling the constructor). Find and display which of the circles are concentric. That is, circles having the same central point (as shown below). (Hint: the logic for this will be implemented in main())





(c) For the identified concentric circles, find the label for each circle. The label is the position number (as shown above; the most internal circle has label 1, and so on... In the end, also display the labels that you assigned. (Hint: the logic for this will be implemented in main()))

(d) Find and display: which of the concentric circle has a diameter greater than 12. You must differentiate the relationship between the diameter and radius of any circle. (Hint: the logic for this will be implemented in main())

#### Output:

```

Microsoft Visual Studio Debug Console
Enter the radius and originx, originy of the circle: 4
5
6

Area: 50.2654
Circumference: 25.1327
Diameter: 8

The concentric circles are:
Circles c2 and c3
Circles c3 and c4
Circles c4 and c5

The labels for the concentric circles are:
Circles c2 and c3: 1 and 2
Circles c3 and c4: 2 and 3
Circles c4 and c5: 3 and 4

The circle with diameter greater than 12 is:
  
```

Fig. 16 (Expected output of Task 3)

#### Task 4: Constructor Implementation

[Estimated 40 minutes / 30 marks]

Consider a class named Job that has a deadline as a data member and relevant getter/setter method(s). Assume you have to schedule two earliest jobs on the basis of their deadlines. That is, if there are three jobs in the system with deadlines (deadline1, deadline2, and deadline3, respectively) then the system should report the top two earliest jobs (with the smallest *deadline* value). You might need to find the *deadline* with the smallest and second smallest value.

**Expected output (console):**

```
Job 1 has deadline 12
Job 2 has deadline 1
Job 3 has deadline 11
Job 4 has deadline 45
Job 5 has deadline 6
Job 6 has deadline 9
Job 7 has deadline 66
Job 8 has deadline 5
Job 9 has deadline 27
Job 10 has deadline 34
Job 2 has earliest and job 8 has second most earliest deadline.
```

You need to do following tasks in main()

```
//Create 10 objects of class and initialize them with hardcoded values using parameterized constructor.
//Display the jobs with deadlines using getter method
//Find and report the two most earliest jobs
```

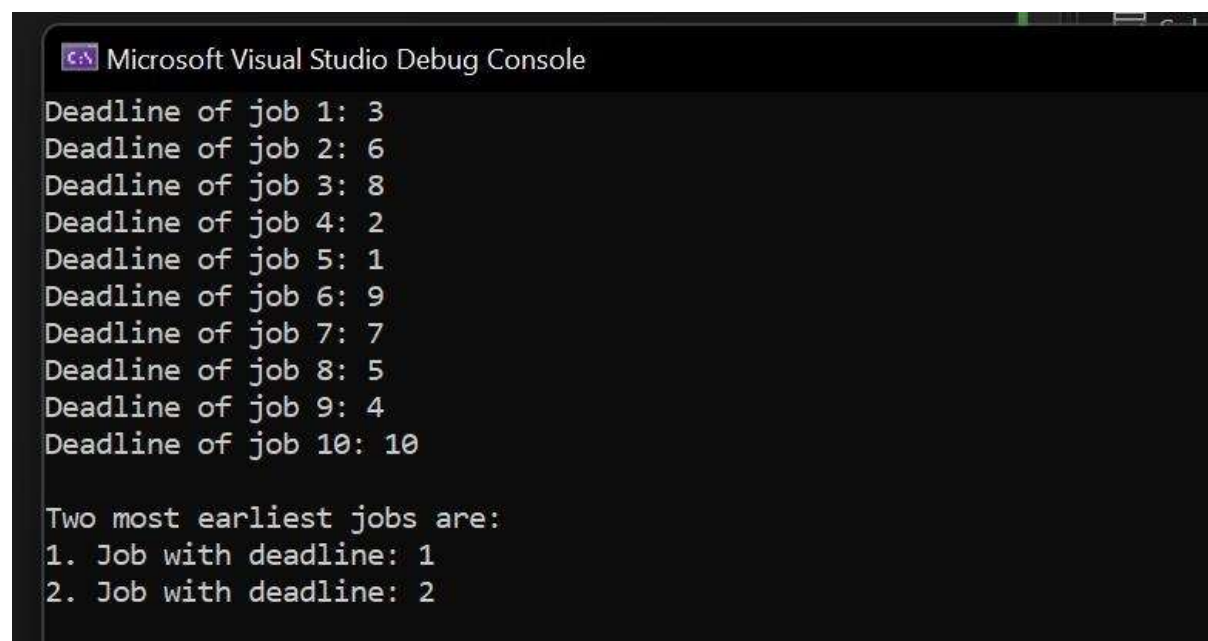
**Output:**The image is a screenshot of the Microsoft Visual Studio Debug Console. It displays the output of a program. The first part of the output lists the deadlines for 10 jobs, with each line formatted as 'Deadline of job X: Y'. The deadlines are: job 1: 3, job 2: 6, job 3: 8, job 4: 2, job 5: 1, job 6: 9, job 7: 7, job 8: 5, job 9: 4, and job 10: 10. The second part of the output states 'Two most earliest jobs are:' followed by a numbered list: '1. Job with deadline: 1' and '2. Job with deadline: 2'. The console window has a dark background and a title bar that reads 'Microsoft Visual Studio Debug Console'.

Fig. 17 (Expected output of Task 4)

**Post-Lab Activities:****Task 01: String Formatting****[Estimated 40 minutes / 30 marks]****Part (a):**

Define a class called `StringFormatter`. The purpose of an object of this class is to store a string variable (you may use the C++ string type or a char array). An object of this class can be created by calling a constructor that accepts a one-string argument. The string to be passed as argument will be a long line of text, such as

“The world is indeed full of peril and in it, there are many dark places. But still, there is much that is fair. And though in all lands, love is now mingled with grief, it still grows, perhaps, the greater.”

The object will also have a function called `printRightAligned()` which accepts one integer argument `n`. The value of the argument represents the maximum number of characters that can be displayed on a line. This function displays the string stored in the object's attribute on the screen, right aligned and with no more than `n` characters per line. Similarly, there should be a function called `printLeftAligned()` which displays the text left aligned, again, with no more than `n` characters per line.

For `printRightAligned()`, you can assume that the display is 20 characters wide, i.e, if a line to be displayed contains 15 characters, you would display 5 spaces followed by the text.

Note: You are not allowed to use `iomanip` or any other library except `string`.

**Expected input/ output:**

Input string:

“The world is indeed full of peril and in it there are many dark places. But still there is much that is fair. And though in all lands, love is now mingled with grief, it still grows, perhaps, the greater.”

For instance, if the `printRightAligned()` function is called with an argument value of 20, the following would be displayed on the screen:

The world is indeed  
full of peril and in  
and in it there are  
many dark places.  
But still there is  
much that is fair.  
And though in all  
lands, love is now  
mingled with grief,  
it still grows,  
perhaps, the greater.

Similarly, if the *printLeftAligned()* function were called with an argument value of 20, it would display the following text:

The world is indeed  
full of peril and in  
and in it there are  
many dark places.  
But still there is  
much that is fair.  
And though in all  
lands, love is now  
mingled with grief,  
it still grows,  
perhaps, the greater.

**Submissions:**

- For In-Lab Activity:
  - Save the files on your PC.
  - TA's will evaluate the tasks offline.
- For Pre-Lab & Post-Lab Activity:
  - Submit the .cpp file on Google Classroom and name it to your roll no.

**Evaluations Metric:**

- All the lab tasks will be evaluated offline by TA's
- **Division of Pre-Lab marks:** [40 marks]
  - Task 01: Class and Object Basics [20 marks]
  - Task 02: Pointers & Addresses [20 marks]
- **Division of In-Lab marks:** [80 marks]
  - Task 01: Class Definition and Implementation [15 marks]
  - Task 02: Constructor Implementation [20 marks]
  - Task 03: Constructor Implementation [30 marks]
  - Task 04: Constructor Implementation [30 marks]
- **Division of Post-Lab marks:** [30 marks]
  - Task 01: String Formatting [30 marks]

**References and Additional Material:**

- Classes and Objects in C++  
<https://www.programiz.com/cpp-programming/object-class>
- Constructors  
<https://www.programiz.com/cpp-programming/constructors>
- Objects and Functions  
<https://www.programiz.com/cpp-programming/pass-return-object-function>
- Operator Overloading  
<https://www.programiz.com/cpp-programming/operator-overloading>

**Lab Time Activity Simulation Log:**

- Slot – 01 – 00:00 – 00:15: Class Settlement
- Slot – 02 – 00:15 – 00:30: In-Lab Task
- Slot – 03 – 00:30 – 00:45: In-Lab Task
- Slot – 04 – 00:45 – 01:00: In-Lab Task
- Slot – 05 – 01:00 – 01:15: In-Lab Task
- Slot – 06 – 01:15 – 01:30: In-Lab Task
- Slot – 07 – 01:30 – 01:45: In-Lab Task
- Slot – 08 – 01:45 – 02:00: In-Lab Task
- Slot – 09 – 02:00 – 02:15: In-Lab Task
- Slot – 10 – 02:15 – 02:30: In-Lab Task
- Slot – 11 – 02:30 – 02:45: Evaluation of Lab Tasks
- Slot – 12 – 02:45 – 03:00: Discussion on Post-Lab Task