

**CC-211L**

**Object Oriented Programming**

**Laboratory 08**

**Stream I/O**

**Version: 1.0.0**

**Release Date: 10-03-2023**

**Department of Information Technology**

**University of the Punjab**

**Lahore, Pakistan**

**Contents:**

- Learning Objectives
- Required Resources
- General Instructions
- Background and Overview
  - C++ Stream
  - Type Conversion
- Activities
  - Pre-Lab Activity
    - Streams
    - iostream Library Headers
    - Standard Stream Objects cin, cout
    - Task 01
    - Task 02
  - In-Lab Activity
    - Stream Output
    - Stream Input
    - Scan and display a single character
    - Display ncount
    - istream\_withassign class
    - ostream\_withassign class
    - Overloading Cast Operator
    - Task 01
    - Task 02
    - Task 03
- Submissions
- Evaluations Metric
- References and Additional Material
- Lab Time and Activity Simulation Log

**Learning Objectives:**

- Streams
- Stream Input
- Stream Output
- Overloading Cast Operator

**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>
Lab Instructor	Azka Saddiqa	<a href="mailto:azka.saddiqa@pucit.edu.pk">azka.saddiqa@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:**

### **C++ Stream:**

Stream in C++ means a stream of characters that gets transferred between the program thread and input or output. There are a number of C++ stream classes eligible and defined which is related to the files and streams for providing input-output operations. All the classes and structures maintaining the file and folders with hierarchies are defined within the file with iostream.h standard library. Classes associated with the C++ stream include ios class, istream class, and ostream class.

### **Type Conversion:**

Type conversion is the process that converts the predefined data type of one variable into an appropriate data type. The main idea behind type conversion is to convert two different data type variables into a single data type to solve mathematical and logical expressions easily without any data loss.

## Activities:

### Pre-Lab Activities:

#### Streams:

C++ I/O occurs in streams, which are sequences of bytes. In input operations, the bytes flow from a device to main memory. In output operations, bytes flow from main memory to a device.

#### **iostream Library Headers:**

The C++ stream libraries provide hundreds of I/O capabilities. Most of our C++ programs include the header, which declares basic services required for all stream I/O operations. The header defines the `cin`, `cout`, `cerr` and `clog` objects, which correspond to the standard input stream, the standard output stream, the unbuffered standard error stream, and the buffered standard error stream. Both unformatted and formatted-I/O services are provided.

The `iostream` library provides many class templates for performing common I/O operations.

- **istream** for stream input operations
- **ostream** for stream output operations
- **iostream** for both stream input and output operations

#### **Standard Stream Objects `cin`, `cout`:**

Predefined object `cin` is an `istream` object and is said to be “connected to” (or attached to) the standard input device, which usually is the keyboard. The stream extraction operator (`>>`) as used in

**`int marks;`**

**`cin >> marks;`**

causes a value for `int` variable `marks` to be input from `cin` to memory. The compiler selects the appropriate overloaded stream extraction operator, based on the type of the variable `marks`. The `>>` operator is overloaded to input data items of fundamental types, strings, and pointer values.

The predefined object `cout` is an `ostream` object and is said to be “connected to” the standard output device, which usually is the display screen. The stream insertion operator (`<<`), as used in the following statement, causes the value of variable `marks` to be output from memory to standard output device:

**`cout << marks;`**

**Task 01: Student Marks****[Estimated time 15 minutes / 10 marks]**

- Create a class "Student" with the following data members:
  - Subjects (int)
  - Marks (int [])
- Implement a copy constructor for the "Rectangle" class.
- Implement getter/setter methods
- Implement a method 'inputMarks' which takes no of subjects and marks as input from user
- Implement a method 'outputMarks' which displays the marks of student

**Task 02: Person Class****[Estimated time 30 minutes / 20 marks]**

Create a class Person with the following private member variables:

- name (string): the name of the person
- age (int): the age of the person
- gender (char): the gender of the person (either 'M' for male or 'F' for female)

Implement the following public member functions:

- A default constructor that sets the name to an empty string, the age to 0, and the gender to 'M'.
- A parameterized constructor that sets the name, age, and gender to the values passed as arguments.
- Write a function to read in a Person object from the standard input. The input should consist of three values: the name (a string), the age (an int), and the gender (a char). The input values should be separated by whitespace. For example, if the input is "John 25 M", the Person object should have name "John", age 25, and gender 'M'.
- Write a function to write a Person object to the standard output in the following format: "Name: [name], Age: [age], Gender: [gender]". For example, if a Person object has name "John", age 25, and gender 'M', the output should be "Name: John, Age: 25, Gender: M".

Once you have implemented the Person class, write a main function that demonstrates how to use the implemented functions.

## In-Lab Activities:

### Stream Output:

It is responsible for handling output stream. It provides number of function for handling chars, strings and objects such as write, put etc.

The stream `std::cout` is connected to the computer's display, usually called console output. When a statement such as `std::cout << x;` is executed, the value of variable `x` is converted into a sequence of characters (corresponding to its text representation).

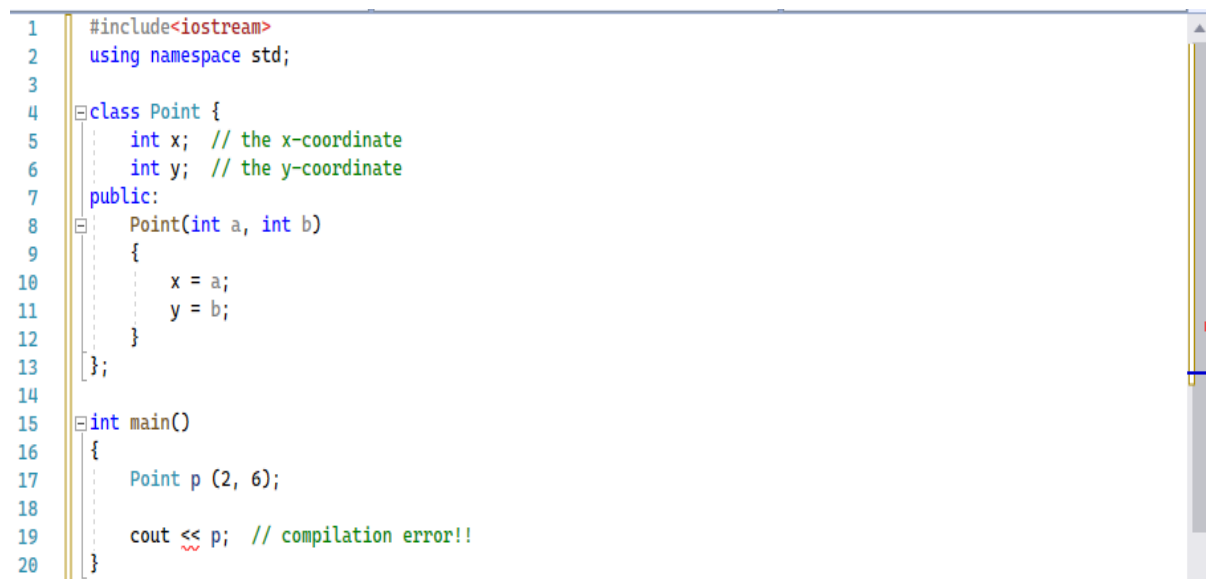
For instance, if variable `x` is a double storing 12.34712.347 then this value is converted to the sequence of characters `'1' '2' '.' '3' '4' '7'`. These characters are then inserted into the stream's buffer for transmission to the output device, i.e. the computer's display. The conversion of variable `x`'s value to text can be controlled by the programmer by using C++ manipulators, like in

```
std::cout <<
std::fixed <<
std::setprecision(2) << x;
```

Stream output operation `<<` is only available for

- the basic types of the language, such as `int`, `double`, `bool`, etc.
- some types defined in the C++-standard library like `std::string`

Thus, if you consider the type `Point` representing a point in a 2D-space then one cannot use operation `<<` to display directly the value of data members of type `Point`.



```
1  #include<iostream>
2  using namespace std;
3
4  class Point {
5      int x; // the x-coordinate
6      int y; // the y-coordinate
7  public:
8      Point(int a, int b)
9      {
10         x = a;
11         y = b;
12     }
13 };
14
15 int main()
16 {
17     Point p (2, 6);
18
19     cout << p; // compilation error!!
20 }
```

Fig. 01 (Stream Output)

Instead, one can define a function to display a point.

```

1  #include<iostream>
2  using namespace std;
3
4  class Point {
5      int x; // the x-coordinate
6      int y; // the y-coordinate
7  public:
8      Point(int a, int b)
9      {
10         x = a;
11         y = b;
12     }
13     void display_point(const Point& p) {
14         cout << "(" << p.x << ", " << p.y << ")";
15     }
16 };
17
18 int main()
19 {
20     Point p (2, 6);
21     p.display_point(p);
22 }

```

Fig. 02 (Stream Output)



Microsoft Visual Studio Debug Console

(2, 6)

Fig. 03 (Stream Output)

### Stream Input:

The stream `std::cin` is connected to the computer's keyboard, usually called console input. Any characters entered by a user through the keyboard are transferred automatically to the buffer of the stream. Consider the piece of code below:

```
double x = 0.0;
```

```
std::cin >> x;
```

When `std::cin >> x;` is executed, the following steps are performed.

- If the stream's buffer is empty, then the program waits until characters are transferred from the device (e.g., computer's keyboard) into the buffer. Otherwise, it proceeds with next step.
- The reading operator `>>` scans the characters in the stream's buffer and converts them automatically to the corresponding value of variable `x`'s type (i.e., a double). This value is then stored in variable `x`. This scanning process starts at the beginning of the buffer and, it stops when it encounters a character that cannot be converted or when it reaches the end of the buffer.
- The sequence of characters scanned and converted in the previous step are removed from the stream's buffer.

The stream input operation `>>` (technically called stream extraction operator) is only available for

- The basic types of the language, such as `int`, `double`, `bool`, etc.
- Some types defined in the C++-standard library like `std::string`.

Thus, one cannot use operation `>>` to read directly the value of data-members of a type defined by the programmer such as `Point`.



```

1  #include<iostream>
2  using namespace std;
3
4  class Point {
5      int x; // the x-coordinate
6      int y; // the y-coordinate
7  public:
8      Point()
9      {
10         x = 0;
11         y = 0;
12     }
13 };
14
15 int main()
16 {
17     Point p;
18     cin >> p;
19 }

```

Fig. 04 (Stream Input)

Instead, one can define a function to display a point.

```

1  #include<iostream>
2  using namespace std;
3
4  class Point {
5      int x; // the x-coordinate
6      int y; // the y-coordinate
7  public:
8      Point()
9      {
10         x = 0;
11         y = 0;
12     }
13     void read_point()
14     {
15         cout << "Enter x coordinate: "; cin >> x;
16         cout << "Enter y coordinate: "; cin >> y;
17     }
18     void display_point(const Point& p) {
19         cout << "(" << p.x << ", " << p.y << ")";
20     }
21 };
22
23 int main()
24 {
25     Point p;
26     p.read_point();
27     p.display_point(p);
28 }

```

Fig. 05 (Stream Input)

Microsoft Visual Studio Debug Console

```

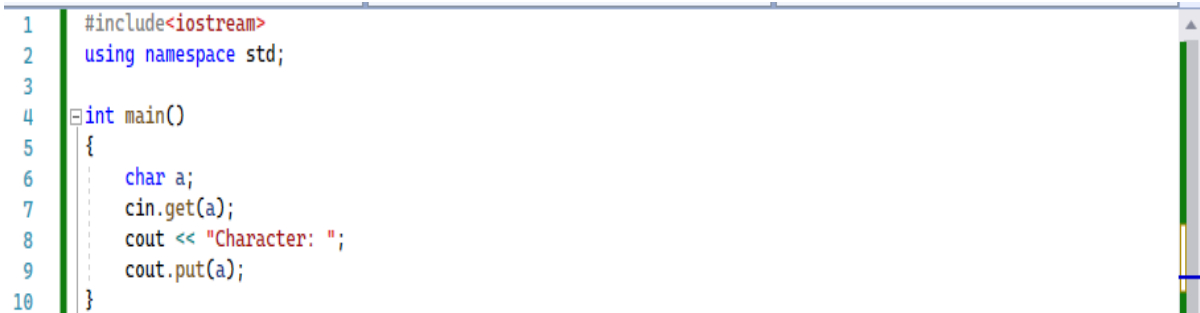
Enter x coordinate: 3
Enter y coordinate: 2
(3, 2)

```

Fig. 06 (Stream Input)

**Scan and display a single character:**

iostream uses a `get()` function to scan a character and a `put()` function to display it.

**Example:**

```
1  #include<iostream>
2  using namespace std;
3
4  int main()
5  {
6      char a;
7      cin.get(a);
8      cout << "Character: ";
9      cout.put(a);
10 }
```

Fig. 07 (Scan and display a single character)

**Output:**

Microsoft Visual Studio Debug Console

Y  
Character: Y

Fig. 08 (Scan and display a single character)

**Display ncount:**

iostream uses a `write()` function to display specified count of elements.

**Example:**

```
1  #include<iostream>
2  using namespace std;
3
4  int main()
5  {
6      cout.write("Enter a text", 3);
7  }
```

Fig. 09 (Display ncount)

**Output:**

Microsoft Visual Studio Debug Console

Ent

Fig. 10 (Display ncount)

**istream\_withassign class:**

This class is variant of `istream` that allows object assignment. The predefined object `cin` is an object of this class and thus may be reassigned at run time to a different `istream` object.

**Example:**

```

1  #include <iostream>
2  using namespace std;
3
4  class demo {
5  public:
6      int dx, dy;
7
8      // operator overloading using friend function
9      friend void operator>>(demo& d, istream& mycin)
10     {
11         // cin assigned to another object mycin
12         mycin >> d.dx >> d.dy;
13     }
14 };
15
16 int main()
17 {
18     demo d;
19     cout << "Enter two numbers dx and dy\n";
20
21     // calls operator >> function and
22     // pass d and cin as reference
23     d >> cin; // can also be written as operator >> (d, cin) ;
24
25     cout << "dx = " << d.dx << "\tdy = " << d.dy;
26 }

```

Fig. 11 (istream\_withassign class)

**Output:**


```

Microsoft Visual Studio Debug Console
Enter two numbers dx and dy
2
3
dx = 2   dy = 3

```

Fig. 12 (istream\_withassign class)

**ostream\_withassign class:**

This class is variant of ostream that allows object assignment. The predefined objects cout, cerr, clog are objects of this class and thus may be reassigned at run time to a different ostream object.

**Example:**

```

1  #include <iostream>
2  using namespace std;
3
4  class demo {
5  public:
6      int dx, dy;
7
8      demo()
9      {
10         dx = 4;
11         dy = 5;
12     }
13     // operator overloading using friend function
14     friend void operator<<(demo& d, ostream& mycout)
15     {
16         // cout assigned to another object mycout
17         mycout << "Value of dx and dy are \n";
18         mycout << d.dx << " " << d.dy;
19     }
20 };
21
22 int main()
23 {
24     demo d; // default constructor is called
25
26     // calls operator << function and
27     // pass d and cout as reference
28     d << cout; // can also be written as operator << (d, cout) ;
29 }

```

Fig. 13 (ostream\_withassign class)

**Output:**


```

Microsoft Visual Studio Debug Console
Value of dx and dy are
4 5

```

Fig. 14 (ostream\_withassign class)

**Overloading Cast Operator:**

A conversion operator (also called a cast operator) also can be used to convert an object of one class to another type. Such a conversion operator must be a non-static member function. The function prototype:

**MyClass::operator string() const;**

declares an overloaded cast operator function for converting an object of class MyClass into a temporary string object. The operator function is declared const because it does not modify the original object. The return type of an overloaded cast operator function is implicitly the type to which the object is being converted.

Overloaded cast operator functions can be defined to convert objects of user-defined types into fundamental types or into objects of other user-defined types.

**Example:**

```

3  class Power {
4      double b, val;
5      int e;
6  public:
7      Power(double base, int exp){
8          b = base, e = exp;
9          val = 1;
10         if (exp == 0)
11             return;
12         for (; exp > 0; exp--)
13             val = val * b;
14     }
15     Power operator+(Power o) {
16         double base;
17         int exp;
18         base = b + o.b;
19         exp = e + o.e;
20         Power temp(base, exp);
21         return temp;
22     }
23     operator double() { return val; } // convert to double
24 };
25 int main() {
26     Power x(4.0, 2);
27     double a;
28     a = x;           // convert to double
29     cout << x + 1.2 << "\n"; // convert x to double and add 100.2
30     Power y(3.3, 3), z(0, 0);
31     z = x + y; // no conversion
32     a = z;     // convert to double
33     cout << a;
34 }

```

Fig. 15 (Overloading Cast Operator)

**Output:**


```

Microsoft Visual Studio Debug Console
17.2
20730.7

```

Fig. 16 (Overloading Cast Operator)

**Task 01: Rectangle Tuple****[Estimated time 40 minutes / 30 marks]**

Create a C++ class named Rectangle that represents a rectangle with integer length and width. The class should have the following members:

- int length and int width as private data members.
- A default constructor that sets both length and width to 0.
- A constructor that takes two integer arguments to set length and width.
- Overloaded insertion operator << that writes a Rectangle object to a stream in the following format: (length, width).
- Overloaded extraction operator >> that reads a Rectangle object from a stream in the following format: (length, width).

Your program should perform the following steps:

- Define the Rectangle class as described above.
- Implement the << operator overload for the Rectangle class.
- Implement the >> operator overload for the Rectangle class.
- Write a main() function that demonstrates the use of the Rectangle class and its overloaded operators. In particular, your main() function should do the following:
  - Create two Rectangle objects, one with length 5 and width 3, and one with length 7 and width 2.
  - Output the two Rectangle objects to the console using the << operator.
  - Prompt the user to enter the length and width of a rectangle in the format (length, width).
  - Read in the user's input using the >> operator and create a Rectangle object with the specified dimensions.
  - Output the user's Rectangle object to the console using the << operator.
- Your program should handle the following error cases:
  - If the user enters invalid input when prompted to enter the dimensions of a rectangle (i.e., input that cannot be parsed as a valid (length, width) tuple), print an error message to the console and prompt the user to enter valid input.
- Also write a driver code to test the above functionalities

**Task 02: Write Characters****[Estimated time 30 minutes / 20 marks]**

Create a C++ program that reads a string, counts the number of occurrences of each character in the string, and displays the results on Console. Your program should use the put() functions to display individual characters.

Your program should perform the following steps:

- Prompt the user to enter a string.
- Use a loop to iterate over the string. For each entry, write the character and its count to the Console using the put() function.

Your program should handle the following error cases:

- Note that the string can contain any text, including whitespace and non-printable characters. The Console should print one line for each character that appears in the string, in ascending order of character code. Each line should be of the form c: n, where c is the character and n is the number of times it appears in the string.

**Task 03: Convert 2D to 1D****[Estimated time 40 minutes / 30 marks]**

- Write a C++ program that defines a Matrix class representing a 2D matrix of integers
- Overload the cast operator to convert a Matrix object to a 1D vector of integers

The program should then create a Matrix object, convert it to a vector, and output the result to the console

**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:** [30 marks]
  - Task 01 Student Marks [10 marks]
  - Task 02 Person Class [20 marks]
- **Division of In-Lab marks:** [80 marks]
  - Task 01: Rectangle Tuple [30 marks]
  - Task 02: Write Characters [20 marks]
  - Task 03: Convert 2D to 1D [30 marks]

**References and Additional Material:**

- **Overloading Cast Operator**  
<https://www.learncpp.com/cpp-tutorial/overloading-typecasts/>
- **C++ Stream**  
<https://www.geeksforgeeks.org/c-stream-classes-structure/>



**Lab Time Activity Simulation Log:**

- Slot – 01 – 00:00 – 00:15: Class Settlement
- Slot – 02 – 00:15 – 00:40: In-Lab Task
- Slot – 03 – 00:40 – 01:20: In-Lab Task
- Slot – 04 – 01:20 – 02:20: In-Lab Task
- Slot – 05 – 02:20 – 02:45: Evaluation of Lab Tasks
- Slot – 06 – 02:45 – 03:00: Discussion on Post-Lab Task