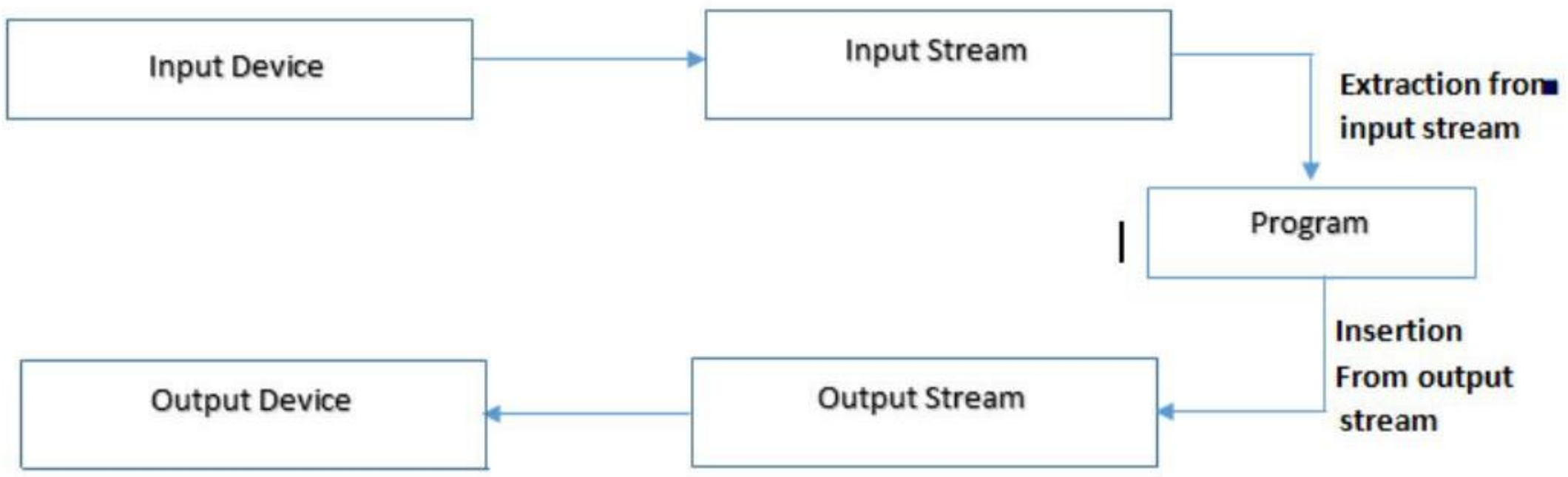
Unit-6 Managing Console I/O Operations

1 Explain Input stream and Output stream in brief

- The I/O system supplies an interface to the programmer that is independent of the actual device being accessed. This interface is known as ***stream.***
- stream is a sequence of bytes. It acts either a source from which the input data can be obtained or as a destination to which the output data can be sent.
- The source stream that provides data to the program is called the *input stream* and the destination stream that receives output from the program is called the *output stream.*
- other words, a program extracts the bytes from an input stream and inserts bytes into an output stream. Which is shown in the below figure:



- We know that cin represents the input stream connected to the standard input device and cout represents the output stream connected to the standard output device.

Header files available in C++ for Input – Output operation are:

- **iostream**: iostream stands for standard input output stream. This header file contains definitions to objects like cin, cout, cerr etc.
- **iomanip**: iomanip stands for input output manipulators. The methods declared in this files are used for manipulating streams. This file contains definitions of setw, setprecision etc.
- **fstream**: This header file mainly describes the file stream. This header file is used to handle the data being read from a file as input or data being written into the file as output.

Example :

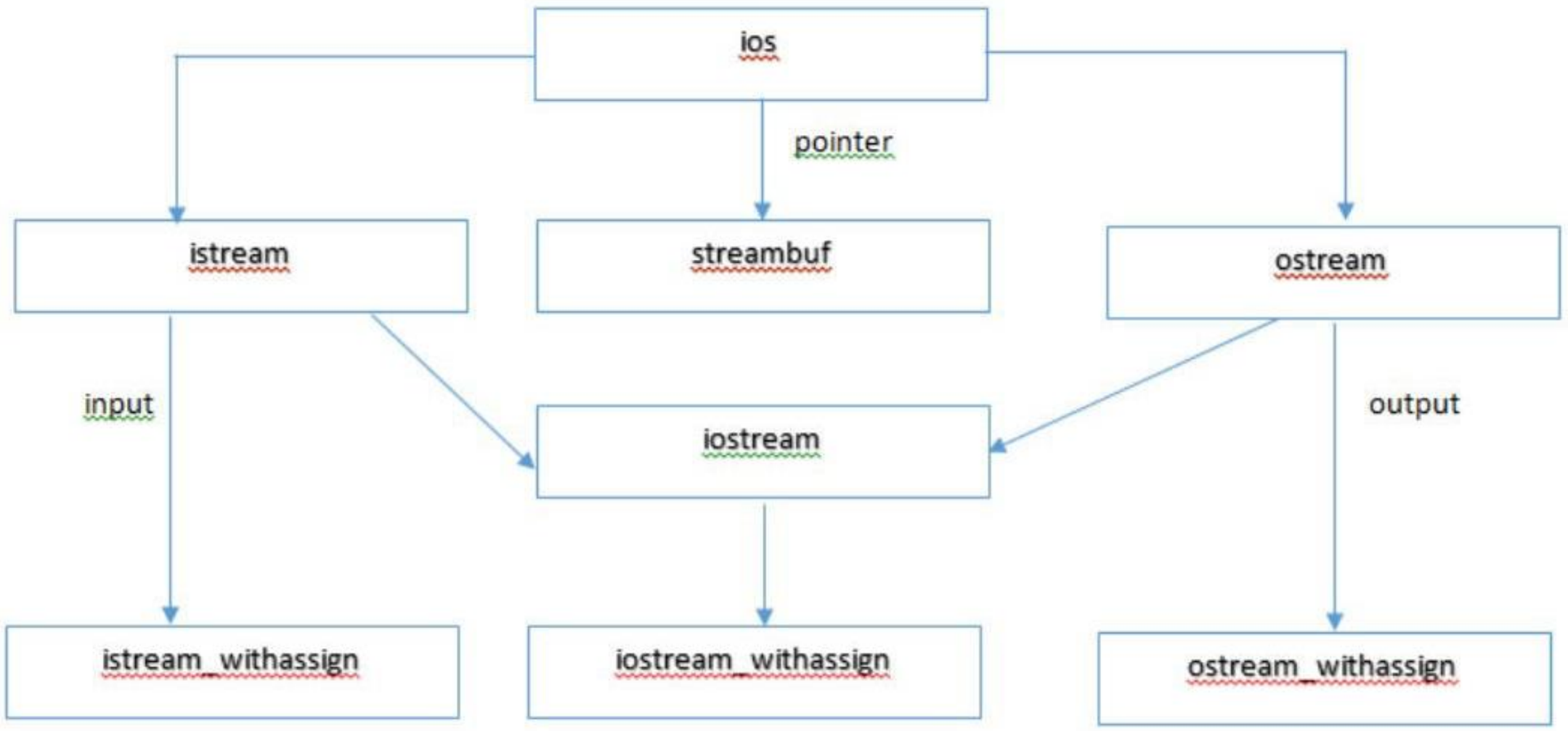
```
#include<iostream>
using namespace std;
int main()
{
   int age;
   cout << "Enter your age:";
   cin >> age;
   cout << "\nYour age is: "<<age;
   return 0;
}
```

Output :

```
Enter your age: 18
Your age is: 18
```

2 Explain C++ Stream class hierarchy.

- The C++ I/O system contains a hierarchy of classes that are used to define various streams to deal with both the console and disk files. These classes are called *stream classes.*
- Figure shows the hierarchy of the stream class's user for input and output operations with console unit. These classes are declared in the header file ***iostream***. This file should be included in all the programs that communicate with the console unit



- As shown in the figure, *ios* is the base class from *istream(input stream)* and *ostream(output stream)* which are, in turn, base classes for *iostream(input/output stream).* The class *ios* is declared as the virtual base class so that only one copy of its members are inherited by the iostream.
- The class *ios* provides the basic support for formatted and unformatted I/O operations. The class *istream* provides the facilities for formatted & unformatted input while the class *ostream* provides the facilities for formatted output.
- The class *iostream* provides the facilities for handling both input and output streams. The classes namely, *istream_withassign , ostream_withassign , iostream_withassign.*
- Following Table gives the detail of the classes.

Class Name	Contents
ios (General I/O Stream Class)	Contains basic facilities that are used by all other input and output classes.
istream (Input Stream)	Inherits the properties of ios, Declared input functions like get(),getline() and read(), Contains extraction operator >>
ostream (Output Stream)	Inherits the properties of ios, Declared output functions like put() and write(), Contains insertion operator <<
iostream (I/O Stream)	Inherits the properties of ios stream and ostream through multiple inheritance and this contains all the input and output functions.
streambuf	Provides an interface to physical devices through buffers

3 Explain Unformatted I/O Operations

Overloaded Operators >> and <<:

- We have used the objects cin and cout for the input and output of data of various types. This has been made possible by overloading the operators >> and << to recognize all the basic C++ types.

- The >> operator is overloaded in the istream class and << is overloaded in the ostream class.
- General form of these operators are as shown below;
- cin >> variable1 >> variable2>>.....>>variableN;
- variable1,variable2..... are valid C++ variable names that have been declared already. This statement will cause the computer to stop the execution and look for input data from the keyboard.

Example is shown below;

```
#include<iostream>
using namespace std;
int main()
{
    cout<<"This is the example of COUT"<<endl;
    int a;
    cout<<"Enter Value of a:";
    cin>>a;
    cout<<"Entered Value of a is: "<<a;
    return 0;
}
```

put() and get() Functions:

- The classes istream and ostream define two member functions get() and put() respectively to handle the single character input/output operations.
- get() functions, We can use get(char *) prototype to fetch a character including the blank space, tab and the newline character.
- The get(char *) version assigns the input character to its argument.
- The function put() a member of ostream class, can be used to output a line of text, character by character.

Example is shown below:

```
#include<iostream>
#include<conio.h>
using namespace std;
int main()
{
        char c1;
        cout<<"First Method: Using get(* char)"<<endl;
        cin.get(c1);
        cout.put(c1);
        return 0;
}
```

getline() and write() Function:

- We can read and display a line of text more efficiently using the line-oriented input/output functions getline() and write().
- The getline() function reads a whole line of text that ends with a newline character. This function can be invoked by using the object cin.
- This function call invokes the function getline() which reads character input into the variable line. The reading is terminated as soon as either the new line character '\n' is encountered. The newline character is read but not saved.
- getline() function is advantage over cin because cin can read strings that do not contain white spaces.
- The write function has the following format;

cout.write(line,size);

- The first argument line represents the name of the string to be displayed and the second argument size indicates the number of character to display.
- Note that it does not stop displaying the characters automatically when the null character is encountered.
- Example is shown below;

```
#include<iostream>
#include<conio.h>
 using namespace std;
 int main()
{
 cout<<"Example of write() and getline() Function"<<endl;
   char city[40];
    cout<<"Enter City and State here:"<<endl;
cin.getline(city,40);
cout.write(city,18);
return 0;
  }
```

4 Explain Formatted I/O Operations

- The ios class contains a large number of member functions that would help us to format the output in a number of ways.
- Following table shows the regarding information;

Sr. No.	Function	Function Syntax	Syntax Explanation	Function Description
1.	width()	cout.width(w);	Here w = filed width	To specify the required filed size for displaying on output value.
2.	precision()	cout.precision(d);	Here d is the number of digit to the right of the decimal point.	The floating numbers are printed 6 digits after the decimal point, we can specify the no. of digit with this function.
3.	fill()	cout.fill(ch);	Here ch represents the character which is used for filling unused positions.	The unused positions of the field are filled with white spaces, by default, However, we can use the fill() function to fill the unused position by any desired character.
4.	setf()	cout.setf(arg1,arg2);	arg1= formatting flags defined in ios arg2= formatting flags defined in ios	The setf() function can be specify format flags that can control the form of output display (Such as left-justified, right-justified)
5.	unsetf()	--	--	To clear the flags specified

Example is shown below;

```
#include<iostream>
#include<conio.h>
using namespace std;
int main()
{
        cout.width(5);
        cout<< 123 <<endl;
        cout.precision(5);
        cout<< 123.4567890 << "\n";
        cout.fill('*');
        cout.width(10);
        cout<< 31190 << "\n";
        cout.setf(ios::left,ios::adjustfield);
        cout.fill('*');
        cout.width(20);
        cout<<"Programming in C++";
        getch();
        return 0;
}
```

5 Explain Formatting with Manipulators

- The header file *iomanip* provides a set of functions called manipulators which can be used to manipulate the output formats.
- To access these manipulators, we must include the file *iomanip* in the file program.
- Following table shows the manipulator and their meaning;

Manipulator	Meaning
setw(int w)	Set the filed width to w.
setprecision(int d)	Set the floating point precision to d.
setfill(int c)	Set the fill character to c.
setiosflags(long f)	Set the format flag f.
resetiosflags(long f)	Clear the flag specified by f
Endl	Insert new line and flush stream.

Example is as shown below :

```
#include<iostream>
#include<conio.h>
#include<iomanip>
using namespace std;
int main()
    {
        cout<<setw(5)<<setprecision(2)<<123.456<<"\n";
        cout<<setiosflags(ios::left)<<345.678<<"\n";
        cout<<setw(10)<<setfill('*')<< 123.45<<"\n";
        return 0;
}
```

We can design our own manipulator also, The general form for creating a manipulator without any argument is;

```
ostream & manipulator (ostream &  output)
{
      ……
      ……
      ……
      return output;
}
```

Here manipulator is the name of the manipulator under creation. The following function defines a manipulator called unit that displays "*inches*"

ostream & unit(ostream & output)

output<< "inches ";

return output;