Managing Console I/O Operations

Sisoft Technologies Pvt Ltd
SRC E7, Shipra Riviera Bazar, Gyan Khand-3, Indirapuram, Ghaziabad
Website: www.sisoft.in Email:info@sisoft.in
Phone: +91-9999-283-283

- C++ uses the concept of stream & stream classes to implement its I/O Operations with the console and disk files.

WHAT IS STREAM:

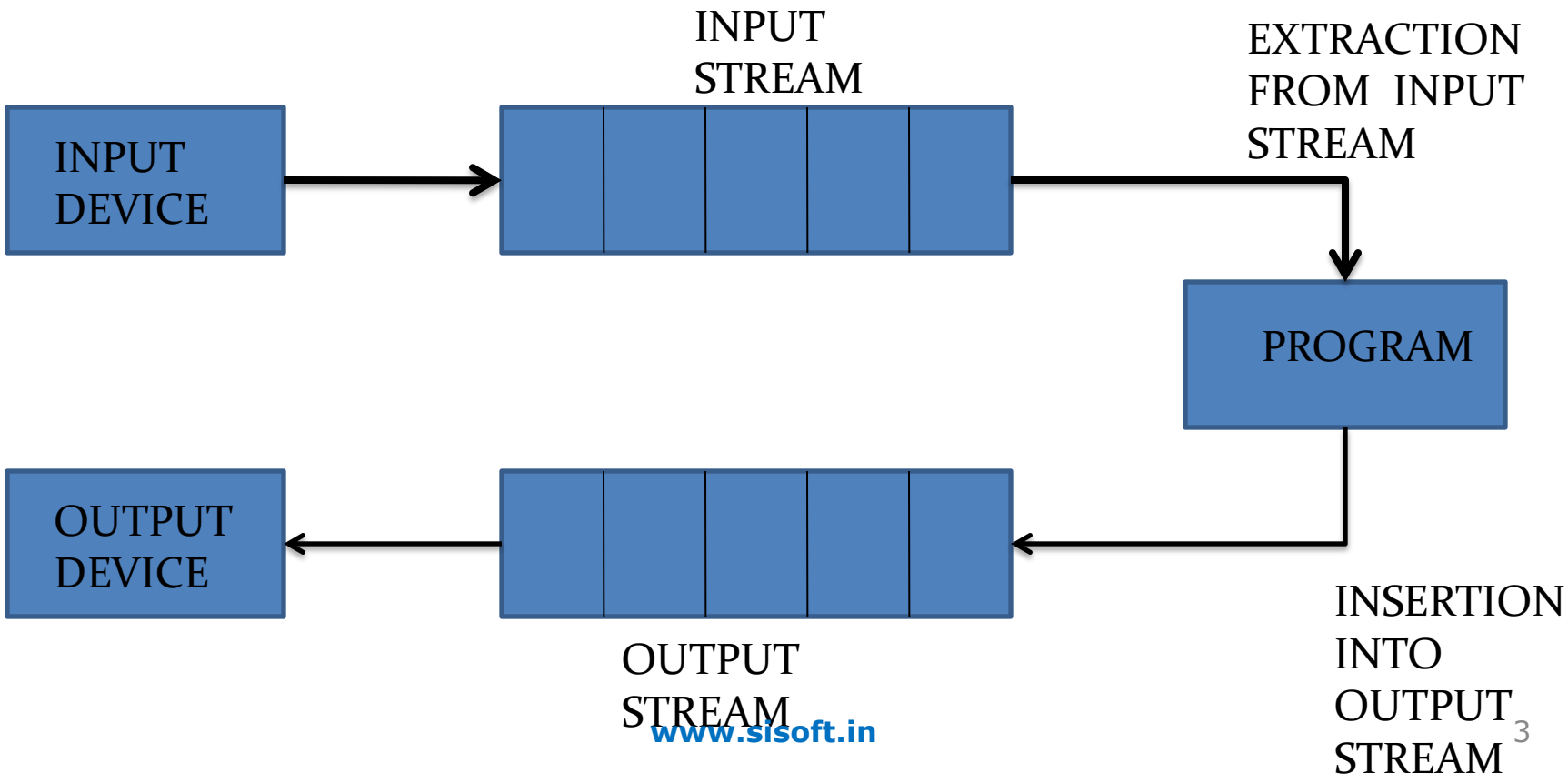
A stream is sequence of bytes. It is an interface between program and device.

It acts either as 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 Input Stream and the designation stream that receives output from the program is called Output Stream.

In other words, a program extracts the bytes from an input stream & inserts bytes into an output stream

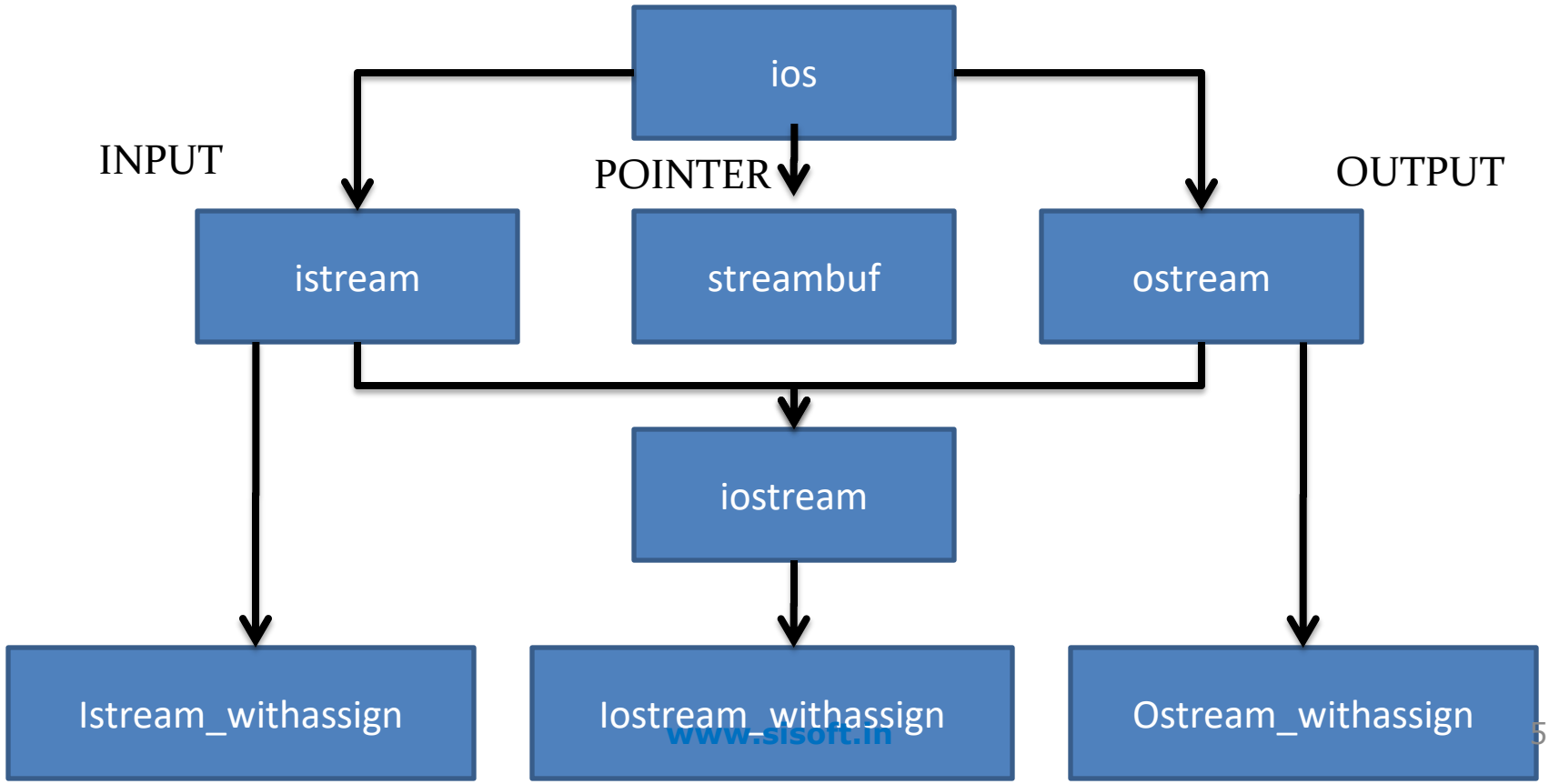
This diagram shows that the data can come from the keyboard or any other storage device. Similarly the data in the output stream can go to the screen or any other storage device.



C++ Stream Classes

C++ contains a hierarchy of classes that are used to define various streams to deal with the both console and disk files. These classes are called stream classes.

See the below Diagram:



Here Diagram shows that :

1) ios is the base class for istream (input stream) and ostream (output stream) which are again base classes for iostream(input/output stream).

2) The class ios declared as the virtual base class so that only one copy of its member are inherited by the iostream.

3) The class ios provides the basic support for formatted and unformatted I/O operations.

4) The class iostream provides the facilities for handling both input and output streams.

5) Three classes, namely istream_with_assign, ostream_withassign and iostream_withassign add assignment operators to these classes.

Table of stream classes for console operations

Class	Function
Istream	Getline(), read()
Ostream	Put(), write()
Iostream	All functions of istream & ostream.

Unformatted Console I/O Operations

Unformatted Input/Output is the most basic form of input/output.

It is the simplest and most efficient form of input/output. It is usually the most compact way to store data. Unformatted input/output transfers the internal binary representation of the data directly between memory and the file.

Advantages and Disadvantages of Unformatted I/O

1) Unformatted input/output is the least portable form of input/output. Unformatted data files can only be moved easily to and from computers when share the same internal data representation.

2) Unformatted input/output is not directly human readable, so you cannot type it out on a terminal screen or edit it with a text editor.

Portable Unformatted Input/Output :

Normally, unformatted input/output is not portable between different machine architectures because of differences in the way various machines represent binary data.

However, it is possible to produce binary files that are portable by specifying the XDR keyword with the OPEN procedures. XDR (for eXternal Data Representation) is a scheme under which all binary data is written using a standard "canonical" representation. All machines supporting XDR understand this standard representation and have the ability to convert between it and their own internal representation.

The following functions performed unformatted I/O Operations.

1) Overloaded Operators >> and <<
2) put() and get() function
3) getline() and write() function

Formatted Console I/O Operations

Formatted input/output is very portable Formatted input reads characters from the input file and converts them to internal form. Formatted I/O can be either "Free" format or "Explicit" format.

Free Format I/O :

With free format input/output, IDL uses default rules to format the data. Free format is extremely simple and easy to use. It provides the ability to handle the majority of formatted input/output needs with a minimum of effort.

Explicit Format I/O :

Explicit format I/O allows you to specify the exact format for input / output.

Advantages of Formatted I/O :

Formatted input/output operation is a simple process to move formatted data files to various computers, even computers running different operating systems, as long as they all use the ASCII character set. (ASCII is the American Standard Code for Information Interchange. It is the character set used by almost all current computers, with the notable exception of large IBM mainframes.) Formatted files are human readable and can be typed to the terminal screen or edited with a text editor.

Disadvantages of Formatted I/O :

Formatted input/output is more computationally expensive than unformatted input/output because of the need to convert between internal binary data and ASCII text. Formatted data requires more space than unformatted to represent the same information.

C++ supports a number of features that could be used for formatting the output. These features are:

1) ios class functions & flags
2) Manipulators
3) User-Defined output functions

ios class function & flags

The ios class contains a large number of member functions that help user to format the output in a number of ways. Here the table is:

List of IOS Format Functions:

Functions	Task
1. Width()	To specify required field size for displaying output value
2. Precision()	To specify no of digits after decimal point of a float value
3. Fill()	To fill unused portion of a field.
4. Setf()	To specify format flags that can control the form of output display(i.e. left justified and right justified).
5. Unsetf()	To clear the flags specified.

width()

The default width of output will be just enough space to print the number, character, or string in the output buffer.

But user can change this by using width() function. Which is invoked with cout object.

It only changes the width of the very next output field and then immediately reverts to the default.

Syntax: cout . width(w);

Here w is the field width(number of columns).

Program:

```
int main()
{
 cout.width(5);
Cout<< First Output \n";
 cout<<543<<12<<"\n";
Cout<< Second Output \n";
Cout.width(5);
Cout<<543;
Cout.width(5);
Cout<<12<<"\n";
 }
```

Output:

First Output:
_ _ 54312
Second Output:
_ _ 543 _ _ _12

precision()

By default, the floating numbers are printed with six digits after the decimal point. If user want numbers print after the decimal point according to his needs, he can do this with the help of precision().

precision() retains its setting until the reset. That's why user declared only one time in his program.

Syntax: cout . Precision(d);

Here d is the number of digits to the right of the decimal point.

Program:

```
int main()
{
 cout.precision(3);
cout<<sqrt(2)<<“\n”;
cout<<3.14159<<“\n”;
cout<<2.50032;
getch();
}
```

Output:

1.141
3.142
2.5

Filling and padding : fill()

We can print the value using much larger field widths than rquired by the value. By default, unused positions of the field are filled with white spaces. To fill the unused positions by any desired character , use fill() function.

Like precision(), fill() stay in effect till we change it.

Syntax: cout . fill(ch);

Here ch represent the character which is used for filling the unused positions.

Program:

```
int main()
{
cout.fill(“#”);
 cout.width(5);
cout<< first output \n”;
 cout<<543<<12<<“\n”;
cout<< second output \n”;
cout.fill(“#”);
cout.width(5);
cout<<543;
cout.width(3);
cout<<12<<“\n”;
 }
```

Output:

First Output:
54312
Second Output:
543 ###12

Formatting Flags, Bit Fields and setf()

We have seen that when the function width() is used, the value (i.e. text or number) is printed right-justified in the field width created.

If we want to print the text or number left-justified or print the floating number in scientific notation we can use setf () function.

setf () - (setf stands for set flags).

Syntax: cout . setf (arg1, arg2);

Here arg1 is one of the formatting flags defined in the class ios.

And arg2 represent the bit field which specifies the group to which hthe formatting flags belongs.

The formatting flag specifies the format action required for the output.

Flags & bit fields for setf() function:

Format Required	Flag(arg1)	Bit-field(arg2)
Left-justified	Ios:: left	Ios::adjustfield
Right-justified	Ios::right	Ios::adjustfield
Padding after sign or base indicator	Ios::internal	Ios::adjustfield
Scientific Notation	Ios::scientific	Ios::floatfield
Fixed Point Notation	Ios:: fixed	Ios::floatfield
Decimal Base	Ios::dec	Ios::basefield
Octal Base	Ios::oct	Ios::basefield
Hexadecimal Base	Ios::hex	Ios::basefield

Example:

```
int main()
 {
 cout.setf(std::ios::left, std::ios::adjustfield);
cout << setfill('^') << setw(10) << "Hello" << "\n";
 cout.setf(std::ios::right, std::ios::adjustfield);
 cout << setfill('0') << setw(10) << "99\n";
 return 0;
 }
```

Output:

```
Hello^^^^^
000000099
```

Manipulators in C++

The header file iomanip provides a set of functions called manipulators which can be used to manipulate the output formats.

They provides the same features as that of the ios member functions and flags.

Some manipulators are more convenient to use tan their counterparts in the class ios.

Ex: cout<< manip1<<manip2<<mani3<<item1;

This kind of concatenation is useful when we want to display several columns of output.

manipulator	meaning	equivalent
setw(int w)	set the feld width to w	width()
setprecision(int d)	set the floating point precision to d	precision()
setfill(int c)	set the fill character to c	fill()
setiosflags(long f)	set the format flag f	setf()
resetiosflags(long f)	clear the flag specified by f	unsetf()
endl	insert new line and flush stream	“\n”

Example:

```
int main()
 {
 cout.setf(std::ios::left, std::ios::adjustfield);
cout << setfill('^') << setw(10) << "Hello" << "\n";
 cout.setf(std::ios::right, std::ios::adjustfield);
 cout << setfill('0') << setw(10) << "99\n";
 return 0;
 }
```

Output:

```
Hello^^^^^
000000099
```

User-Defined Manipulators

User can design oen manipulators for certain purposes.

The general form for creating a manipulators without any argument is

```
Ostream & manipulator ( ostream & output)
{
……..
……..                    //    Code
……..
Return output;
}
```

Here, manipulator is the name of manipulator which user wants.

Example:

```
#include < iostream.h>
#include < iomanip.h>

ostream&curr(ostream&ostrObj)
{
cout << fixed << setprecision(2);
cout << "Rs.";
return ostrObj;
}

void main()
{
floatamt = 10.5478;
cout << curr << amt;
}
```

Output:

Rs. 10.54