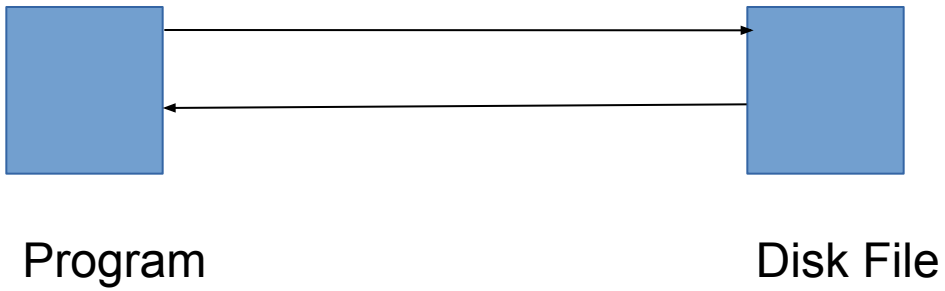
Managing Console I/O Operations

I/O System in C++

- C++ supports two complete I/O systems: the one inherited from C and another is object-oriented I/O system defined by C++ (hereafter called simply the C++ I/O system).
- The different aspects of C++'s I/O system, such as console I/O (terminal I/O) and disk file I/O (or simply disk I/O), are actually just different perspectives on the same mechanism.
- Every program takes some data as input and generates processed data as output following the **input-process-output cycle**.
- C++ supports all of C's rich set of I/O functions that can be used in the C++ programs.

I/O System in C++ (Cont...)

- C++ programmer should not use C I/O system due to two reasons :
- First I/O methods in C++ supports the concept of OOP but I/O method of C does not.
- Secondly I/O methods in C can not handle the user defined data types such as class objects.
- C++ uses the concept of streams and stream classes to implement its I/O operation with the console and disk files.
- A stream is a general name given to a flow of data. In C++ a stream is represented by an object of a particular class. So far we've used the cin and cout stream objects. Different streams are used to represent different kinds of data flow. For example, the ifstream class represents data flow from input disk files.



C++ streams

- C++ comes with libraries which provides us many ways for performing input and output. In C++ input and output is performed in the form of sequence of bytes or more commonly known as **streams**.
- **Input Stream:** If the direction of flow of bytes is from device(for example: Keyboard) to the main memory then this process is called input.
- **Output Stream:** If the direction of flow of bytes is opposite, i.e. from main memory to device(display screen) then this process is called output.
- **Note:**

cin, is an object of istream_withassign, normally used for keyboard input

cout, is an object of ostream_withassign, normally used for screen display

C++ Stream Classes

- 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.

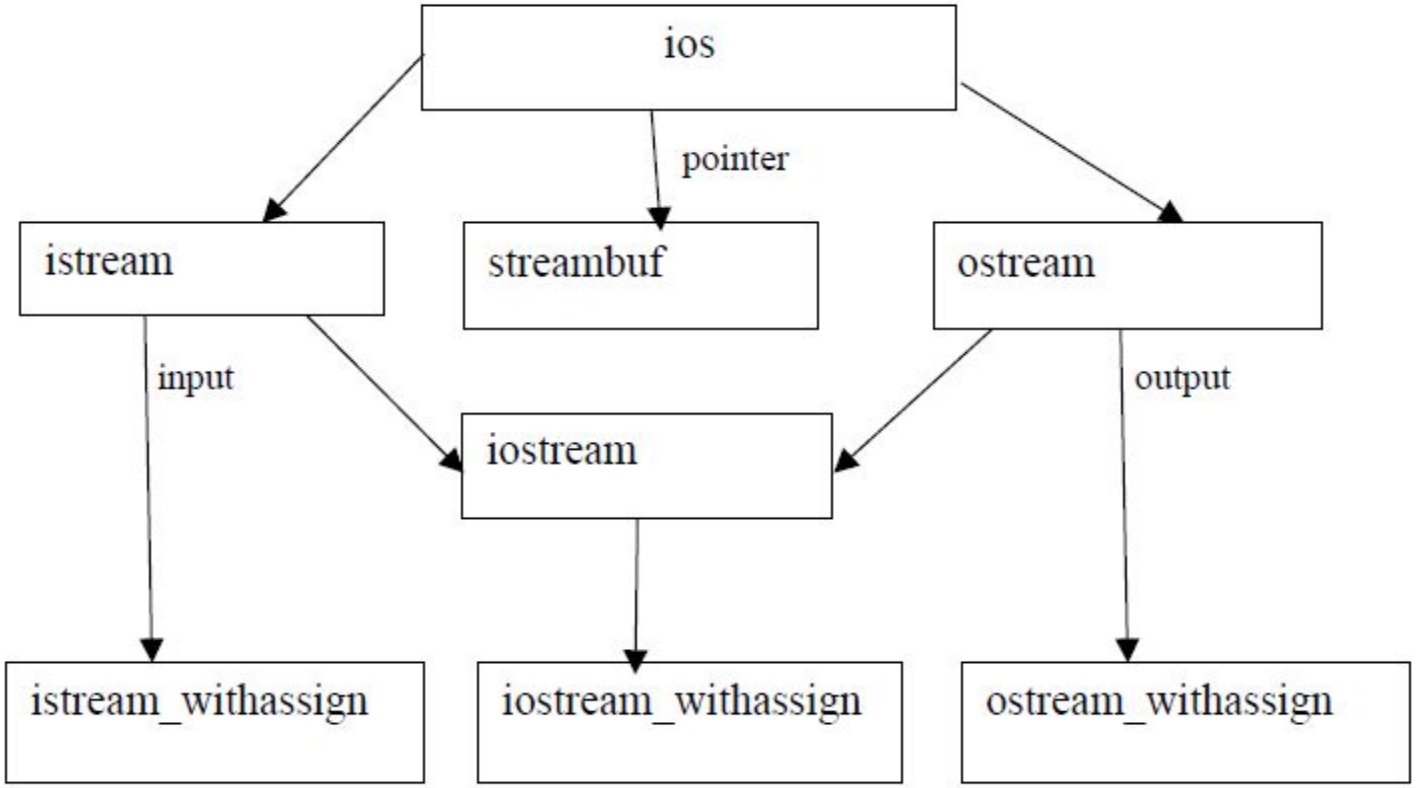


Fig:Stream classes for console I/O operations

C++ Stream Classes

Class name	Contents
ios(General input/output stream class)	Contains basic facilities that are ued by all other input and output classes Also contains a pointer to buffer object(streambuf object) Declares constants and functions that are necessary for handling formatted input and output operations
istream(input stream)	Inherits the properties of ios Declares input functions such as get(),getline() and read() Contains overloaded extraction operator>>
ostream(output stream)	Inherits the property of ios Declares output functions put() and write() Contains overloaded insertion operator <<
iostream (input/output stream)	Inherits the properties of ios stream and ostream through multiple inheritance and thus contains all the input and output functions
streanbuf	Provides an interface to physical devices through buffer Acts as a base for filebuf class used ios files

Overloaded operators >> and<<

- Objects cin and cout are used for input and output of data by using the overloading of >> and << operators.
- The >> operator is overloaded in the istream class and << is overloaded in the ostream class.
- The following are the formats for reading and displaying :

  cin>>variable1>>variable2>>.....>>variable n

  cout <<item1<<item2<<.....<<item n

put() and get() functions

- The classes istream and ostream define two member functions get(),put() respectively to handle the single character input/output operations.
- There are two types of get() functions.
- Both get(char *) and get(void) prototype can be used to fetch a character including the blank space,tab and newline character. **The get(char *) version assigns the input character to its argument**
- and **the get(void) version returns the input character.**
- Since these functions are members of input/output Stream classes,these must be invoked using appropriate objects.

put() and get() functions

```
#include<iostream>
using namespace std;
int main()
{
char c;
cin.get( c ); //get a character from the keyboard and assigns it to c
while( c!='\n')
{ cout<< c; //display the character on screen
  cin.get( c ); //get another character
}
```

} The get(void) version is used as follows:

```
char c;
c= cin.get();
```

- **The value returned by the function get() is assigned to the variable c.**

put() and get() functions

```
#include <iostream>
using namespace std;
int main()
{
int count=0;
char c;
cout<<”INPUT TEXT \n”;
cin.get( c );
while ( c !=’\n’ )
{ cout.put( c);
count++;
cin.get( c );
}
cout<< “\n Number of characters =” <<count <<”\n”;
return 0;
}
```

getline() and write() functions:

- 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 as follows:

  cin.getline(line,size);
- The reading is terminated as soon as either the newline character '\n' is encountered or size-1 characters are read(whichever occurs first).
- The newline character is read but not saved. instead it is replaced by the null character.
- For example consider the following code:

  char name[20];

  cin.getline(name,20);

getline() and write() functions:

Assume that we have given the following input through key board:

Bjarne Stroustrup<press Enter >

This input will be read correctly and assigned to the character array name.

Let us suppose the input is as follows:

Object Oriented Programming<press Enter>

In this case ,the input will be terminated after reading the following 19 characters

Object Oriented Pro

Reading Strings With getline()

```
int main() {
    int size=20;
    char city[20];
    cout<<"enter city name: ";
    cin>>city; // cin cannot read white spaces
    cout<<"city name:"<<city<<"\n";
    cin.ignore();
    cout<<"enter city name again: ";
    cin.getline(city,size);
    cout<<"city name now:"<<city<<"\n";
    cout<<"enter another city name: ";
    cin.getline(city,size);
    cout <<"New city name:"<<city<<"\n";
    cin.get();
    return 0; }
```

Reading Strings With getline()

output would be:

first run:

```
enter city name: Indore
city name:Indore
enter city name again: Kota
city name now:Kota
enter another city name: Guna
New city name:Guna
```

Second run :

```
enter city name: New Delhi
city name:New
enter city name again: city name now:Delhi
enter another city name: New Indore
New city name:New Indore
```

write() function

- The write() function displays an entire line and has the following form:

  cout.write(line,size);

  line represents the name of the string to be displayed

  second argument **size** indicates the number of characters

- **Write function does not stops displaying the characters automatically when the null character is encountered.**
- Eg. char *str="C++";
- cout.write(str,10);
- It will display C++ [7 garbage characters]

Displaying String With write()

```
#include <iostream>
#include<string>
using namespace std;
int main()
{  char * string1="C++";
   char * string2 ="Programming";
   int m=strlen(string1);
   int n =strlen(string2);
   for (int i=1;i<n;i++)
   {
       cout.write(string2,i);     cout<<"\n";
}
for (i=n;i>0;i--)
{  cout.write(string2,i);  cout<<"\n";}

cout.write(string1,m).write(string2,n); //concatenating strings
cout<<"\n";
cout.write(string1,10); //crossing the boundary
return 0;
}
```

Displaying String With write()

output

```
P
Pr
Pro
Prog
Progr
Progra
Program
Programm
Programmi
Programmin
Programming
Programmin
Programmi
Programm
Program
Progra
Progr
Prog
Pro
Pr
P
C++ Programming
C++ Progr @!
```

Formatting Console I/O Operations

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

1. ios class function and flags.
2. manipulators.
3. User-defined output functions.

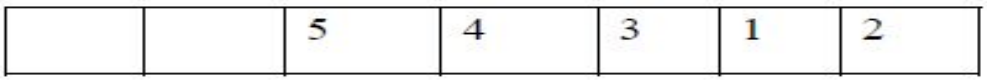
Formatted Console I/O Operations

Manipuators	Equivalent ios function	Task
setw()	width()	To specify the required field size for displaying an output value
setprecision()	precision()	To specify the number of digits to be displayed after the decimal point of float value
setfill()	fill()	To specify a character that is used to fill the unused portion of a field
setiosflags()	setf()	To specify format flags that can control the form of output display(such as left-justification and right-justification)
		To clear the flags specified

Defining Field Width:width()

- The width() function is used to define the width of a field necessary for the output of an item.As it is a member function, object is required to invoke it like
- cout.width(w); here w is the field width.
- Example:

```
cout.width(5);
cout<<543<<12<<"\n";
```



put:

- cout.width(5); cout<<543; cout.width(5); cout<<12<<"\n";
- This produces the following output

Defining Field Width:width()

```
#include <iostream>
using namespace std;
int main()
{
int items[4] ={ 10,8,12,15};
int cost[4]={75,100,60,99};
cout.width(5);
cout<<"Items";
cout.width(8);
cout<<"Cost";
cout.width(15);
cout<<"Total Value"<<"\n";
int sum=0;
```

```
for(int i=0;i<4 ;i++)
{
cout.width(5);
cout<<items[i];
cout.width(8);
cout<<cost[i];
int value = items[i] * cost[i];
cout.width(15);
cout<<value<<"\n";
sum= sum + value;
}
cout<<"\n Grand total = ";
cout<<sum<<"\n";
return 0;
}
```

Setting Precision: precision():-

- We can specify the number of digits to be displayed after the decimal point while printing the floating point numbers.
- This can be done by using the precision () member function as follows:

  cout.precision(d); where d is the number of digits.
- Eg:

```
cout.precision(4);
cout<<sqrt(2)<<"\n";
cout<<3.14159<<"\n";
cout<<2.50032<<"\n";
```

Note:Unlike the function width(),precision() retains the setting in effect until it is reset.That is why we have declared only one statement for precision setting which is used by all the three outputs.

will produce the following output:

1.141 (truncated)

3.142 (rounded to nearest cent)

2.5 (no trailing zeros)

cout<<std::fixed; should be use if expected precision after decimal point is needed

FILLING AND PADDING :fill()

- The unused portion of field width are filled with white spaces, by default. The fill() function can be used to fill the unused positions by any desired character.It is used in the following form:

  cout.fill(ch);

Where ch represents the character which is used for filling the unused positions.

Example:

```
cout.fill('*');
cout.width(10);
cout<<5250<<"\n";
```

The output would be:

*	*	*	*	*	*	5	2	5	0

FILLING AND PADDING :fill()

```
#include<iostream>
using namespace std;
int main()
{ cout.fill('<');
cout.precision(3);
for(int n=1;n<6;n++)
{
cout.width(5);
cout<<n;
cout.width(10);
cout<<1.0/float(n)<<"\n";
if(n==3)
cout.fill('>');
}
cout<<"\nPadding changed \n\n";
cout.fill('#'); //fill() reset
cout.width(15);
cout.precision(5);
cout<<12.345678<<"\n";
return 0;
}
```

```
<<<<1<<<<<<<<<1
<<<<2<<<<<<<<0.5
<<<<3<<<<<<0.333
>>>>4>>>>>>>0.25
>>>>5>>>>>>>>0.2
PADDING CHANGED
##########12.346
```

FORMATTING FLAGS,Bit Fields and setf():-

- The setf() a member function of the ios class, can provide answers left justified.The setf() function can be used as follows:
- cout.setf(arg1,arg2)

  arg1- formatting flags defined in the class ios. The formatting flag specifies the format action required for the output

  arg2-known as **bit field** which specifies the group to which the formatting flag belongs.

for example:

- cout.setf(ios::left,ios::adjustfield);
- cout.setf(ios::scientific,ios::floatfield);
- Note that the first argument should be one of the group member of second argument.

FORMATTING FLAGS,Bit Fields and setf():-

Table shows the bit fields, flags and their format actions. There are three bit fields and each has a group of format flags which are mutually exclusive.

Format required	Flag(arg1)	Bit-Field(arg2)
Left-justified output	ios::left	ios::adjustfield
Right-justified output	ios::right	ios::adjustfield
Padding after sign or base Indicater(like +##20)	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

FORMATTING FLAGS,Bit Fields and setf():-

- Consider the following segment of code:

```
cout.fill('*');
cout.setf(ios::left,ios::adjustfield);
cout.width(15);
cout<<"TABLE  1"<<"\n";
```

- This will produce the following output:

T	A	B	L	E		1	*	*	*	*	*	*	*	*

cout.fill('*'); cout.precision(3);
cout.setf(ios::internal,ios::adjustfield);
cout.setf(ios::scientific,ios::floatfield);
cout.width(15);
cout<<-12.34567<<"\n";
Will produce the following output:

-	*	*	*	*	*	1	.	2	3	5	e	+	0	1

Managing Output with Manipulators:-

- The header file **iomanip** provides a set of functions called manipulators which can be used to manipulate the output format. They provide the same features as that of the ios member functions and flags.
- For example,two or more manipulators can be used as a chain in one statement as follows

  cout<<manip1<<manip2<<manip3<<item;

  cout<<manip1<<item1<<manip2<<item2;

Managing Output with Manipulators:-

- Examples of manipulators are given below:

  cout<<setw(10)<<12345;

  This statement prints the value 12345 right-justified in a field of 10 characters. The output can be made left-justified by modifying the statement as follows:
- cout<<setw(10)<<setiosflags(ios::left)<<12345;
- **One statement can be used to format output for two or more values.**
- For example, the statement
- cout<<setw(5)<<setprecision(2)<<1.2345<<setw(10)<<setprecision(4)<<sqrt(2)<<setw(15)<<setiosflags(ios::scientific)<<sqrt(3);
- will print all the three values in one line with the field sizes of 5,10,15 respectively.

Display Trailing Zeros and Plus sign

- If we print the number 10.75,25.00 and 15.50 using a field width of 8 position and with two digit fixed precision what will be output

- How will we

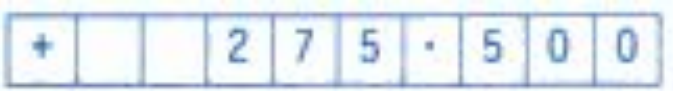
			1	0	.	7	5
						2	5
				1	5	.	5

10.75
25.00
15.50

Display Trailing Zeros and Plus sign

- setf() can be used for this purpose with single argument ios::showpoint
- Example: cout.setf(ios::showpoint);//display trailing zeros
- Similarly, plus sign can be printed before a positive number using following statement:
- cout.setf(ios::showpos);//Shows + sign
- Example

```
cout.setf(ios::showpoint);  cout.setf(ios::showpos);
cout.precision(3);
cout.setf(ios::fixed,ios::floatfield);
cout.setf(ios::internal,ios:: adjustfield);
cout.width(10); cout<<275.5<<endl;
```



Predict the Output

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    double pi = 3.14159, npi = -3.14159;
    cout << fixed << setprecision(0) << pi <<" "<<npi<<endl;
    cout << fixed << setprecision(1) << pi <<" "<<npi<<endl;
    cout << fixed << setprecision(2) << pi <<" "<<npi<<endl;
    cout << fixed << setprecision(3) << pi <<" "<<npi<<endl;
    cout << fixed << setprecision(4) << pi <<" "<<npi<<endl;
    cout << fixed << setprecision(5) << pi <<" "<<npi<<endl;
    cout << fixed << setprecision(6) << pi <<" "<<npi<<endl;
}
```

```
3 -3
3.1 -3.1
3.14 -3.14
3.142 -3.142
3.1416 -3.1416
3.14159 -3.14159
3.141590 -3.141590
```

Predict the Output

```
int main ()
{
 int n = -77;
std::cout.width(6);
std::cout << std::internal << n << “\n”;
std::cout.width(6); std::cout << std::left << n << “\n”;
std::cout.width(6);
std::cout << std::right << n << “\n”;
 return 0;
}
```

Creating user defined manipulators

- **Syntax for creating user defined manipulators in C++**
- ostream & manipulator_name(ostream & output_str)
  {
  //set of statements;
  return output_str;
  }
- Example:

```
ostream & unit (ostream & output)
{
    output<<“inches”;
    return output;
}
```

The statement **cout<<50<<unit;** will print **50 inches**

Example

Consider the following example which creates a user defined manipulator named curr for displaying Rs. and sets the precision to 2.

```
#include <iostream>
using namespace std;
#include <iomanip>
ostream & curr(ostream & ostr)
{
ostr<< setprecision(2);
ostr<<"Rs." ;

return ostr;
}
int main()
{
float amt = 4.5476;
cout<<curr<<amt;
return 0;
}
```

Output?

```
cout.precision(2);
cout<<3.555<<endl;
int p=cout.precision(4);//returns the last
   precision value
cout<<3.555<<endl;
cout.precision(p);
cout<<3.555;
```

Output:

3.5

3.555

3.5