Console Input / Output Operations

* C++ uses the concept of streams and stream classes to perform I/o operations with console.
* A stream is a sequence of bytes.
* In C++, stream refers to flow of data b/w a particular device (console, keyboard, files, memory etc) and the program's variables.
* Streams are classified into

(i) Input Stream:

- source stream that provides data to the program.

(ii) Output stream:

- Destination stream that receives output from the program.

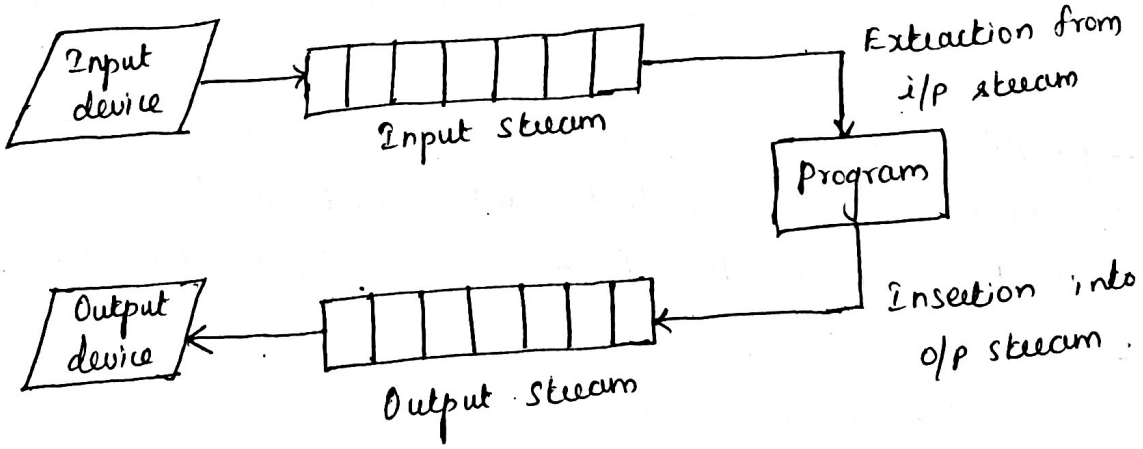


Fig: Data streams

Predefined console streams

* C++ supports pre-defined streams, that are automatically opened when a program begins its execution. They are
  - cin → represents input stream connected to standard i/p device (keyboard)
  - cout → represents output stream connected to standard o/p device (Monitor)

C++ Stream Classes

* The C++ I/O system contains a hierarchy of classes, that are used to define various streams to deal with both console and disk files. These classes are called stream classes.

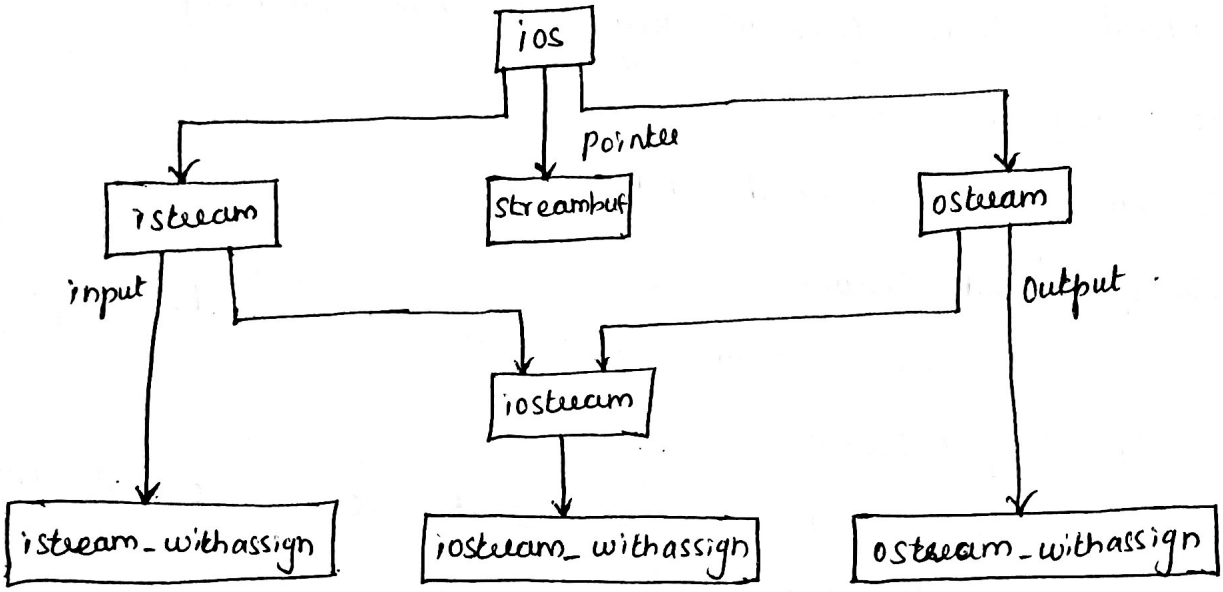


Fig: Stream classes for console I/o operations.

* ios class
  - provides operations common to both input and output.
  - supports both formatted and unformatted I/o operations.
  - contains a pointer to a buffer object (streambuf)
  - base class for
    - ↳ istream (Input stream)
    - ↳ ostream (Output stream)
    - ↳ iostream (Input/Output stream)
  - The class ios is declared as virtual base class, so that only one copy of its members are inherited by iostream.

* istream class
  - inherits the properties of ios
  - defines input functions such as get(), getline() and read()
  - Has overloaded stream extraction operator >> , to read data from standard i/p device.

* ostream class
  - inherits properties of ios
  - defines output functions such as put() and write().
  - Has overloaded stream insertion operator << , to write data to standard output device.

* iostream class
  - derived from multiple base classes, istream and ostream, which are in turn inherited from ios class.
  - handles both input and output streams, and supports all operations provided by istream and ostream classes.

* istream_withassign, ostream_withassign and iostream_withassign classes
  - add assignment operators to their parent classes.

* streambuf
  - provides an interface to physical devices through buffers.

Note: cin is the instance of the class istream_withassign and
cout is " " " ostream_withassign.

Unformatted I/O operations

* Overloaded operators >> and <<

- The extraction operator >> , gets data from the predefined input stream cin .

Syntax :

```
cin >> var1 ;
```

- The operator >> reads the data character by character and assigns it to indicated location, until a white space is encountered .
- Multiple stream operators can be cascaded on a single line .

Syntax :

```
cin >> var1 >> var2 >> var3 ;
```

- The insertion operator << , outputs data to predefined output stream cout .

Syntax :

```
cout << var1 ;
```

- cascaded insertion operator is as follows

Syntax :

```
cout << var1 << " " << var2 ;
```

* put() and get() functions :

- get()
  - → member function of istream class
  - → used to read single character from i/p device .
  - → can read blank space , tab and new line character .

, Two types of get() functions are available.

(i) get (char *)
- assigns i/p character to its argument

(Eg)
```
char ch;
cin.get(ch) // gets a char from KB
cout << ch;   * assigns to ch.
```

* To read and display a line (terminated by newline character)

```
#include <iostream.h>
void main()
{
 char ch;
 cin.get(ch);
 while (ch != '\n')
  {
   cout << ch;
   cin.get(ch);
  }
}
```

(ii) get (void)
- returns i/p character.

(Eg)
```
char ch;
ch = cin.get();
cout << ch;
```

* To read * display a line

```
void main()
{
 char ch;
 ch = cin.get();
  .
  .
  .
 ch = cin.get();
}
```

- put()
  → member function of ostream class.
  → used to output a line of text character by character.

(Eg)
```
cout.put('A');   // prints A.
cout.put(ch);    // prints the value of variable ch.
cout.put(68);    // prints character D by converting int to char value * display D whose ASCII value is 68.
```

* To read line of text * display,

```
char c;
cin.get(c);
while (c != '\n')
  {
   cout.put(c);
   cin.get(c);
  }
```

* getline() and write() functions (line-oriented function)

- getline()
  - ⇝ reads a whole line of text until a new line character is encountered or until the maximum limit-1 is reached.
  - ⇝ can also read white space character.

Syntax:

```
cin.getline (line, size);
```

(Eg)
```
void main()
{
char c[5];
cout << "Enter text \n";
cin.getline (c,5);
cout << c;
}
```

o/p

```
Enter text :
Apple
Appl
```

Note: Maximum no. of characters extracted is size-1.

- write()
  - ⇝ displays an entire line

Syntax:

```
cout.write (line, size);
```

(size → no. of characters to display; line → name of string to be displayed.)

(Eg)
```
#include<iostream.h>
void main()
{
char *string1 = "Programming";
cout.write (string1, 5);
cout << "\n";
cout.write (string1, 15); // size is greater than length & displays beyond bounds of line.
cout.write (string1, 3).write (string1, 11); // concatenates 2 strings.
}
```

o/p:

```
Progr
Programming
Pro programming
```

Formatted console I/O Operations

* C++ supports the following features for formatting the output.
  (i) ios class functions and flags
  (ii) Manipulators
  (iii) User-defined output functions (or) User-defined manipulator functions.

(i) ios class functions and flags

* Most important functions are as follows:

a) width()

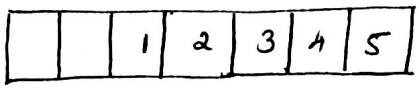
- defines field width of the output

Syntax:

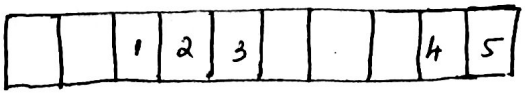
```
cout.width(w);
```

- w → field width (no. of columns)
- o/p will be displayed at right end of field by default.
- This fn is applicable for only one item that immediately follows the fn.

(Eg)
```
cout.width(5);
cout << 123 << 45;
```




```
cout.width(5);
cout << 123;
cout.width(5);
cout << 45;
```



* Thus field width should be specified for each item separately

Note: If size of value is greater than field width, C++ expands the field to fit the value.

b) precision()

- used to specify the no. of digits to be displayed after decimal point while printing floating-point number.
- by default, the precision size is 6.

Syntax:

```
cout.precision(d);
```

d → no. of digits to the right of decimal point.

(Eg) cout.precision(2);

cout << 2.232 << endl; ⇒ prints 2.23

cout << 9.168 << endl; ⇒ 9.17 } rounded

cout << 3.500 << endl; ⇒ 3.5 } no trailing zeros

cout << 6.002 << endl; ⇒ 6

Note: precision() will retains its settings until it is reset. Hence only one precision() stmt is used for all o/p stmts.

c) fill()

- used to fill and display the unused positions of the field by any desired character.
- By default, the unused positions are filled with white spaces.

Syntax:

```
cout.fill(ch);
```

(Eg)
```
cout.fill('*');
cout.width(6);
cout << 625;
```

*	*	*	6	2	5

Note:- fill() also retains its settings until it is reset.
- used in banks for filling or padding unused characters, while printing cheques to avoid forgery

(Eg)
```
#include <iostream.h>
void main()
{
  cout.fill('#')
  cout.precision(2);
  for(int i=1; i<=3; i++)
  {
    cout.width(3);
    cout<<i;
    cout.width(6);
    float f=0.333;
    cout<<(i+f)<<endl;
  }
}
```

o/p:

```
##1##1.33
##2##2.33
##3##3.33
```

d) setf()

- used to set flags and bit-fields that control the output.
  - (Eg) to display the o/p in left justified, scientific notation for floating pt. number etc.

Syntax:

```
cout.setf(arg1, arg2);
```

arg1 → one of formatting flags defined in ios.
arg2 → bit field to which the formatting flag belongs.

- There are 3 bit fields and each has a group of format flags.

Format	Flag(arg1)	Bitfield(arg2)
Left-justified o/p	ios::left	ios::adjustfield
Right-justified o/p	ios::right	ios::adjustfield
Padding after sign(or) base	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

(Eg) cout.fill(*);
cout.setf(ios::left, ios::adjustfield);
cout.width(10);
cout << "APPLES" << "\n";

A	P	P	L	E	S	*	*	*	*

(Eg) cout.fill(*);
cout.precision(3);
cout.setf(ios::internal, ios::adjustfield);
cout.setf(ios::scientific, ios::floatfield);
cout.width(15);
cout << -12.31567 << "\n";

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

* To display trailing zeros

`cout.setf(ios::showpoint);` ⇒ single arg flag.
⇒ do not have bit fields.

* To display plus sign before positive number

`cout.setf(ios::showpos);`

(Eg)
```
void main()
{
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 << "\n";
}
```

o/p:

+			2	7	5	.	5	0	0

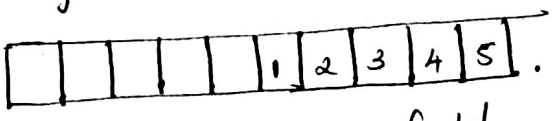
(ii) Manipulators

* Manipulators are set of functions used to control input/output streams using extraction (>>) and insertion (<<) operators.
  - Thus manipulators manipulate the output formats.

* To access the manipulators, should include iomanip.h header file in the program.

Manipulators	Meaning	Equivalent
(i) setw(int w)	Set the field width to w	width( )
(ii) setprecision(int d)	Set the floating point precision to d.	precision( )
(iii) setfill(int c)	Set the fill character to c.	fill( ).
(iv) endl	Insert new line and flush stream	"\n".
(v) setiosflags(long f)	Set the format flag f	setf( )

(Eg(i)) cout << setw(10) << 12345;

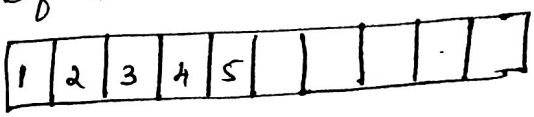


↳ prints the value right justified in a field width of 10 characters.

(Eg(ii)) cout << setw(10) << setiosflags(ios::left) << 12345;

↳ one statement is used to add more than one manipulators

↳ here the o/p is left justified



(iii) User-defined Manipulator Functions

* Based on the program requirement, manipulators can be defined onown own.

Syntax for defining manipulator:

```
ostream & manipulatorname (ostream & output)
    {
      Stmts;
      =
      return output;
    }
```

(Eg)

```
#include <iostream.h>
#include <iomanip.h>
ostream & curr (ostream & op)
  {
   cout << setprecision(2);
   cout << "Rs.";
   return op;
  }
void main()
 {
  float amt = 75.6785;
  cout << curr << amt;
 }
```

O/P

Rs. 75.67

* Here the user-defined manipulator named 'curr' is created for displaying Rs. and to set the precision to 2.