FAST

الذی علم بالقلم۔ علم الانسان ما لم یعلم۔

National University of Computer and Emerging Sciences

DEPARMENT OF COMPUTER SCIENCE

Object Oriented Programming - Lab

Lab-06

OOP LAB-06
INPUT/OUTPUT IN C++

Engr. Khuram Shahzad
(Instructor)

OOP Lab 06 Content

- Stream: Examine input and output streams
- Read data from the standard input device
- Explore how to use the input stream functions get, ignore, putback, and peek.
- How to use predefined functions in a program
- Become familiar with input failure
- Learn how to write data to the standard output device
- Discover how to use manipulators in a program to format output

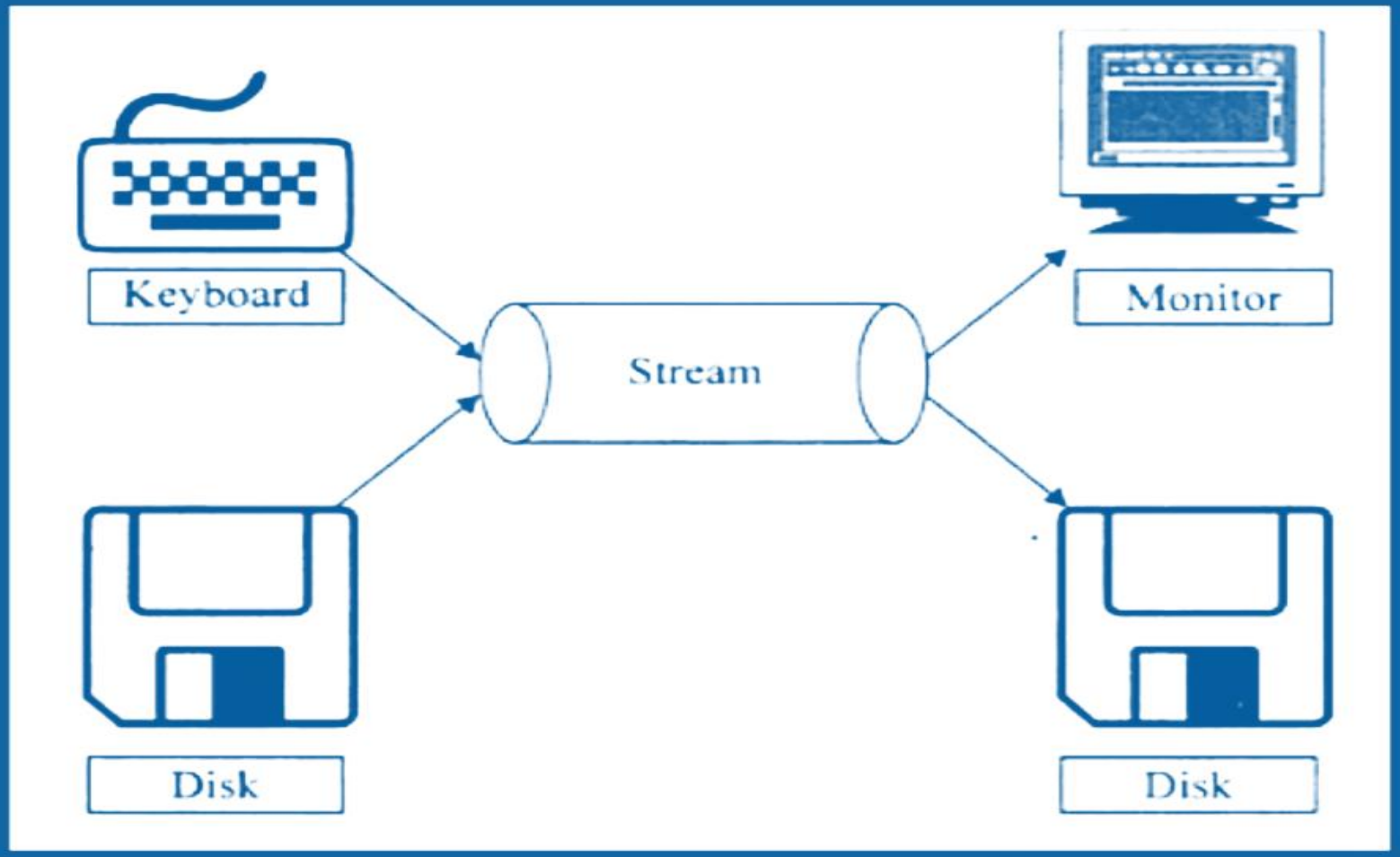
Introduction

- *Every program takes some data as input and generates processed data as output.*
- *C++ supports set of I/O functions.*
- *C++ uses the concepts of stream and stream classes to implement its I/O operations with console and disk files.*
- *In this chapter we will discuss how stream classes support the console-oriented I/O operations.*

Streams in C++

- **Stream** *is a sequence of bytes.*
- *If data is received from input devices in sequence then it is called as* **source stream.**
- *When data is passed to output devices then it is called* **destination.**
- *The data is received from keyboard or disk and can be passed on to monitor or to the disk.*

Streams in C++



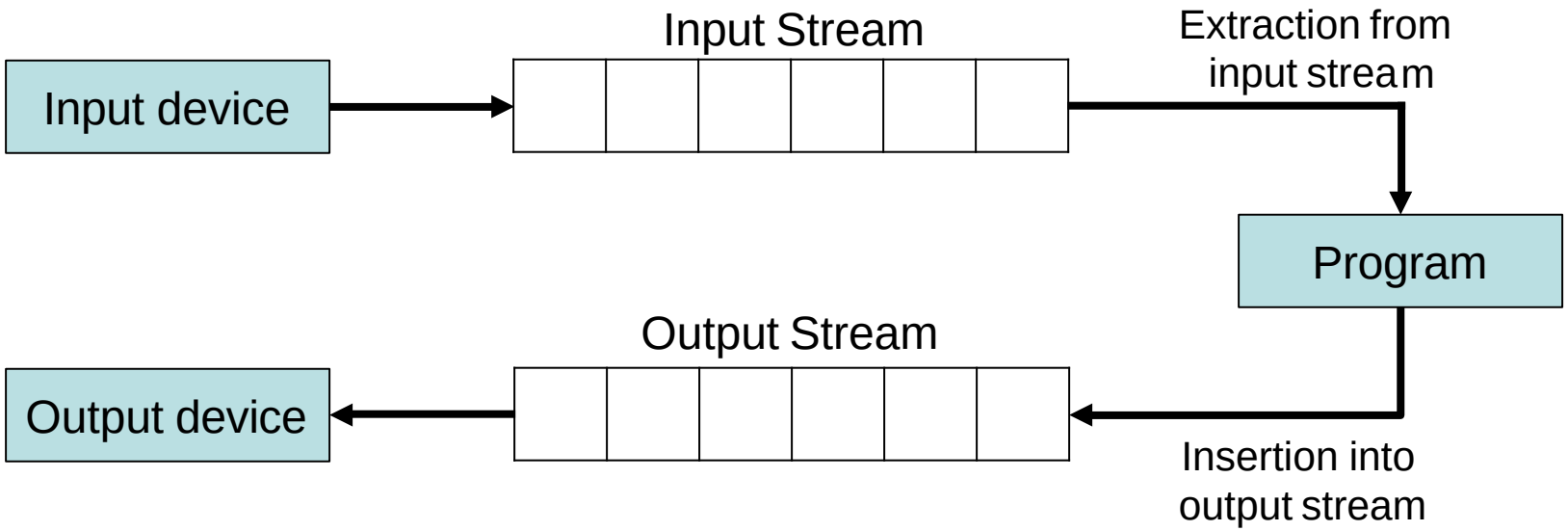
Streams in I/O devices

Streams in C++

- *Data in source stream can be used as input data by program .*
- ***Source** stream is called as **input** stream.*
- ***Destination** stream that collects output data from the program is known as **output** stream.*

STREAM IN C ++

- **Input Stream - The source stream that provides data to the program.**



- **Output Stream - The destination stream that receives output from the program.**

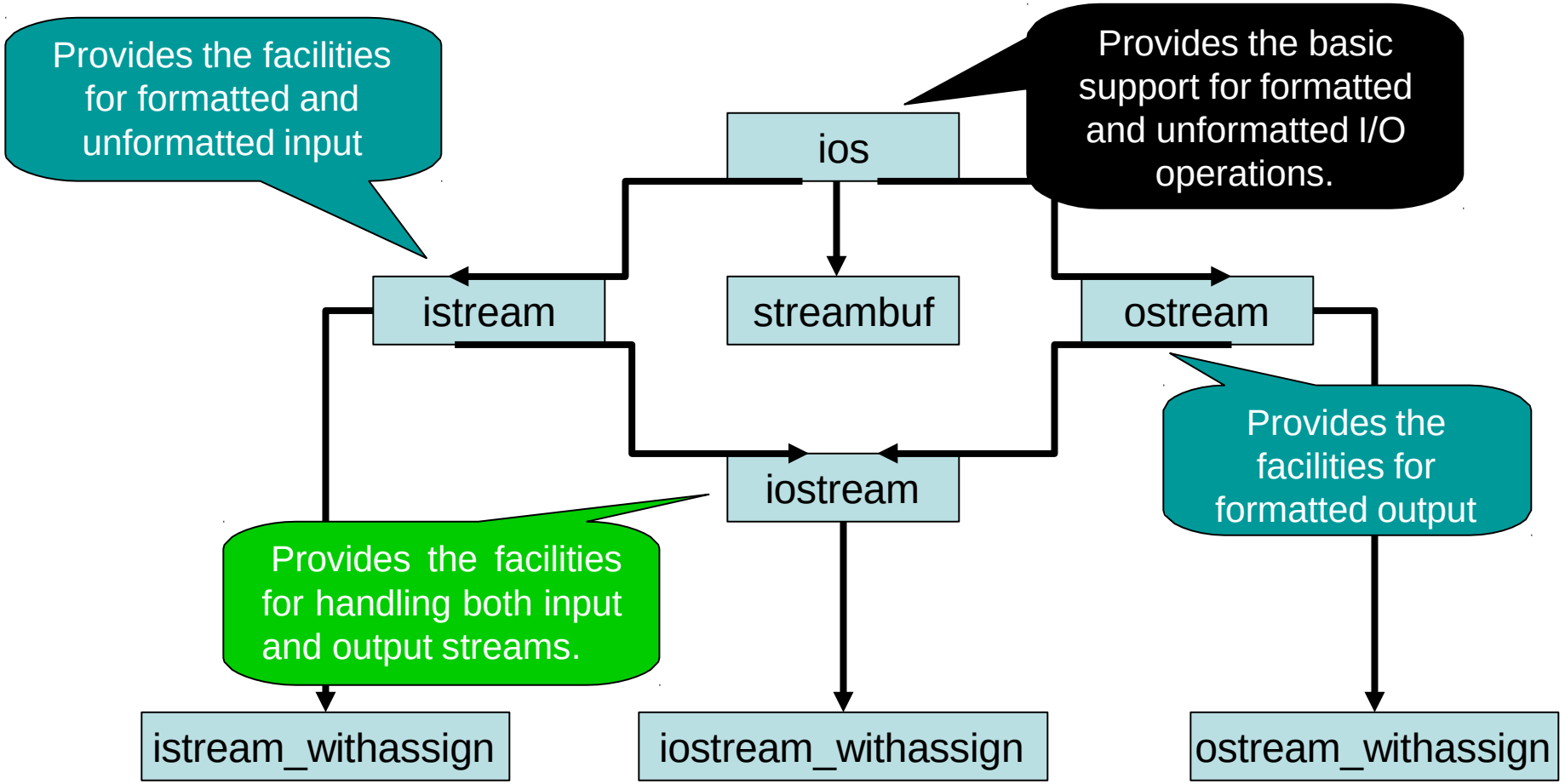
Formatted vs Unformatted I/O Function in C++ (Differences)

Input output functions in C++ programming falls into two categories, namely, **formatted input output (I/O) functions and unformatted input output (I/O) functions**. In this article we will point out major differences between them:

Difference between Formatted and Unformatted Functions

1	Formatted I/O functions allow to supply input or display output in user desired format.
1	Unformatted I/O functions are the most basic form of input and output and they do not allow to supply input or display output in user desired format.
2	Formatted input and output functions contain format specifier in their syntax.
2	Unformatted input and output functions do not contain format specifier in their syntax.
3	Formatted I/O functions are used for storing data more user friendly.
3	Unformatted I/O functions are used for storing data more compactly.
4	Formatted I/O functions are used with all data types.
4	Unformatted I/O functions are used mainly for character and string data types.

C ++ STREAM CLASSES



Streams Classes

Class Name	Contents
ios	• Contains basic facilities that are used by all other input and output classes
istream	• Inherits properties of ios. • Declares input function get(), getline(), read(). • Contains overloaded extraction >> operator
ostream	• Inherits properties of ios. • Declares input function put(), write(). • Contains overloaded insertion << operator
iostream	• Inherits properties of ios, istream and ostream.

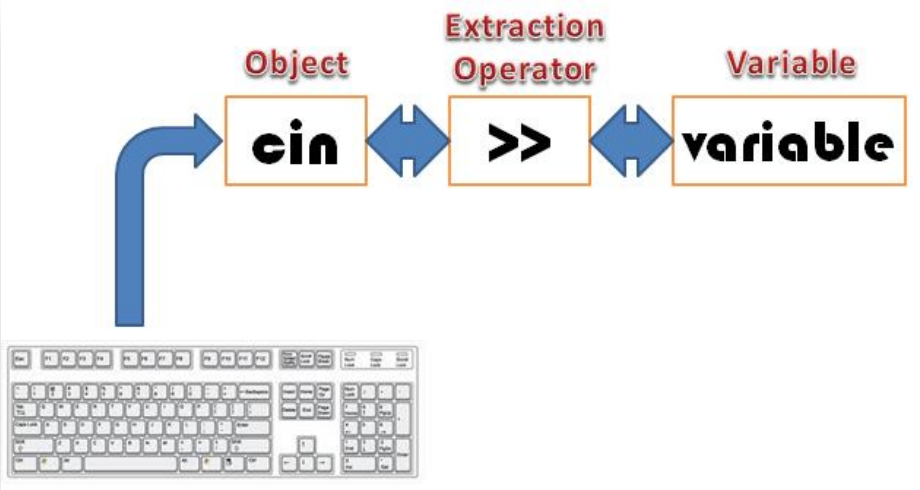
Formatted Console I/O Operations

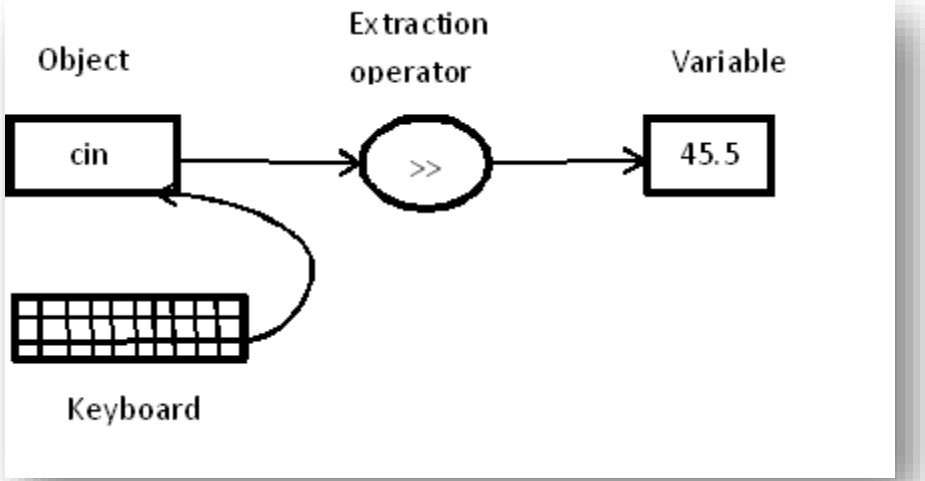
- *C++ provides various formatted console I/O functions for formatting the output.*
  1. *ios class functions and flags.*
  2. *Manipulators*
  3. *User-defined output functions*
- *ios grants operations common to both input and output.*

Manipulators are functions specifically designed to be used in conjunction with the insertion (<<) and extraction (>>) operators on stream objects

cin and the Extraction Operator >>

- The extraction operator ›› is defined only for putting data into variables of simple data types.
- Therefore, the right-side operand of the extraction operator ›› is a variable of the simple data type.
- However, C++ allows the programmer to extend the definition of the extraction operator ›› so that data can also be put into other types of variables by using an input statement.





>>	Extraction Operator
<<	Insertion Operator

cin and the Extraction Operator >>

- As you can see in the preceding syntax, a single input statement can read more than one data item by using the operator ›› several times.
- Every occurrence of ›› extracts the next data item from the input stream.
- For example, you can read both payRate and hoursWorked via a single input statement by using the following code:

Lab06`.cpp | [*] UnformattedExample.cpp | FormattedExample.cpp | CinWithExtractoreOperator.cpp

```cpp
#include <iostream>
using namespace std;
//cin and the Extraction Operator >>
int main () {

    float payRate ;
    int hoursWorked;

    cout << "Enter Pay Rate: ";
    cin>>payRate;
    cout << "Enter Hours Worked: ";
    cin>>hoursWorked;
    cout<< "Entered pay rate: " << payRate<< endl;
    cout<< "Entered Hours Worked: " << hoursWorked<< ;

    cout<<endl<< endl;
    cout<<"--------------------------"<<endl;
    cout << "Enter Pay Rate & Hours Worked respectively: ";
    cin>>payRate>>hoursWorked;
    cout<< "Entered pay rate: " << payRate<< endl;
    cout<< "Entered Hours Worked: " << hoursWorked<< endl;
    return 0;
}
```

D:\Object Oriented Language\OOP Lab- 06\CinWithExtractoreOperator.exe

```
Enter Pay Rate: 456.66
Enter Hours Worked: 554
Entered pay rate: 456.66
Entered Hours Worked: 554

--------------------------
Enter Pay Rate & Hours Worked respectively: 454.6 44
Entered pay rate: 454.6
Entered Hours Worked: 44

--------------------------------
Process exited after 16.53 seconds with return value 0
Press any key to continue . . .
```

Using Predefined Functions in a Program

- C++ has a library of predefined functions. ... For example the function that generates a random integer: rand( ). Each of the predefined function returns a single value of a specific type when it is called.
- A function is a group of statements that together perform a task. Every C++ program has at least one function, which is main
- C++ comes with a wealth of functions, called predefined functions, that are already written.

Lab06\.cpp | [*] UnformattedExample.cpp | FormattedExample.cpp | CinWithExtractoreOperator.cpp | predefinedFunction0.cpp

```
2  #include <iostream>
3  #include <cmath>
4  #include <string>
5  using namespace std;
6  int main() {
7      double u, v;
8      string str;
9      cout << "Line 1: 2 to the power of 6 = "
10         << static_cast<int>(pow(2.0, 6.0))
11         << endl; //Line 1
12     u = 12.5; //Line 2
13     v = 3.0; //Line 3
14     cout << "Line 4: " << u << " to the power of "
15         << v << " = " << pow(u, v) << endl; //Line 4
16     cout << "Line 5: Square root of 24 = "
17         << sqrt(24.0) << endl; //Line 5
18     u = pow(8.0, 2.5); //Line 6
19     cout << "Line 7: u = " << u << endl; //Line 7
20     str = "Programming with C++"; //Line 8
21     cout << "Line 9: Length of str = "
22         << str.length() << endl; //Line 9
23     return 0;
24 }
```

In C++ type conversion, the static_cast operator performs an explicit type conversion

D:\Object Oriented Language\OOP Lab- 06\predefinedFunction0.exe

```
Line 1: 2 to the power of 6 = 64
Line 4: 12.5 to the power of 3 = 1953.12
Line 5: Square root of 24 = 4.89898
Line 7: u = 181.019
Line 9: Length of str = 20

--------------------------------
Process exited after 0.1001 seconds with return value 0
Press any key to continue . . .
```

Take string in C++

Take String Input in C++

- We can take string input in four ways in C++.
- **1) Using cin**

```
#include<iostream>
using namespace std;
int main()
{
  char ch[10];
  cout<<"Enter any String: ";
  cin>>ch;
  cout<<ch;
}
```

```
Enter any String: Adil Aslam
Adil
```

Note Here We passed Adil Aslam but on console Adil is there only because **cin** when white space is encountered, the **cin** function terminates.

Take String Input in C++

- **2) Using gets()**
- Using **gets()** method we can read more than one string at a time with white space.

```
#include<iostream>
using namespace std;
int main()
{
    char str[10];
    cout<<"Enter any string: ";
    gets(str);
    cout<<"String are: ";
    puts(str);
}
```

```
Enter any String: Adil Aslam
String are: Adil Aslam
```

Note Please observe we pass **Adil Aslam** with white space both strings shown on console.

Take String Input in C++

- **3) Using get()**
- get() is function and it is use to get single character. It will work with cin object only.

```
#include<iostream>
using namespace std;
int main()
{
    char ch;
    //reads single character
    cin.get(ch);
    cout<< ch;
 return 0;
}
```

```
Adil Aslam
A
```

```
// cin and strings
#include <iostream>
#include <string>
using namespace std;
int main ()
{
        string name;
        cout << "Enter your name";
        getline (cin, name);
        cout << "Hello " << name << "!\n";
        return 0;
}
```

Output

Enter your name : Anjum

Hello Anjum!

Take String Input in C++

- **4) Using getline()**
- **getline()** is function in C++ and it is use to read one line of string that ends with new line(\n).
- It will work with **cin.**

```
#include<iostream>
using namespace std;
int main()
{
    char ch[100];
    cin.getline(ch , 11);  //reads one line
    cout<< ch;
 return 0;
}
```

```
Adil Aslam
Adil Aslam
```

Formatted Console I/O Operations

- *C++ provides various formatted console I/O functions for formatting the output.*
  1. *ios class functions and flags.*
  2. *Manipulators*
  3. *User-defined output functions*
- *ios grants operations common to both input and output.*

Manipulators are functions specifically designed to be used in conjunction with the insertion (<<) and extraction (>>) operators on stream objects

Formatted Console I/O Operations

Function	Working
width()	*To set required field width. o/p will be displayed with given width.*
precision()	*To set number of decimal point for a float value.*
fill()	*To set character to fill in the blank space of the field.*
setf()	*To set various flags for formatting output.*
unsetf()	*To remove the flags setting*

Table: *ios class functions*

Formatted Console I/O Operations

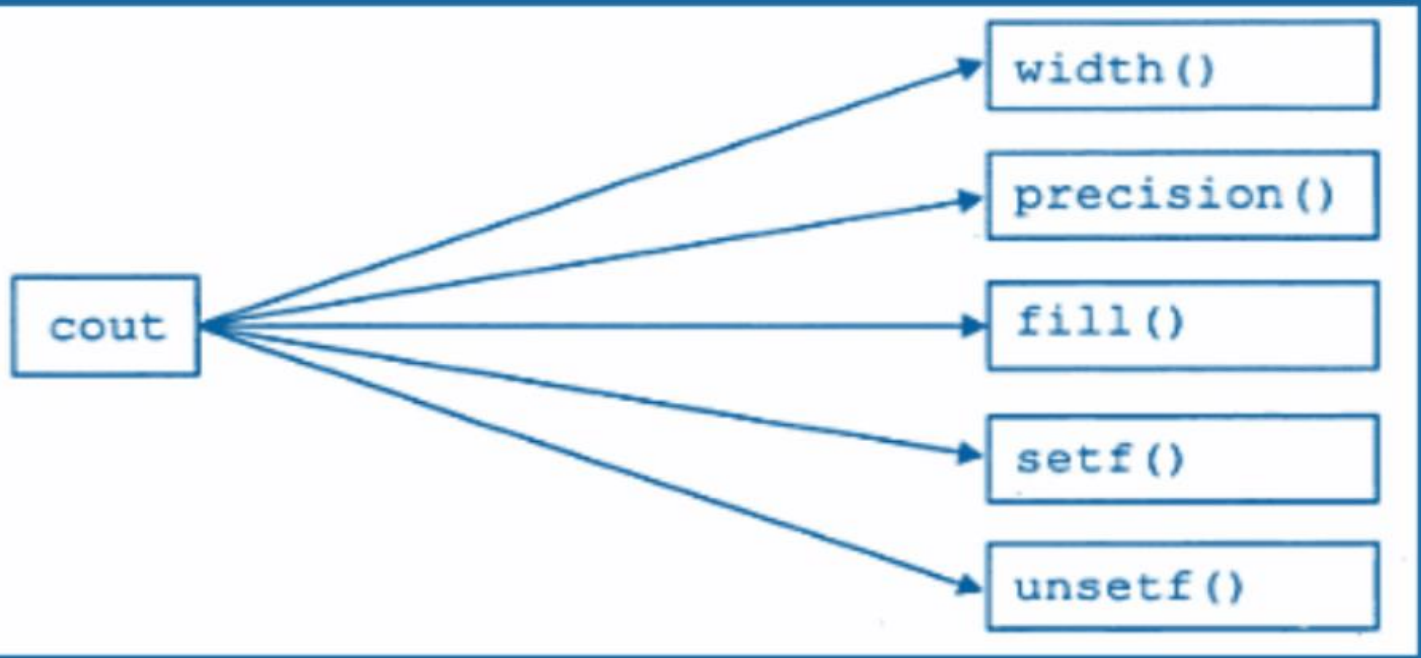


Fig: *Formatted functions with cout object*

Defining field width: width()

- width() – defines width of a field necessary for the output of an item.

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

- w - field width(number of columns).
- Output will be printed in a field of w characters wide at the right end of the field.
- Field width should be specified for each item separately.

Defining field width: width()

- For example, the statements

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

will produce following output:

- 543 is printed right-justified in the first five columns.
- width(5) does not retain the setting for printing the 12.

Defining field width: width()

- For example, the statements

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

This produce the following output:

```cpp
#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 VALUES"<<"\n";

    int sum = 0;

    for(int i = 0; i < 4; i++)
    {
        cout.width(5);
        cout<<items[i];

        cout.width(8);
        cout<<cost[i];

        int vaule = items[i] * cost[i];

        cout.width (15);
        cout<<vaule<<"\n";

        sum = sum + vaule;
    }
    cout<<"\nTotal = " << sum;

    return 0;
}
```

Executing the program....

```
$demo

ITEMS     COST    TOTAL VALUES
   10       75             750
    8      100             800
   12       60             720
   15       99            1485

Total = 3755
```

Setting precision : precision()

- *Default floating numbers are printed with six digits after the decimal point.*
- *precision() – specifies number of digits to be displayed after the decimal point while printing the floating-point numbers, has the form:*

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

- **d** *is the number of digits to the right of the decimal point.*

Setting precision : precision()

- *For Examples:*

```
cout.precision(3);
cout<<1.23456<<"\n";
cout.precision(4);
cout<<3.14159<<"\n";
```

width() and precision()

- *We can also combine the field specification with the precision setting. Example:*

```
cout.precision(2);
cout.width(5);
cout<<1.2345;
```

- *The output will be:*

	1	.	2	3

Filling and Padding :fill()

- *We can use the fill() function 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.*

Filling and Padding :fill()

- *For Example:*

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

- *The output would be:*

- *Financial institutions and banks use this kind of padding while printing cheques so that no one can change the amount easily.*

cin and the ignore Function

When you want to process only partial data (say, within a line), you can use the stream function ignore to discard a portion of the input. The syntax to use the function ignore is:

```
cin.ignore(intExp, chExp);
```

- Suppose intExp yields a value of, say 100. This statement says to ignore the next 100 characters or ignore the input until it encounters the character specified by chExp, whichever comes first. To be specific, consider the following statement: cin.ignore(100, '\n');

Consider the declaration:

```
int a, b;
```

and the input:

```
25 67 89 43 72
12 78 34
```

Now consider the following statements:

```
cin >> a;
cin.ignore(100, '\n');
cin >> b;
```

The first statement, `cin >> a;`, stores `25` in `a`. The second statement, `cin.ignore(100, '\n');`, discards all of the remaining numbers in the first line. The third statement, `cin >> b;`, stores `12` (from the next line) in `b`.

```
#include <iostream>
using namespace std;
int main () {
int a, b;
cin >> a;
cin.ignore(100, '\n');
cin >> b;
cout<< "a: "<< a<< endl;
cout<< "b: "<< b<< endl;
    return 0;
}
```

```
D:\Object Oriented Language\OOP Lab- 06\CinWithIgnore.exe
25 67 89 43 72
12 78 34
a: 25
b: 12
```

The putback and peek Functions

- The syntax for `putback`:

```
istreamVar.putback(ch);
```

  - `istreamVar`: an input stream variable (`cin`)
  - `ch` is a `char` variable

- The syntax for `peek`:

```
ch = istreamVar.peek();
```

  - `istreamVar`: an input stream variable (`cin`)
  - `ch` is a `char` variable

The peek function returns the next character from the input stream but does not remove the character from that stream.

he peek function returns the next character from the input stream but does not remove the character from that stream.

Lab06`.cpp | [*] UnformattedExample.cpp | FormattedExample.cpp | CinWithExtractoreOperator.cpp | predefinedFunction0.cpp | CinWithIgnore.c

```cpp
//Functions peek and putback
#include <iostream>
using namespace std;
int main() {
    char ch;
    cout << "Line 1: Enter a string: "; //Line 1
    cin.get(ch); //Line 2
    cout << endl; //Line 3
    cout << "Line 4: After first cin.get(ch); "
         << "ch = " << ch << endl; //Line 4
    cin.get(ch); //Line 5
    cout << "Line 6: After second cin.get(ch); "
         << "ch = " << ch << endl; //Line 6
    cin.putback(ch); //Line 7
    cin.get(ch); //Line 8
    cout << "Line 9: After putback and then "
         << "cin.get(ch); ch = " << ch << endl; //Line 9
    ch = cin.peek(); //Line 10
    cout << "Line 11: After cin.peek(); ch = "
         << ch << endl; //Line 11
    cin.get(ch); //Line 12
    cout << "Line 13: After cin.get(ch); ch = "
         << ch << endl; //Line 13
    return 0;
}
```

```
Select D:\Object Oriented Language\OOP Lab- 06\peekandputback.exe
Line 1: Enter a string: ABCD

Line 4: After first cin.get(ch); ch = A
Line 6: After second cin.get(ch); ch = B
Line 9: After putback and then cin.get(ch); ch = B
Line 11: After cin.peek(); ch = C
Line 13: After cin.get(ch); ch = C

--------------------------------
Process exited after 6.241 seconds with return value 0
Press any key to continue . . .
```

The Dot Notation between I/O Stream Variables and I/O Functions:

- In the preceding sections, you learned how to manipulate an input stream to get data into a program.
- You also learned how to use the functions get, ignore, peek, and putback.
- It is important that you use these functions exactly as shown. For example, to use the get function, you used statements such as the following

```
cin.get(ch);
```

Omitting the dot—that is, the period between the variable cin and the function name get—results in a syntax error. For example, in the statement:

```
cin.get(ch);
```

cin and get are two separate identifiers separated by a dot. In the statement:

```
cinget(ch);
```

Input Failure

```cpp
//Input Failure program
#include <iostream>
using namespace std;
int main() {
    int a = 10; //Line 1
    int b = 20; //Line 2
    int c = 30; //Line 3
    int d = 40; //Line 4
    cout << "Line 5: Enter four integers: "; //Line 5
    cin >> a >> b >> c >> d; //Line 6
    cout << endl; //Line 7
    cout << "Line 8: The numbers you entered are:"
         << endl; //Line 8
    cout << "Line 9: a = " << a << ", b = " << b
         << ", c = " << c << ", d = " << d << endl; //Line 9
    return 0;
}
```

```
D:\Object Oriented Language\OOP Lab- 06\InputFailure.exe
Line 5: Enter four integers: 34 K 67 28

Line 8: The numbers you entered are:
Line 9: a = 34, b = 0, c = 30, d = 40
```

The second input value is the character 'K'. The cin statement tries to input this character into the variable b. However, because b is an int variable, the input stream enters the fail state. Note that the values of b, c, and d are unchanged, as shown by the output of the statement in Line 9

```cpp
//Input Failure program
#include <iostream>
using namespace std;
int main() {
    int a = 10; //Line 1
    int b = 20; //Line 2
    int c = 30; //Line 3
    int d = 40; //Line 4
    cout << "Line 5: Enter four integers: "; //Line 5
    cin >> a >> b >> c >> d; //Line 6
    cout << endl; //Line 7
    cout << "Line 8: The numbers you entered are:"
         << endl; //Line 8
    cout << "Line 9: a = " << a << ", b = " << b
         << ", c = " << c << ", d = " << d << endl; //Line 9
    return 0;
}
```

```
D:\Object Oriented Language\OOP Lab- 06\InputFailure.exe
Line 5: Enter four integers: 37 653.89 23 76

Line 8: The numbers you entered are:
Line 9: a = 37, b = 653, c = 0, d = 40
```

In this sample run, the cin statement in Line 6 inputs 37 into a and 653 into b and then tries to input the decimal point into c. Because c is an int variable, the decimal point is regarded as a character, so the input stream enters the fail state. In this sample run, the values of c and d are unchanged, as shown by the output of the statement in Line 9.

The clear Function

- When an input stream enters the fail state, the system ignores all further I/O using that stream.
- You can use the stream function clear to restore the input stream to a working state.
- The syntax to use the function clear is:

```
istreamVar.clear();
```

Here, istreamVar is an input stream variable, such as cin. After using the function clear to return the input stream to a working state, you still need to clear the rest of the garbage from the input stream. This can be accomplished by using the function ignore.

Lab06`.cpp | [*] UnformattedExample.cpp | FormattedExample.cpp | CinWithExtractoreOperator.cpp | predefinedFunction0.cpp | CinWithIgnore.cpp | peekandputback.cpp | InputFailure.cpp | ClearFunction.cpp

```cpp
//Input failure and the clear function
#include <iostream>
using namespace std;
int main() {
    int a = 23; //Line 1
    int b = 34; //Line 2
    cout << "Line 3: Enter a number followed"
         << " by a character: "; //Line 3
    cin >> a >> b; //Line 4
    cout << endl << "Line 5: a = " << a
         << ", b = " << b << endl; //Line 5
    cin.clear(); //Restore input stream; Line 6
    cin.ignore(200,'\n'); //Clear the buffer; Li
    cout << "Line 8: Enter two numbers: "; //Lir
    cin >> a >> b; //Line 9
    cout << endl << "Line 10: a = " << a
         << ", b = " << b << endl; //Line 10
    return 0;
}
```

```
D:\Object Oriented Language\OOP Lab- 06\ClearFunction.exe
Line 3: Enter a number followed by a character: 78 d

Line 5: a = 78, b = 0
Line 8: Enter two numbers: 65 88

Line 10: a = 65, b = 88

--------------------------------
Process exited after 66.57 seconds with return value 0
Press any key to continue . . .
```

Output and Formatting Output

Other than writing efficient programs, generating the desired output is one of a programmer's highest priorities.

- setprecision & fiexed Manipulator

  - You use the manipulator setprecision to control the output of floating-point numbers.
  - To print floating-point output to two decimal places, you use the setprecision manipulator to set the precision to 2
  - To use the manipulator setprecision, the program must include the header file iomanip. Thus, the following include statement is required:
  - #include <iomanip>

The general syntax of the **setprecision** manipulator is:

```
setprecision(n)
```

where **n** is the number of decimal places.

cout << fixed;
The following statement sets the output of floating-point numbers in a fixed decimal format on the standard output device:

```
1  //setprecision Manipulator
2  #include <iostream>
3  #include <iomanip>
4  using namespace std;
5  int main() {
6      double num1 = 3.12345678;
7      cout<<fixed<<setprecision(3)<<num1; ///for precision upto 3 digit
8      return 0;
9  }
```

```
D:\Object Oriented Language\OOP Lab- 06\setprecisionManipulator.exe
3.123
--------------------------------
```

Output and Formatting Output

```cpp
//Example: setprecision, fixed, showpoint
#include <iostream> //Line 1
#include <iomanip> //Line 2
using namespace std; //Line 3
const double PI = 3.14159265; //Line 4
int main() { //Line 5
    //Line 6
    double radius = 12.67; //Line 7
    double height = 12.00; //Line 8
    cout << fixed << showpoint; //Line 9
    cout << setprecision(2)
         << "Line 10: setprecision(2)" << endl; //Line 10
    cout << "Line 11: radius = " << radius << endl; //Line 11
    cout << "Line 12: height = " << height << endl; //Line 12
    cout << "Line 13: volume = "
         << PI * radius * radius * height << endl; //Line 13
    cout << "Line 14: PI = " << PI << endl << endl; //Line 14
    cout << setprecision(3)
         << "Line 15: setprecision(3)" << endl; //Line 15
    cout << "Line 16: radius = " << radius << endl; //Line 16
    cout << "Line 17: height = " << height << endl; //Line 17
    cout << "Line 18: volume = "
         << PI * radius * radius * height << endl; //Line 18
    cout << "Line 19: PI = " << PI << endl << endl; //Line 19
    cout << setprecision(4)
         << "Line 20: setprecision(4)" << endl; //Line 20
    cout << "Line 21: radius = " << radius << endl; //Line 21
    cout << "Line 22: height = " << height << endl; //Line 22
    cout << "Line 23: volume = "
                 << PI * radius * radius * height << endl; //Line 23
    cout << "Line 24: PI = " << PI << endl << endl; //Line 24
    cout << "Line 25: "
         << setprecision(3) << radius << ", "
         << setprecision(2) << height << ", "
         << setprecision(5) << PI << endl; //Line 25
    return 0; //Line 26
} //Line 2
```

```
D:\Object Oriented Language\OOP Lab- 06\showpointManipulator.exe
Line 10: setprecision(2)
Line 11: radius = 12.67
Line 12: height = 12.00
Line 13: volume = 6051.80
Line 14: PI = 3.14

Line 15: setprecision(3)
Line 16: radius = 12.670
Line 17: height = 12.000
Line 18: volume = 6051.797
Line 19: PI = 3.142

Line 20: setprecision(4)
Line 21: radius = 12.6700
Line 22: height = 12.0000
Line 23: volume = 6051.7969
Line 24: PI = 3.1416

Line 25: 12.670, 12.00, 3.14159

--------------------------------
```

THANKS ☺