- All extractors skip leading white spaces

- Integral extractors
  - Read input characters until any that cannot be part of the type
  - 123x456 is read as 123 with input pointer at x

```
int var ;   cin>>var ; // enter data 123x456
cout<<var ;
```

- Floating point extractors
  - Read input characters until any that cannot be part of the type
  - 123e-4x5 is read as 123.0e-4 with input pointer at x

- Character extractor
  - Reads the text character in the input stream skipping leading white spaces
  - char * or string extractors read all input characters, skipping leading white spaces, upto the next white space
  - “ This string” is read as “This”

- Multiple data items can be read in a single statement by chaining operators together.

Example :

```
int main(int) {
  char c ; int i ; float f;
  char * buf[30];
  cin >> c >> i >> f ;
  }
```

```
int main(int) {
int     i ; char   c ; double     d ;

cin>>i>>c>>d ;
cout<<i<<endl ;
cout<<c<<endl ;
cout<<d<<endl ;
}
```

INPUT : 23x1.2

OUTPUT :

23

X

1.2

I/O Formatting

Special methods are defined for formatting stream objects.

- The ***width*** method specifies I/O width
  - For cin , only the specified numbers of characters are read
  - For cout , the output is displayed right justified in a field of the specified width
  - If the length of input is greater than the current width , the entire value is displayed for cout

Example :

```
char buf[21];
cin.width(20);  //only 20 characters read at a time
cin >> buf ;
int x = 1;
cout.width(5);
cout << x ;  /*x is displayed right justified in a field five characters long */
```

- The ***fill*** method sets the character used for padding extra space when calling *width*
  - ➢ <space> is default fill character

  - ➢ Invoked only if the value set using width is greater than the length of the inserted value

Example :

```
int main ( ) {
    int x = 10 ;
     cout.fill('#');
     cout.width(5);
     cout<<x ;
}
```

Output : ###10

Other input streams function

- ***tie(ostream)***
  - ➢ Attaches an output stream to an input stream
  - ➢ Output stream is flushed before extractions on the input stream

```
cin.tie(cout) ;  // flushes cout before cin
cin.tie(0) ;     // unties cin from cout
```

- ***ignore(int i)***
  - ➢ Unconditionally ignores the next i characters for cin

```
cin.ignore(5);
```

- ***peek( )***
  - ➢ The next character in the input stream can be looked at without actually fetching i

```
char ch = cin.peek()
```

❖ ***putback(char ch)***

- ➢ Returns the last character that has been fetched to the input buffer
- ➢ An error may occur if the stream cannot accept the putback character

❖ ***get( )***

- ➢ Reads the next character from the input stream

  char ch = cin.get( );
- ➢ get() is overloaded with other version

❖ ***get(char *str, int len, char delim = '\n')***

- ➢ Fetches characters from the input stream into the array str
- ➢ fetching is stopped if the len characters have been fetched
- ➢ Fetching is also stopped if delim character is encountered
- ➢ Next read occurs at the delimiter

- ***get(char &ch)***
  - ➢ Fetches the next character in the stream and stores it in ch

- ***getline(char *str, int len, char delim = '\n')***
  - ➢ similar to get(char *, int, char)
  - ➢ Extracts the terminator also
  - ➢ Next read occurs after the delimiter

- ***put(char ch)***
  - ➢ A single character is written to an output stream without translation

- *cerr* is a predefined error output stream attached to the standard error device.

Inserters

- *cout* is a predefined output stream attached to the standard output device

- << sends character to the output stream

- The text string on the right is stored in the left stream

```
#include <iostream>
int main()
  {    cout << "Hello World ! "}
```

- Same precedence as the standard left shift operator

```
cout << x + y << "\n" ; // OK
cout << x & y << "\n" ; //Error
  /* & has lower precedence than << */
cout << (x & y) << "\n" ; // OK
```