Object Oriented Programming

Unit 6:
Streams I/O Operations

Streams I/O Operations

CE: 6.0. Introduction

- Every program follows input-process-output cycle.
- C++ supports a rich set of I/O functions and operation to do this.
- C++ supports the concepts ***streams*** and ***stream classes*** to implement its I/O operations with the console and disk file.
- In C++ input and output is performed in the form of ***sequence of bytes*** which is commonly known as ***streams***.

CE: 6.0. Introduction (Conti…)

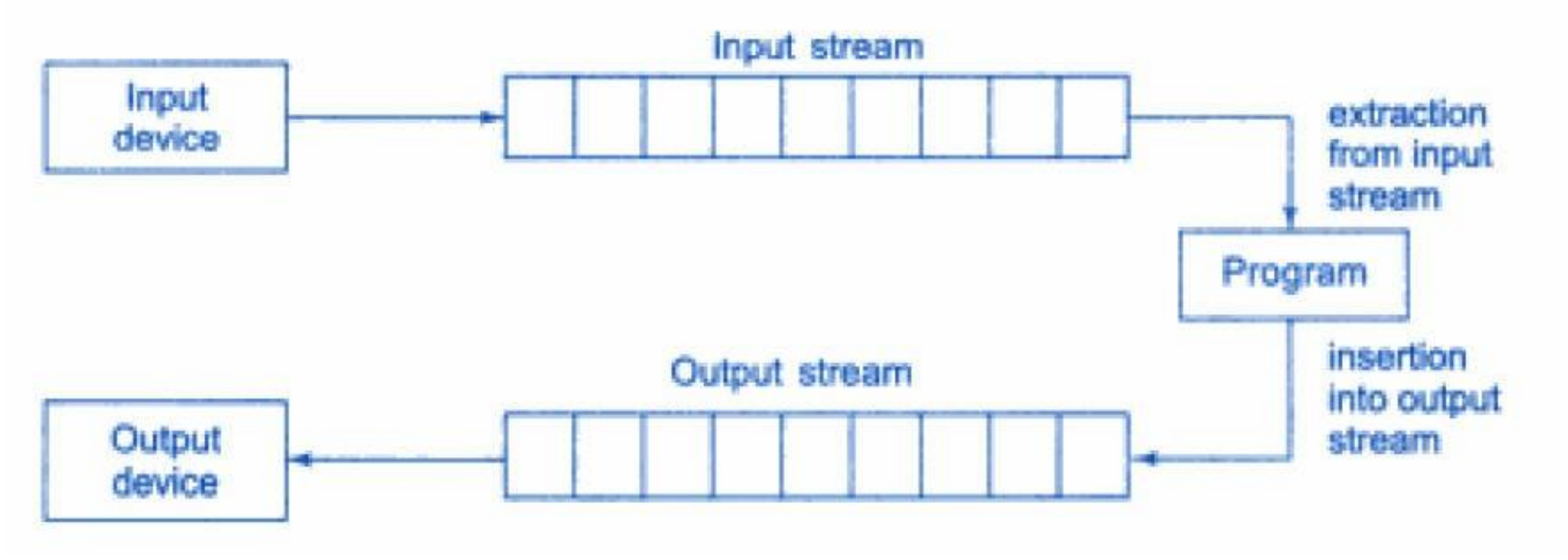
What are C++ stream?

- A ***stream*** is a sequence of bytes. 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* provides data to the program is called the ***input stream***.

- The *destination stream* that receives output for the program is called ***output stream***.

CE: 6.0. Introduction (Conti…)



- The data in the input stream can come from the keyboard or any other storage device.
- The data in the output stream go to the screen or any other storage device.

CE: 6.0. Introduction (Conti…)

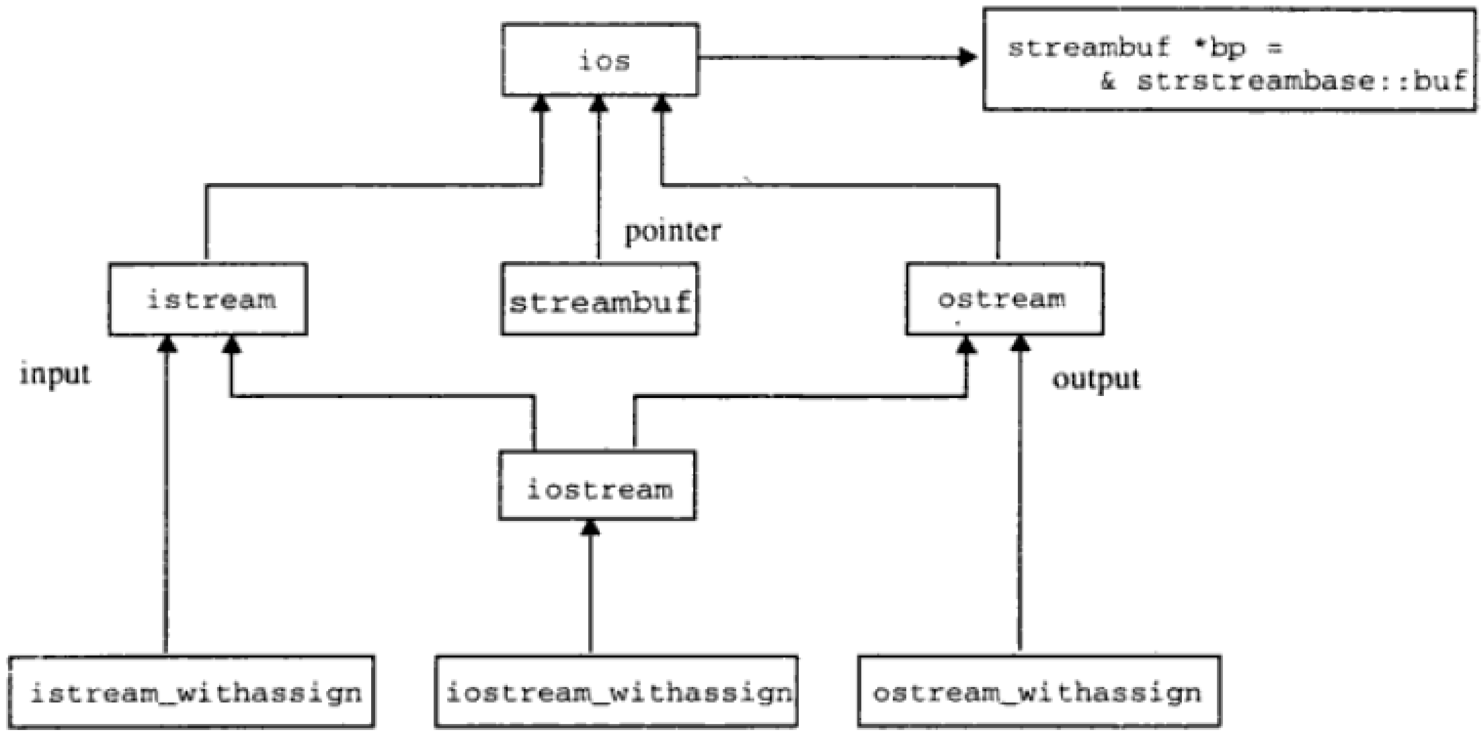
- C++ contains several pre-defined streams that are automatically opened when a program begin its execution.

- These includes **cin** and **cout** which have been used very often; and also **cerr** and **clog**.

  - ✓ **cin** represent the input stream connected to the standard input device.
  - ✓ **cout** represent the output stream connected to the standard output device.
  - ✓ **cerr** represent standard error output.
  - ✓ **clog** represent a fully-buffered version of **cerr**.

CE: 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 file. These classes are called ***stream classes***.



CE: C++ Stream Classes (Conti…)

- **ios** is the base class for **istream**(input stream) and **ostream**(output stream), which are, in turn of base classes for iostream (input/output stream).

- The class **ios** is declared as **virtual base class,** so that only one copy of its member are inherited by the iostream.

- The class **ios** provide the basic support for **formatted** and **unformatted** I/O operation.

- The class **istream** provides the facilities for formatted and unformatted input,
- while the class **ostream** provides the facilities for formatted output.
- and the class **iostream** provides the facilities for handling both input and output streams.

CE: C++ Stream Classes (Conti…)

- Three classes, namely, istream_withassign, ostream_withassign, and iostream_withassign add assignment operator to these classes.

CE: C++ Stream Classes (Conti…)

1. ios (input/output stream class)

- Contains basic facilities that are used by all other input and output classes.
- Also contains a pointer to a buffer object (streambuf object)
- Declares constant and functions that are necessary for handling formatted input and output operations.

2. **istream (input stream)**

- inherited the properties of ios
- Declares input function such as get(), getline() and read()
- Contains overloaded extraction operator>>

3. ostream(output stream)

- Inherit the properties of ios
- Declares output function put() and write()
- Contains overloaded insertion operator <<.

CE: C++ Stream Classes (Conti…)

4. iostream (input/output stream)

- Inherits the properties of ios. istream and ostream through multiple inheritance and thus contains all the input and output functions.

5. streambuf

- Provides an interface to physical device through buffers.
- Acts as a base of filebuf class used ios files.

CE: 6.1
Managing Console I/O Operations

CE: 6.1. Managing Console I/O Operations

- Console I/O Operations can be managed in two ways:
  1. Unformatted I/O Operations
  2. Formatted I/O Operations

CE: 6.1.1. Unformatted I/O Operations

- The stream classes of C++ support two member functions, to handle the single character input/output operations:
  1. get()
  2. put()

1. get():

- It is member function of the input stream class istream.
- It used to read a single character from the input device.
- There are two types of get functions:
  1. get(char &)
  2. get(void)

- Both the functions area used to fetch a character including the blank space, tab and newline character.
- The get(char &) version assign the input character to its argument and the get(void) version return the input character.

CE: 6.1.1. Unformatted I/O Operations (Conti…)

- It well known that, the member functions are invoked by their objects using dot operators.

- These two functions can be used to perform input operation either by using the predefined object **cin** or an user defined object of the **istream** class.

- For Example:

```
char c;
cin.get(c); // It is the replacement of cin>>c;
```

CE: 6.1.1. Unformatted I/O Operations (Conti…)

```
#include<iostream>
using namespace std;

int main()
   {
          char c;
          cout<<"Enter the your characters: ";
          cin.get(c); // get a character from keyboard and assign to c
          while(c  != '\n')
             {
                    cout<<c;  // display the character on screen
                    cin.get(c); // get another character
              }
   }
```

CE: 6.1.1. Unformatted I/O Operations (Conti…)

- The operator >> can also be used to read a character, but it will skip the white spaces and newline character.

- The above while loop will not work properly, if the statement cin>>c;

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

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

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

CE: 6.1.1. Unformatted I/O Operations (Conti…)

```
#include<iostream>
using namespace std;

int main()
   {
        char c;
        cout<<"Enter the your characters: ";
        c = cin.get(); // get a character from keyboard and assign to c
        while(c  != '\n')
           {
                cout<<c;  // display the character on screen
                c = cin.get(); // get another character
            }
   }
```

CE: 6.1.1. Unformatted I/O Operations (Conti…)

2. put():

- It is member function of the output stream class ostream.
- It is used to write a single character (character by character) from the output device.
- It prints a character representation of the input parameter.

- For Example:

  cout.put('c'); // It display the character and the replacement of cout<<"c";

  cout.put(c); // It displays the value of variable and the replacement of cout<<c;

  cout.put(68); // int value 68 convert into char and display ASCII

CE: 6.1.1. Unformatted I/O Operations (Conti…)

```
#include<iostream>
using namespace std;

int main()
   {
          char c;
          cout<<"Enter the your characters: ";
          c = cin.get(); // get a character from keyboard and assign to c
          while(c  != '\n')
             {
                    cout.put(c);  // display the character on screen
                    c = cin.get(); // get another character
              }
   }
```

CE: 6.1.1. Unformatted I/O Operations (Conti…)

```
#include<iostream>
using namespace std;

int main()
   {
          char c=65;
          cout.put(c); // print the value of c
          cout.put(66);
          cout.put('c');
   }
```

Class Work

- Write a program that accept character and display its ASCII value.

CE: 6.1.1. Unformatted I/O Operations (Conti…)

getline():

- It is used to read a line of text more effectively using the line-oriented input function getline().

- 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, len);

- The reading is terminated as soon as the newline character “\n” is encountered or len-1 character are read.

- The newline character is read, but not saved into a buffer. Instead, it is replaced by the **null** character.

CE: 6.1.1. Unformatted I/O Operations (Conti…)

getline(): (Conti…)

- For Example:

```
char name[20];
cin.getline(name, 20);
```

```
Input: Bajrane Stroustrup
Output: Bajrane Stroustrup

Input: Object Oriented Programming
Output: Object Oriented Pro
```

CE: 6.1.1. Unformatted I/O Operations (Conti…)

write():

- It is used to write a line of text more effectively using the line-oriented output function write().

- The write() function display an entire line by using the following form:

cout.write(line, len);

- ✓ line - represents the name of the string to be displayed.
- ✓ len - indicate the number of character to display.

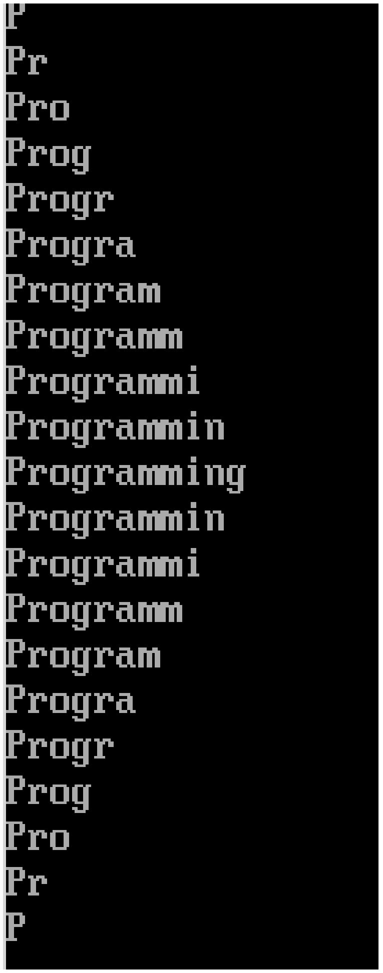
CE: 6.1.1. Unformatted I/O Operations (Conti…)

```
#include<iostream>
using namespace std;

int main()
  {
          char name[20];
          cout<<"Enter your name: ";
          cin.getline(name, 20);
          cout<<name<<"\n";
          cout.write(name, 10);
  }
```

Class Work

- Write a C++ program that print the following pattern, if a string is “Programming” using write().



```
P
Pr
Pro
Prog
Progr
Progra
Program
Programm
Programmi
Programmin
Programming
Programmin
Programmi
Programm
Program
Progra
Progr
Prog
Pro
Pr
P
```

Class Work

- Write a C++ program that accept two different string and concatenate both the string.

CE: 6.1.2. Formatted I/O Operations

- C++ support a number of feature that could be used for formatting the output.

- These feature include:
  1. ios stream class member function and flags
  2. Standard Manipulators
  3. User-defined Manipulators (output functions)

- The ios stream class contains a large number of member functions that would help us to format the output in a number of ways.

CE: 6.1.2. Formatted I/O Operations (Conti…)

- The most important among these functions are:
  1. width()
  2. precision()
  3. fill()
  4. setf()
  5. unsetf()

CE: 6.1.2. Formatted I/O Operations (Conti…)

1. width():

- It is used to define the width of the field to be used while displaying the output value.
- It must be accessed using objects of the ios class i.e. cout.

- Syntax:

  cout.width(w);

  - where **w** is the field width (number of columns)

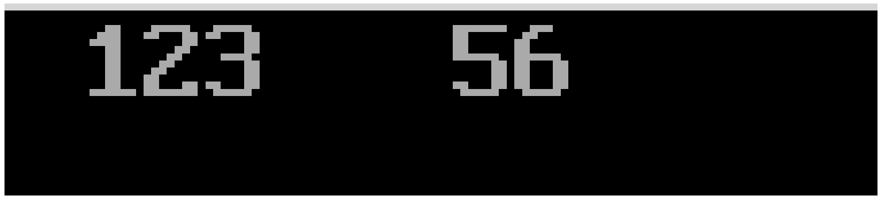
- The width() can specify the field width for only one item.
- After printing one item it will return back to the default.

CE: 6.1.2. Formatted I/O Operations (Conti…)

```
#include<iostream>
using namespace std;

int main()
  {
        cout.width(5);
        cout<<123;
        cout.width(5);
        cout<<56;
  }
```

		1	2	3				5	6



```
  123   56
```

CE: 6.1.2. Formatted I/O Operations (Conti…)

2. precision():

- It is used to specify the number of digits to be displayed after the decimal point while printing a floating-point number.
- By default, it size is six.
- It must be accessed using objects of the ios class i.e. cout.

- Syntax:

  cout.precision(d);

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

- It sets the floating-point precision and returns the previous setting.

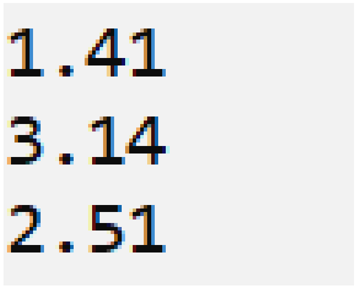
- For Example:

```
cout.precision(2);
cout<<2.23;
```

CE: 6.1.2. Formatted I/O Operations (Conti…)

```
#include<iostream>
#include<math.h>
using namespace std;

int main()
  {
         cout.precision(3);
         cout<<sqrt(2)<<"\n";
         cout<<3.14235 <<"\n";
         cout<<2.50563<<"\n";
  }
```



```
1.41
3.14
2.51
```

CE: 6.1.2. Formatted I/O Operations (Conti…)

3. fill():

- It is used to specify the character to be displayed in the unused portion of the display width.
- By default, blank character is displayed in the unused portion if the display width is larger than that required by the value.
- Syntax:

  cout.fill(ch);

  - where **ch** is the character to be filled in the unused portion.
- For Example:

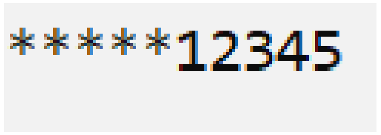
```
cout.fill(‘*’);
cout.width(10);
cout<<12345<<"\n";
```

CE: 6.1.2. Formatted I/O Operations (Conti…)

```
#include<iostream>
using namespace std;

int main()
  {
         cout.fill('*');
         cout.width(10);
         cout<<12345<<"\n";
  }
```

*	*	*	*	*	1	2	3	4	5



```
*****12345
```

Class Work

- Write a C++ program that print following output:

$	$	5	4	3	@	@	@	1	2

CE: 6.1.2. Formatted I/O Operations (Conti…)

4. setf(): (set flags)

- It can be noted that, by using width(), the results are printed in the right-justified form (but it is not usual practice).

  How do we get a value printed left-justified?

- It used to set flags and bit-fields that control the output.

- Syntax:

  cout.setf(setbits, bitfield);

  - where, **setbits** is one of the flags defined in the class ios. It specifies the format action required for the output.
  - **bitfields** specifies the group to which the formatting flag belongs.
  - Both the forms return the previous settings.

CE: 6.1.2. Formatted I/O Operations (Conti…)

4. setf(): (Conti…)

Flags value	Bit field	Effect produced
ios::left	ios::adjustfield	Left-justified output
ios::right	ios::adjustfield	Right-adjust
ios::internal	ios::adjustfield	The output is expanded to the field width (width) by inserting fill characters (fill) at a specified internal point, the numerical values is between the sign and/or numerical base and the number size. For non-numerical values it is equivalent to right.
ios::dec	ios::basefield	Decimal conversion
ios::oct	ios::basefield	Octal conversion
ios::hex	ios::basefield	Hexadecimal conversion
ios::scientific	ios::floatfield	Use exponential floating notation
ios::fixed	ios::floatfield	Use ordinary floating notation

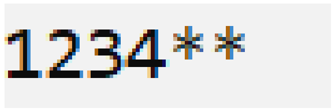
Flags and bit fields for setf function

CE: 6.1.2. Formatted I/O Operations (Conti…)

```
#include<iostream>
using namespace std;

int main()
  {
          cout.setf(ios::left, ios::adjustfield);
          cout.fill('*');
          cout.precision(3);
          cout.width(6);
          cout<<1234;
  }
```

1	2	3	4	*	*

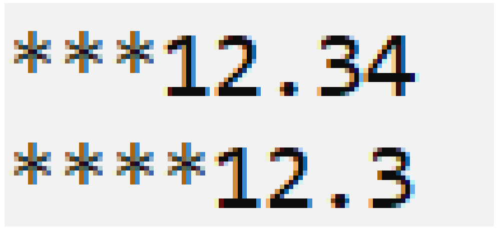


```
1234**
```

CE: 6.1.2. Formatted I/O Operations (Conti…)

```
#include<iostream>
using namespace std;
int main()
  {
        cout.setf(ios::internal, ios::adjustfield);
        cout.fill('*');
        cout.precision(4);
        cout.width(8);
        cout<<12.34;

        cout<<"\n";
        cout.precision(3);
        cout.width(8);
        cout<<12.34;
  }
```



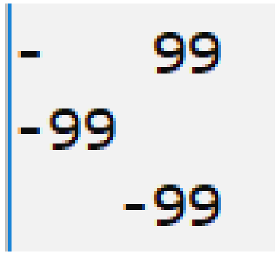
```
***12.34
****12.3
```

CE: 6.1.2. Formatted I/O Operations (Conti…)

```
#include<iostream>
using namespace std;
int main()
   {
         int n = -99;
         cout.setf(ios::internal, ios::adjustfield);
         cout.width(6);
         cout<<n<<'\n';

         cout.setf(ios::left, ios::adjustfield);
         cout.width(6);
         cout<<n<<'\n';

         cout.setf(ios::right, ios::adjustfield);
         cout.width(6);
         cout<<n<<'\n';
   }
```



```
-   99
-99
   -99
```

CE: 6.1.2. Formatted I/O Operations (Conti…)

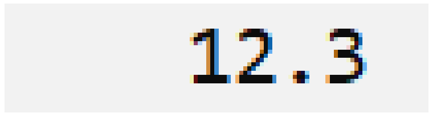
Displaying Trailing Zeros and Plus sign:

- Streams support the feature of avoiding truncation of the trailing zero in the output.

- For Example:

```
#include<iostream>
using namespace std;

int main()
 {
        cout.setf(ios::internal, ios::adjustfield);
        cout.precision(4);
        cout.width(8);
        cout<<12.30;
 }
```



```
    12.3
```

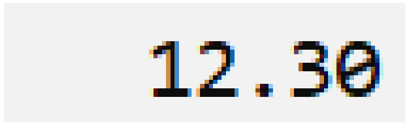
CE: 6.1.2. Formatted I/O Operations (Conti…)

Displaying Trailing Zeros and Plus sign: (Conti…)

- The ios class has the flag, **showpoint** which when set, prints the trailing zero also.

- For Example:

```
#include<iostream>
using namespace std;
int main()
  {
        cout.setf(ios::showpoint);
        cout.setf(ios::internal, ios::adjustfield);
        cout.precision(4);
        cout.width(8);
        cout<<12.30;
  }
```



```
   12.30
```

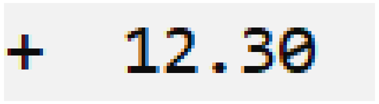
CE: 6.1.2. Formatted I/O Operations (Conti…)

Displaying Trailing Zeros and Plus sign: (Conti…)

- For Example:

```
#include<iostream>
using namespace std;

int main()
  {
        cout.setf(ios::showpos);  // For positive sign
        cout.setf(ios::showpoint);
        cout.setf(ios::internal, ios::adjustfield);
        cout.precision(4);
        cout.width(8);
        cout<<12.30;
  }
```



```
+   12.30
```

CE: 6.1.2. Formatted I/O Operations (Conti…)

More flags that do not have bit fields for the setf() are:

Flags value	Effect produced
ios::showbase	Use base indicator on output
ios::showpos	Add '+' to positive integers
ios::showpoint	Include decimal point and trailing zeros in output
ios::uppercase	Uppercase hex output
ios::skipws	Skips white-space characters on input.
ios::unitbuf	Flush after insertion (i.e. use a buffer of size 1)
ios::stdio	Flush stdout and stderr after insertion

- The flag setting ios::skipws is set by default.
- The white-space characters are space, tab, newline, carriage return, form feed and vertical tab.

CE: 6.1.2. Formatted I/O Operations (Conti…)

5. unsetf():

- It is used to reset the flag.

- For Example:

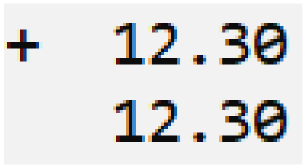
  cout.unsetf(ios::showpos);

  - It clears the bits corresponding to show positive-sign symbol and returns the previous settings.

CE: 6.1.2. Formatted I/O Operations (Conti…)

```
#include<iostream>
using namespace std;
int main()
  {
        cout.setf(ios::showpos);
        cout.setf(ios::showpoint);
        cout.setf(ios::internal, ios::adjustfield);
        cout.precision(4);
        cout.width(8);
        cout<<12.30;
        cout.unsetf(ios::showpos);

        cout<<"\n";
        cout.width(8);
        cout<<12.30;
  }
```



```
+   12.30
    12.30
```

CE: 6.2
Hierarchy of File Stream Classes

CE: Program-console and file interaction

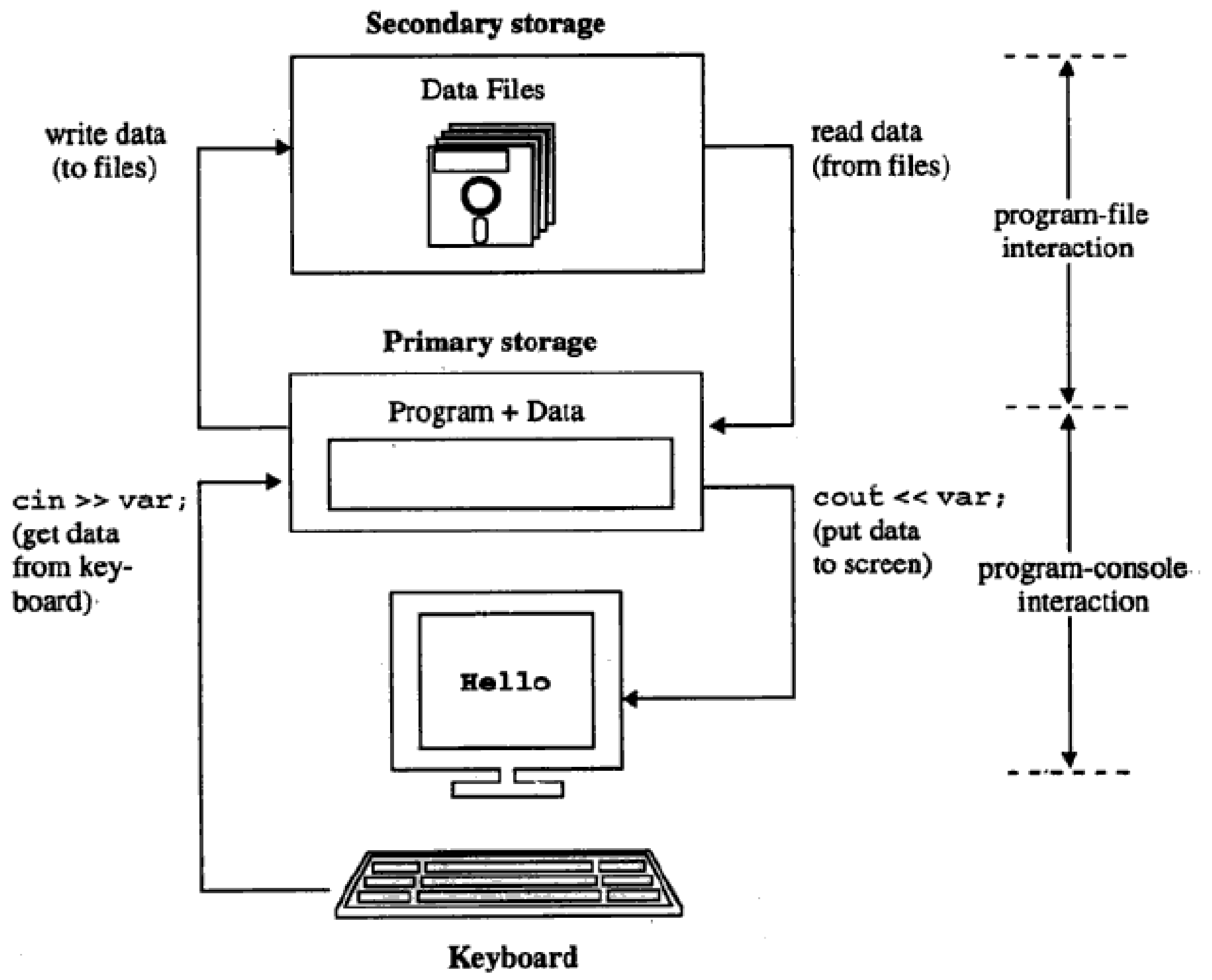


Figure 18.1: Program-console and file interaction

CE: File Input and Output Classes

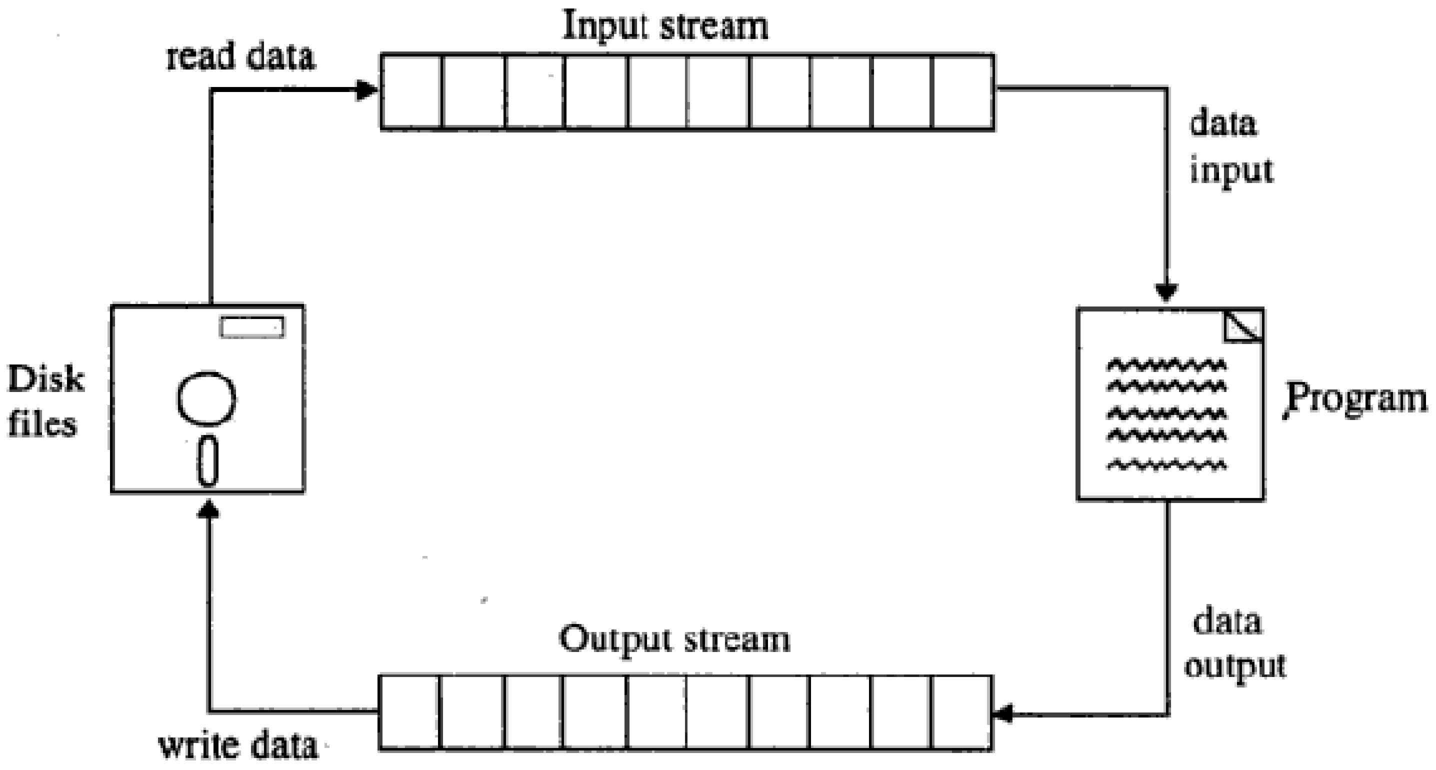
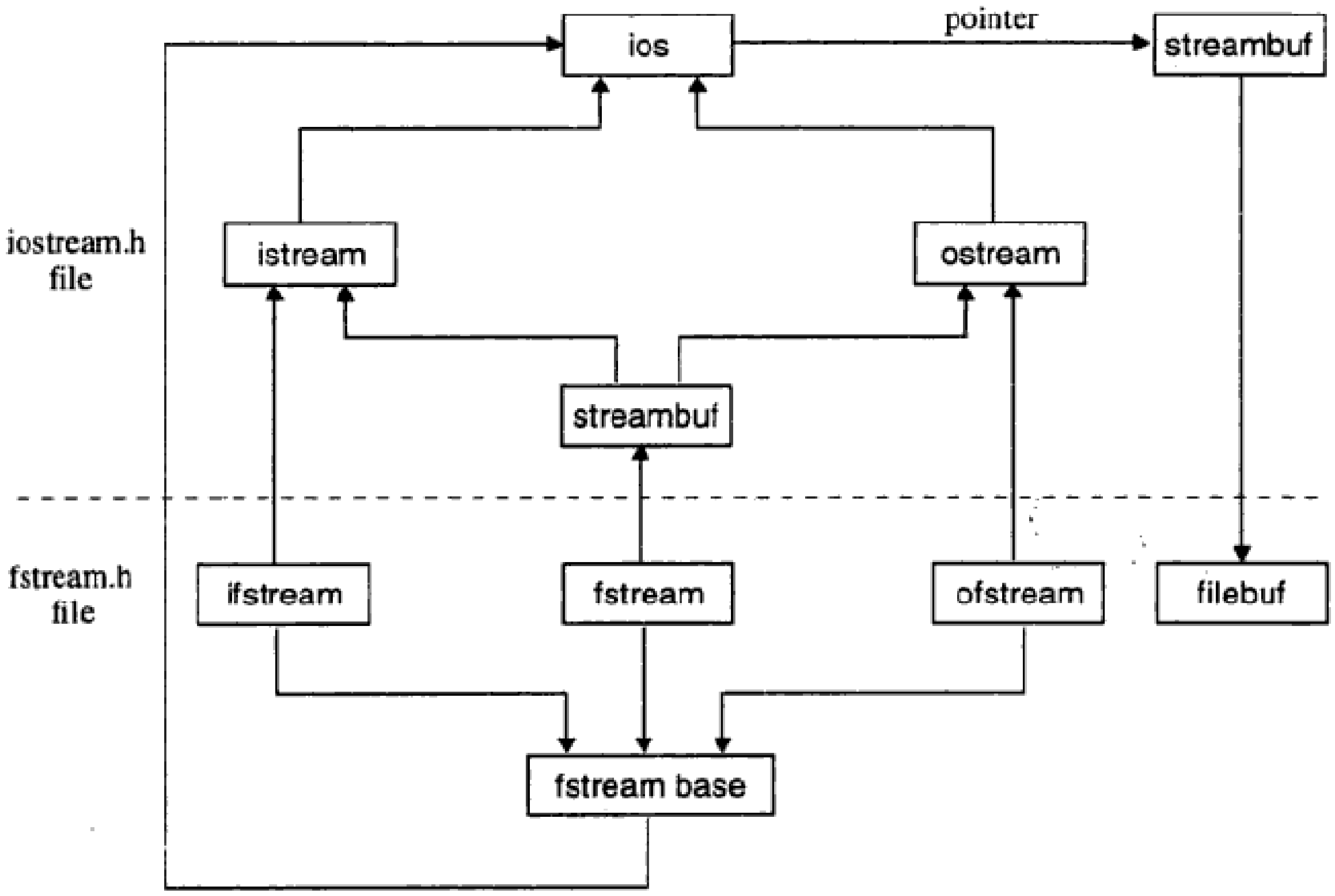


Figure 18.2: File input and output streams

CE: 6.2. Hierarchy of File Stream Classes



CE: 6.2. Hierarchy of File Stream Classes (Conti…)

- The file handling techniques of C++ support file manipulation in the form of stream objects.

- There are three classes for handling files:
  1. ifstream – for handling input files
  2. ofstream – for handling output files
  3. fstream – for handling files on which both input and output can be performed.

- These classes are designed exclusively to manage the disk files; and their declaration exists in the header file fstream.h.

CE: 6.2. Hierarchy of File Stream Classes (Conti…)

- The actions performed by classes related to file management are:

1. filebuf:

- It sets the file buffers to read and write.
- It contains **close()** and **open()** member functions in it.

2. fstreambase:

- This is the base class for **fstream**, **ifstream** and **ofstream** classes.
- Therefore, it provides operations common to these file stream.
- It also contains **open()** and **close()** functions.

3. ifstream:

- Being an input file stream class, it provides input operations for file.
- It inherits the functions **get()**, **getline()**, **read()** and functions supporting random access (**seekg()** and **tellg()**) from istream class defined inside iostream.h file.

CE: 6.2. Hierarchy of File Stream Classes (Conti…)

4. ofstream:

- Being an output file stream class, it provides output operations.
- It inherits **put()**, and **write()** functions along with functions supporting random access (**seekp()** and **tellp()**) from ostream class defined inside iostream.h file.

5. fstream:

- It is an input-output file stream class.
- It provides support for immediate input and output operations.
- It inherits all the functions from **istream** and **ostream** classes through iostream class defined inside iostream.h.

CE: 6.3
Opening And Closing a File

CE: 6.4
Reading And Writing a File

CE: 6.3. Opening And Closing a File

- In order to process a file, first it must be opened to get a handle.
- The file handle serves as a pointer to the file.
- The following steps are involved in using a file, in a C++ program:
  1. Name the file on the disk
  2. Open the file to get the file pointer
  3. Process the file(read/write)
  4. Check for error while processing
  5. Close the file after its complete usage

CE: 6.3. Opening And Closing a File (Conti…)

- **Name the file on the disk:**
- The file name is a string of character that makes up a valid filename for the operating system.
- It may contain two parts, a *primary name* and an *optional period with extension*.

- For Example:
  - ✓Student.cpp
  - ✓Employee.txt
  - ✓Data.exe
  - ✓result

CE: 6.3. Opening And Closing a File (Conti…)

- Opening of files can be achieved in two ways:
  1. Using the constructor function of the stream class.
  2. Using the function open().

- The first method is useful when a single file is used with a stream.
- The second method is used when we want to manage multiple files using same stream.

CE: 6.3. Opening And Closing a File (Conti…)

1. Opening files using constructors:

- To open a file, “datafile” as an input file, we shall create a file stream object of input type. i.e. ifstream type.

- For Example:

```
ifstream file(“test.txt”);
char ch;
file >> ch;
float amount;
file << amount;
```

- Opens the file test.txt for input.
- A constructor is used to initialize an object during its creation.
- Hence the constructor can be utilized to initialize the filename to be used with the file stream object.

CE: 6.3. Opening And Closing a File (Conti…)

2. Opening files using Open() Function:

- For Example:

  ofstream fout; // create an input stream
  ……
  fout.open("student.dat"); //associate fin stream with the file student.dat
  fout.close(); // disconnect stream from student.dat
  …
  fout.open("marks.dat"); //associate fin stream with the file marks.dat

CE: 6.3. Opening And Closing a File (Conti…)

2. Opening files using Open() Function: (Conti…)

- C++ provides a mechanism of opening a file in different modes in which case the second parameters must be explicitly passed.

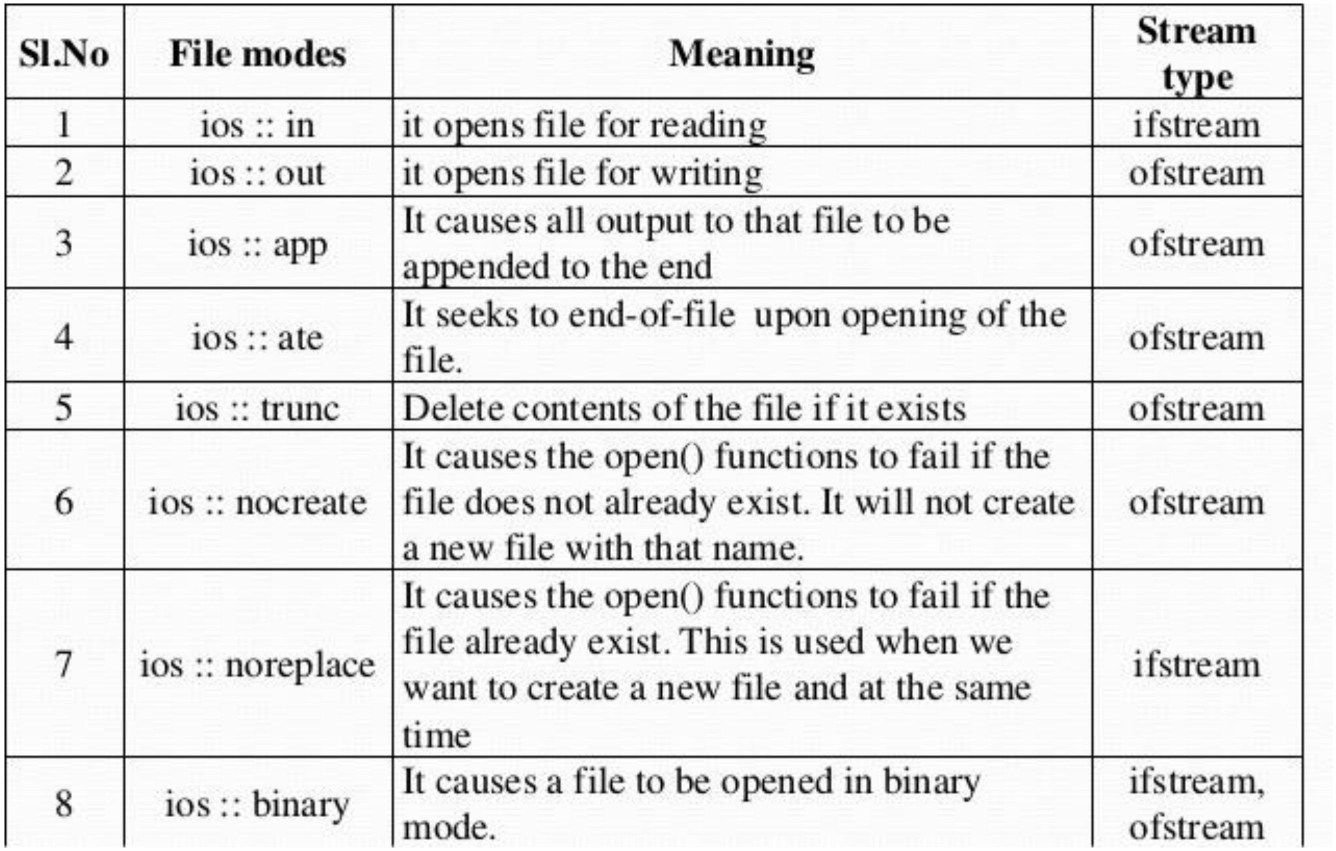
- Syntax:

  stream-object.open("filename", mode);

  - It opens the file in the specified mode.

CE: 6.3. Opening And Closing a File (Conti…)

File Open Modes:



Sl.No	File modes	Meaning	Stream type
1	ios :: in	it opens file for reading	ifstream
2	ios :: out	it opens file for writing	ofstream
3	ios :: app	It causes all output to that file to be appended to the end	ofstream
4	ios :: ate	It seeks to end-of-file upon opening of the file.	ofstream
5	ios :: trunc	Delete contents of the file if it exists	ofstream
6	ios :: nocreate	It causes the open() functions to fail if the file does not already exist. It will not create a new file with that name.	ofstream
7	ios :: noreplace	It causes the open() functions to fail if the file already exist. This is used when we want to create a new file and at the same time	ifstream
8	ios :: binary	It causes a file to be opened in binary mode.	ifstream, ofstream

CE: 6.3. Opening And Closing a File (Conti…)

- For Example:

  Write a C++ program to store enro, name and percentage of a student into a file called “student.txt”.

Program to write in file.

```
#include<iostream>
#include<fstream>
using namespace std;

int main()
  {
        ofstream fout;
        fout.open("student.txt");

        int enro;
        char name[30];
        float per;

        cout<<"Enter your enrollment number: ";
        cin>>enro;

        cout<<"Enter your name: ";
        cin>>name;

        cout<<"Enter your percentage: ";
        cin>>per;

        fout<<enro;
        fout<<name;
        fout<<per;

        fout.close();
        return 0;
  }
```

CE: 6.4. Reading And Writing a File

Detecting End Of File:

- While reading a file, a situation can arise, when we do not know the number of objects to be read from the file i.e. we do not know where the file is going to end?

- A simple method of detecting end of file (eof) is by testing the stream in a while loop as shown below:

```
while (<stream>)
 {
       ….
 }
```

- The condition <stream> will evaluate to 1 as long as the end of file is not reached; and it will return 0 as soon as end of file is detected.

CE: 6.4. Reading And Writing a File (Conti…)

Detecting End Of File: (Conti…)

- You can detect when end of the file is reached by using the member function eof() which has the prototype…int eof();

- For Example:

```
  while(! find.eof())  // as long as eof() is zero, i.e. the file's end is not
                                        reached process the file
       {

       }
  if(fin.eof())  // if nonzero
     {
             cout<<"End of the file reached!";
     }
```

Program to read from file.

```
#include<iostream>
#include<fstream>
using namespace std;

int main()
  {
          const int n = 80;
          char line[n];
          fstream fin;
          fin.open("country.txt", ios::in);

           if(!fin)
             {
                      cout<<"Cannot open file! \n";
                      return 1;
              }
```

```
          while(fin)
             {
                      fin.getline(line, n);
                      cout<<line;
             }
           fin.close();
           return 0;
    }
```

Program to append data in existing file.

```
#include<iostream>
#include<string>
#include<fstream>
using namespace std;

int main()
   {
         ifstream fin;
         ofstream fout;  // Create Object of Ofstream

         fin.open("student.txt");
         fout.open ("student.txt", ios::out|ios::app); // Append mode

         int enro;
         char name[30];
         float per;

         cout<<"Enter your enrollment number: ";
         cin>>enro;
```

Program to append data in existing file. (Conti…)

Program (Conti…)

```
    cout<<"Enter your name: ";
    cin>>name;

    cout<<"Enter your percentage: ";
    cin>>per;

    if(fin.is_open())
      {
          fout<<enro;    // Writing data to file
          fout<<name;
          fout<<per;
      }
    cout<<"\n Data has been appended to file!";

    fin.close();
    fout.close(); // Closing the file
    return 0;
}
```

Class Work

1. Write a C++ program to store enro, name and percentage of five students into a file named “student.txt”.

2. Write a C++ program to create two files named “student.txt” and “result.txt” which stores respective details. Fetch the records both files in a proper manner.

Practical Work

1. Write a C++ program to create the file country and capital, which store the name of the country and name of its capital respectively. Use getline function and display the content of both the files on the screen.

2. Consider the above program and write a C++ program to read the file of country and capital, which will fetch the name of the country first and then read the name of its capital respectively. Use getline function and display the content of both the files accordingly for the data.

CE: 6.5
File Pointer and Their Manipulators for Random Access

CE: 6.5. File Pointer and Their Manipulators for Random Access

- The file management system associates two pointers with each file, called file pointers.

- In C++, they are called…
  1. get pointer (input pointer) and
  2. Put pointer (output pointer)

- These pointer facilitate the movement across the file while reading or writing.

- The **get** pointer specifies a location from where the current reading operation is initiated.

- The **put** pointer specifies a location from where the current writing operation is initiated.

CE: 6.5. File Pointer and Their Manipulators for Random Access (Conti…)

- The file pointers are set to a suitable location initially based on the mode in which the file is opened.

- A file can be the…
  1. Read mode,
  2. Write mode or
  3. Append mode

CE: 6.5. File Pointer and Their Manipulators for Random Access (Conti…)

1. Read-only Mode:

- When a file is opened in read-only mode, the input pointer is at the beginning of the file. So that the file can be read from the start.

2. Write-only Mode:

- When a file is opened in write-only mode, the exiting contents of the file is deleted, and the output pointer is set at the beginning of the file. So that the data can be written from the start.

3. Append Mode:

- When a file is opened in append mode, the exiting contents of the file is remain unaffected, and the output pointer is set at the end of the file. So that the data can be written at the end of the existing contents.

CE: 6.5. File Pointer and Their Manipulators for Random Access (Conti…)

- C++ does not provide commands organizing and processing files as sequential or direct (random) files.
- It provides *files manipulation* commands which can be used by the programmer to device access to files sequentially or **randomly**.
- The functions, put() and get() are designed to manage a single character at a time.
- The other functions, **write()** and **read()** are designed to manipulation blocks of character data.
- In C++, **random access is achieved by manipulating functions**.

CE: 6.5. File Pointer and Their Manipulators for Random Access (Conti…)

Functions for Manipulation of file pointers:

Function	Member of the class	Action Performed
seekg()	ifstream	Moves get pointer(input) to a specified location
seekp()	ofstream	Moves put pointer (output) to a specified location
tellg()	ifstream	Returns the current position of the get pointer
tellp()	ofstream	Returns the current position of the put pointer

- These functions allow the programmer to have control over a position in the file where the read or write operation take place.

CE: 6.5. File Pointer and Their Manipulators for Random Access (Conti…)

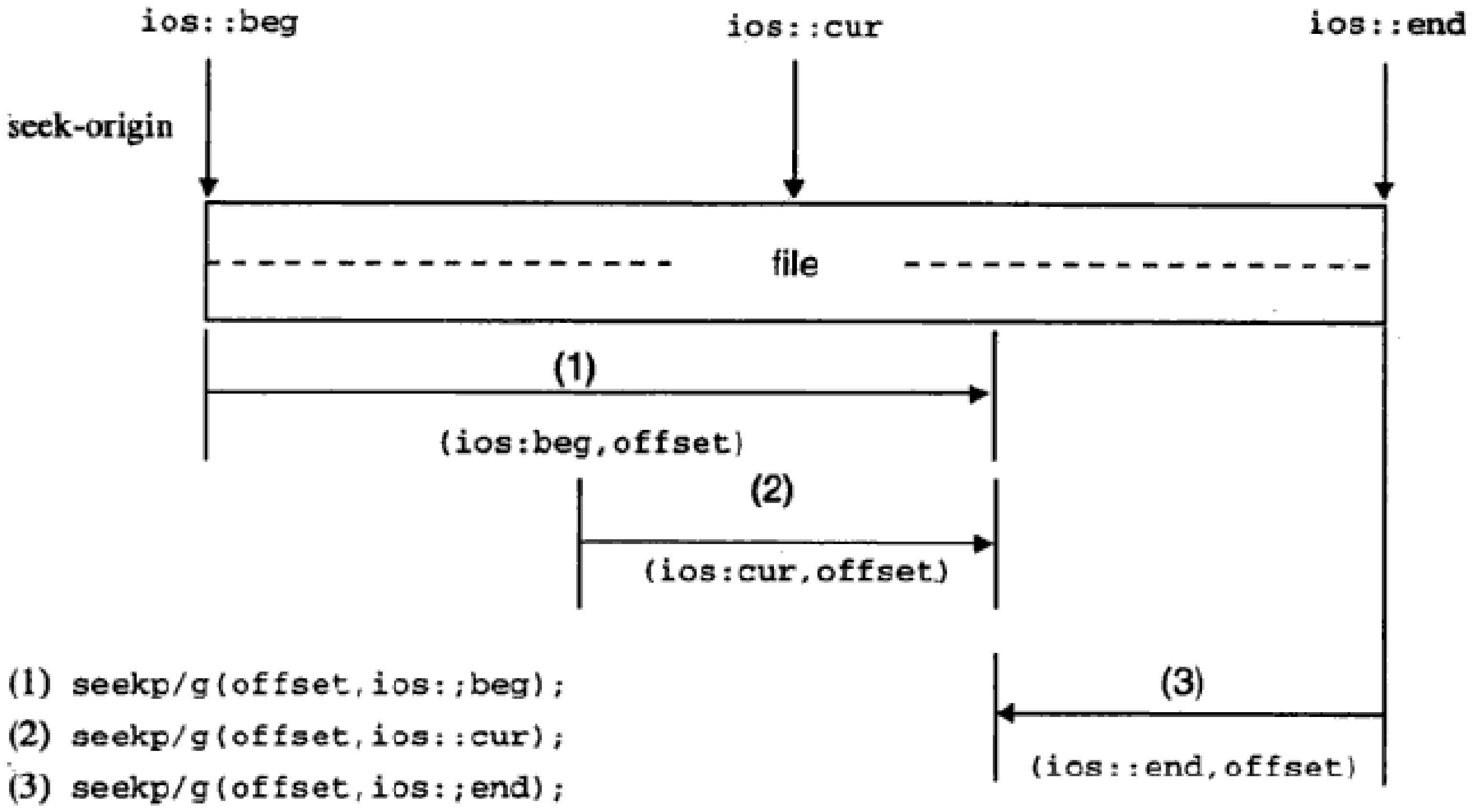
- The two seek functions have the following prototypes:

```
istream & seekg(long offset, seek_dir origin = ios::beg);
ostream & seekp(long offset, seek_dir origin = ios::beg);
```

Origin value	Seeks from…
ios::beg	Seek from beginning of file
ios::cur	Seek from current location
ios::end	Seek from end of file

File seek origins

CE: 6.5. File Pointer and Their Manipulators for Random Access (Conti…)



Seek position and their origin

CE: 6.5. File Pointer and Their Manipulators for Random Access (Conti…)

- For Example:

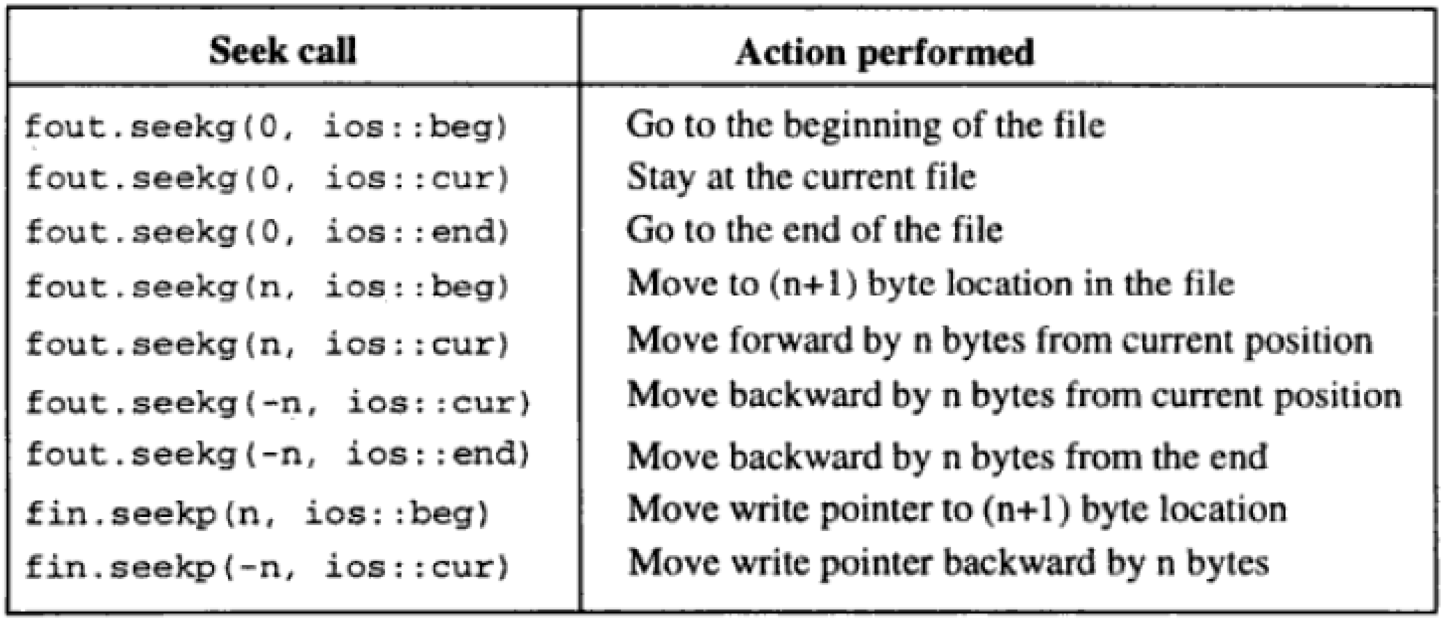
```
infile.seekg(20, ios::beg);
          or
infile.seekg(20);

outfile.seekp(20, ios::beg);
          or
outfile.seekp(20);

infile.seekg(0, ios::end);
infile.tellg();
```

CE: 6.5. File Pointer and Their Manipulators for Random Access (Conti…)

- Some of the pointer offset calls and their actions are:



Seek call	Action performed
`fout.seekg(0, ios::beg)`	Go to the beginning of the file
`fout.seekg(0, ios::cur)`	Stay at the current file
`fout.seekg(0, ios::end)`	Go to the end of the file
`fout.seekg(n, ios::beg)`	Move to (n+1) byte location in the file
`fout.seekg(n, ios::cur)`	Move forward by n bytes from current position
`fout.seekg(-n, ios::cur)`	Move backward by n bytes from current position
`fout.seekg(-n, ios::end)`	Move backward by n bytes from the end
`fin.seekp(n, ios::beg)`	Move write pointer to (n+1) byte location
`fin.seekp(-n, ios::cur)`	Move write pointer backward by n bytes

CE: write() and read() functions

- These member functions are used to store or retrieve data in binary form.

- Unlike put() and get(), the write() and read() functions access in binary format.

- The binary form is more accurate and provides faster access to the file, because no conversion is required while performing read or write.

- Syntax for write() and read();
  - ✓ To write object's data members in a file :

    ```
    fout.write((char *) & class_obj, sizeof(class_obj));
    ```

  - ✓ To read file's data members into an object :

    ```
    fin.read((char *) & class_obj, sizeof(class_obj));
    ```

<u>CE: write() and read() functions (Conti…)</u>

- The functions read() and write() can also be used for reading and writing class objects.
- These functions handle the entire structure of an object as a single unit.
- One thing that must be remembered is that only data members are written to the disk file and not the member functions.
- The length of an object in obtained by **sizeof** operator and it represents the sum total of lengths of all data members of the object.

Program for reading and writing class object.

```
#include<iostream>
#include<fstream>
using namespace std;
class student
   {
                  char name[40];
                   float per;
             public:
                   void getdata()
                      {
                              cout<<"Enter your name: ";
                              cin.getline(name,40);
                              cout<<"Enter your percentage: ";
                              cin>>per;
                      }
                  void addrecords()
                      {
                              cout<<"\nName: "<<name;
                              cout<<"Percentage: "<<per;      }     };
```

Program for reading and writing class object. (Conti…)

```
int main()
  {
        student s;
        ofstream fout;
        fout.open("student.txt", ios::out);
        if(!fout)
          {
                cout<<"Error in creating file.."<<endl;
                return 0;
          }
        cout<<"\nFile created successfully."<<endl;

        s.getdata();
        fout.write((char *) & s, sizeof(s));

        fout.close();
        return 0;
  }
```

Program to search in a file having records.

```
#include<iostream>
#include<fstream>
using namespace std;
class student
  {
            int enro;
            char name[40];
            char Class[4];
            float per;
            char grade;
        public:
            void getdata()
              {
                    cout<<"Enter your enro: ";
                    cin>>enro;
                    cout<<"Enter your name: ";
                    cin>>name;
                    cout<<"Enter your class: ";
                    cin>>class;
```

Program to search in a file having records. (Conti…)

```
            cout<<"Enter your percentage: ";
            cin>>per;

            if(per >= 75)
                    grade='A';
           else if(per >= 60)
                    grade='B';
           else if(per >= 50)
                    grade='C';
           else if(per >= 40)
                    grade='D';
           else
                    grade='F';
       }
   void putdata()
       {
            cout<<"\nEnro. : "<<enro;
            cout<<"Name: "<<name;
            cout<<"Class: "<<Class;
```

Program to search in a file having records. (Conti…)

```
                        cout<<"Percentage: "<<per;
                        cout<<"Grade: "<<grade;
                   }
                int getenro()
                   {
                        return erno;
                    }
    };
  int main()
    {
          student s;
          char found='n';
          ifstream fin("student.txt", ios::in);
          cout<<"Enter enro. that is to be searched for: ";
          cin>>enro;
          while(!fin.eof)
            {
                fin.read((char *) & s, sizeof(s));
```

Program to search in a file having records. (Conti…)

```
                    if(s.getenro() == enro)
                        {
                                s.putdata();
                                found = 'y';
                                break;
                         }
            }
        if(found == 'n')
            {
                    cout<<"\nEnrollment no. not found!"<<endl;
            }
         fin.close();
         return 0;
    }
```

CE: 6.6
File Error Handling During File Manipulations

CE: 6.6. File Error Handling During File Manipulations

- Sometimes during file operations, errors may be arise in.

- For instance,
  - ✓ a file being opened for reading might not exist.
  - ✓ a file name used for a new file may already exist.
  - ✓ An attempt could be made to read past the end-of-file.
  - ✓ An invalid operation may be performed.
  - ✓ There might not be enough space in the disk for storing data.

- To check for such errors and to ensure smooth processing, C++ file streams inherit 'stream-state' members from the ios class.

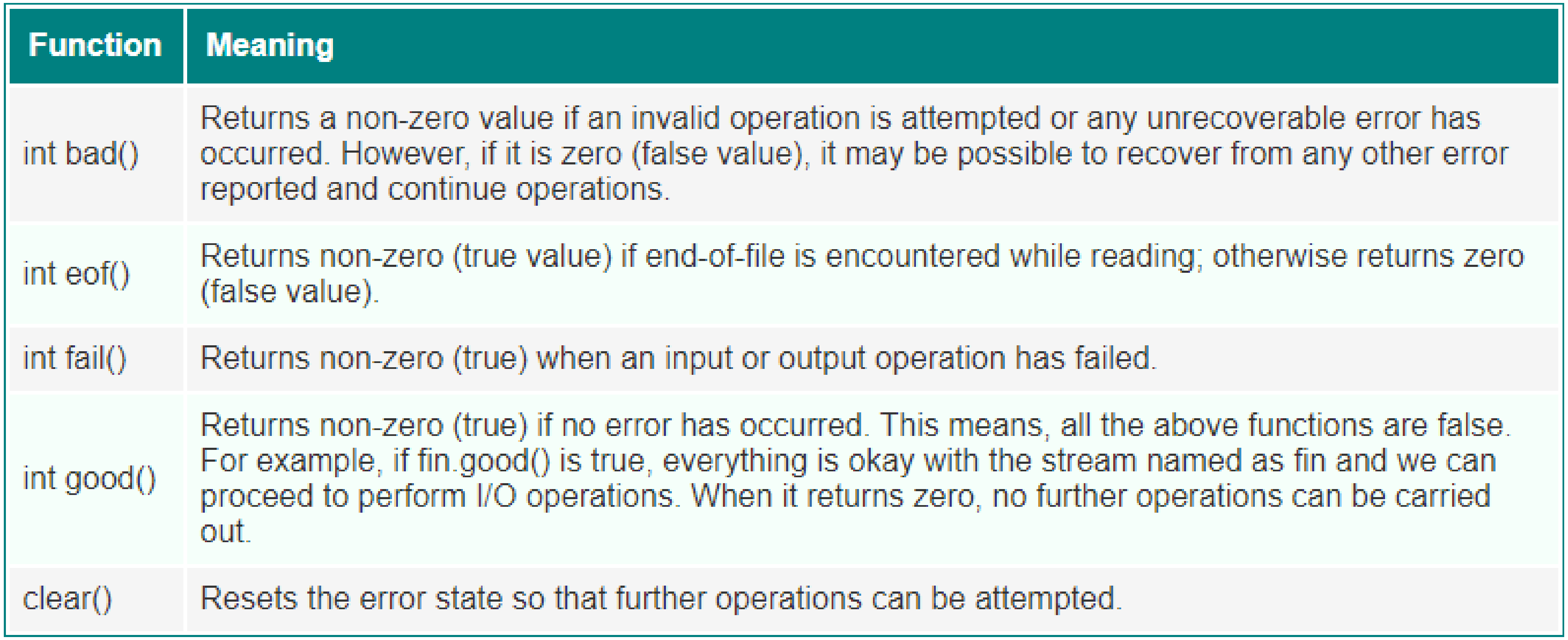
CE: 6.6. File Error Handling During File Manipulations (Conti…)

- The current state of the I/O system is held in an integer, in which the following flags are encoded:

Name	Meaning
eofbit	1 when end-of file is encountered, 0 otherwise
failbit	1 when a non-fatal I/O error has occurred, 0 otherwise
badbit	1 when a fatal I/O error has occurred, 0 otherwise
goodbit	0 value

CE: 6.6. File Error Handling During File Manipulations (Conti…)

- There are also several error handling functions supported by class **ios** that help you read and process the status recorded in a file stream.



Function	Meaning
int bad()	Returns a non-zero value if an invalid operation is attempted or any unrecoverable error has occurred. However, if it is zero (false value), it may be possible to recover from any other error reported and continue operations.
int eof()	Returns non-zero (true value) if end-of-file is encountered while reading; otherwise returns zero (false value).
int fail()	Returns non-zero (true) when an input or output operation has failed.
int good()	Returns non-zero (true) if no error has occurred. This means, all the above functions are false. For example, if fin.good() is true, everything is okay with the stream named as fin and we can proceed to perform I/O operations. When it returns zero, no further operations can be carried out.
clear()	Resets the error state so that further operations can be attempted.

CE: 6.6. File Error Handling During File Manipulations (Conti…)

- Therefore,
  - ✓ eof() returns true, if eofbit is set;
  - ✓ bad() returns true, if badbit is set.
  - ✓ fail() returns true, if failbit is set;
  - ✓ good() returns true, it there are no errors.
  - ✓ otherwise, they return false.

- These functions may be used in the appropriate places in a program to locate the status of a file stream

CE: 6.6. File Error Handling During File Manipulations (Conti…)

```
ifstream fin;
fin.open("master", ios::in);
while(!fin.fail())
     {
             ………. // process the file
      }
if(fin.eof())
    {
             ………. // terminate the program
    }
else if(fin.bad())
   {
             ………. // report fatal error
    }
else
   {
             fin.clear();     // clear error-state flags
             ……….
   }
```

Home Work

1. What is stream?
2. What is input stream?
3. What is output stream?
4. Draw the hierarchical diagram of stream classes.
5. Which are the unformatted I/O functions?
6. Which are the formatted I/O functions?
7. Which stream class is used to write the files?
8. Describe briefly the features of I/O system supported by C++?
9. Give purpose of any two flags used with IOS class member function setf().
10. What is return type of tellg() function?
11. List atleast four ios format functions. Write task of each functions in one line.
12. How do the following two statements are differ in operation?
    1. cin>>c;
    2. cin.get(c);
13. How can you create user defined manipulators? Explain with example.

Home Work

14. What is the key difference between ifstream and fstream?
15. Which functions are used to read and write data into binary file?
16. How many arguments are there in getline()? Which are they?
17. What is the use of eof() in file operation? What is its return value?
18. By how many ways you can open and close file? State it with proper syntax.
19. Name two member functions that are supported by stream classes of C++.
20. Write a code to open a file for reading. Also display message "File not open" if file is not opened successfully.
21. Write four error handling functions with their return values.
22. What is the usability of ifstream, ofstream and fstream class? When do you use fstream class?
23. Which ios flag is used to show the trailing zeros in fraction value?
24. Which function is used to accept a line as an input? Give example.
25. Differentiate ios::beg, ios::cur and ios::end file seek origins.

Home Work

26. Draw files stream class hierarchy.
27. State the use of seekg() and seekp() in file.
28. Give output for the given code.

    ```
    Cout.fill(‘*’);
    Cout.precision(2);
    Cout.width(6);
    Cout<<12.53;
    Cout.width(6);
    Cout<<20.5;
    ```
29. Enlist different situation in file handling where error handling is required.
30. When one should use eof() function?
31. State the use of precision function in I/O manipulation.
32. State the use of tellg() and tellp() in file.
33. Write two different ways to open a file.

Home Work

34. Create Student class with data members like roll number, name and course. Write a program to store object to file and read object from file.

35. Write a C++ program to input a word and a file name. Display occurrences of word in a file. Do appropriate data validation.

36. How do you store object information in data file in C++? Create a class employee to maintain information such as employee id, name, designation and salary. Write a C++ program to store employee’s object information in data file emp.txt.

37. What are the different file opening modes while opening data files in C++? Also write a program to count number of characters from the inputted file.

Thank You!