

CS217 – Object Oriented Programming (OOP)

Week – 15

May 24-28, 2021

Instructor: **Basit Ali**

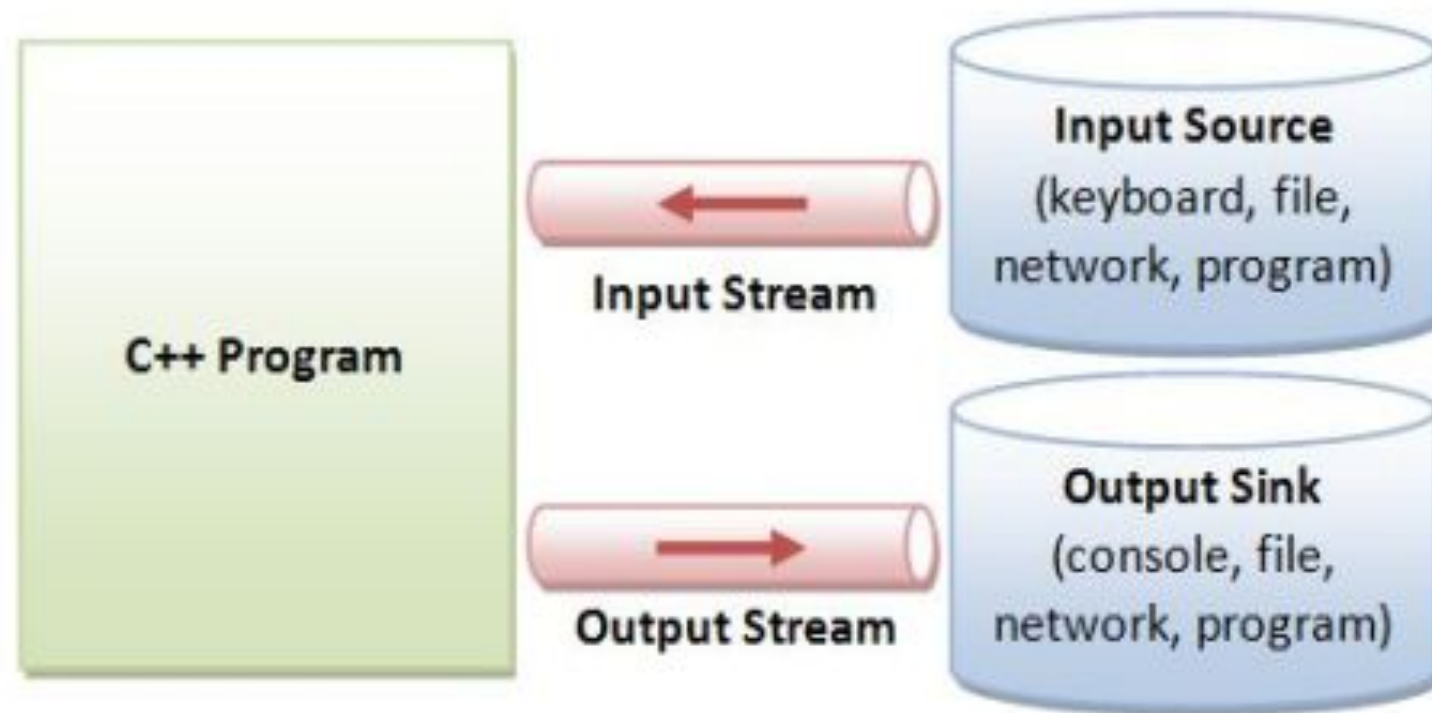
Streams

□ Stream

- A transfer of information in the form of a sequence of bytes

□ I/O Operations

- Input: A stream that flows from an input device (i.e.: keyboard, disk drive, network connection) to main memory
- Output: A stream that flows from main memory to an output device (i.e.: screen, printer, disk drive, network connection)



Iostream Library Header Files

iostream library

- **<iostream.h>**: Contains **cin** & **cout** objects
- **<fstream.h>**: Contains information important to user-controlled file processing operations

file (fstream)

- **ifstream** - defines new input stream (normally associated with a file).
- **ofstream** - defines new output stream (normally associated with a file).

General File I/O Steps

1. Include the header file `fstream` in the program.
2. Declare file stream variables.
3. Open the file
4. Use the file stream variables with `>>`, `<<`, or other input/output functions.
5. Close the file.

```

2  #include<iostream>
3  #include <string>
4  #include <fstream>
5  using namespace std;
6
7  int main ()
8  {
9      /* Declare file stream variables such as
10       the following */
11
12     ifstream fsIn;//input
13     ofstream fsOut; // output
14     fstream both; //input & output
15
16     //Open the files
17     fsIn.open("prog1.txt"); //open the input file
18     fsOut.open("prog2.txt"); //open the output file
19
20     //Code for data manipulation
21     //Close files
22     fsIn.close();
23     fsOut.close();
24     return 0;
25 }

```

```
#include <fstream.h>
```

```

int main (void)
{
    // Local Declarations
    ifstream    fsIn;

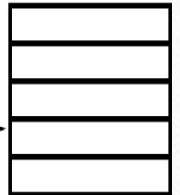
    ofstream    fsOut;
    •
    •
    •

} // main

```

fsIn is an input
instance of ifstream

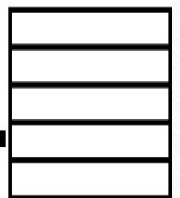
fsIn



memory

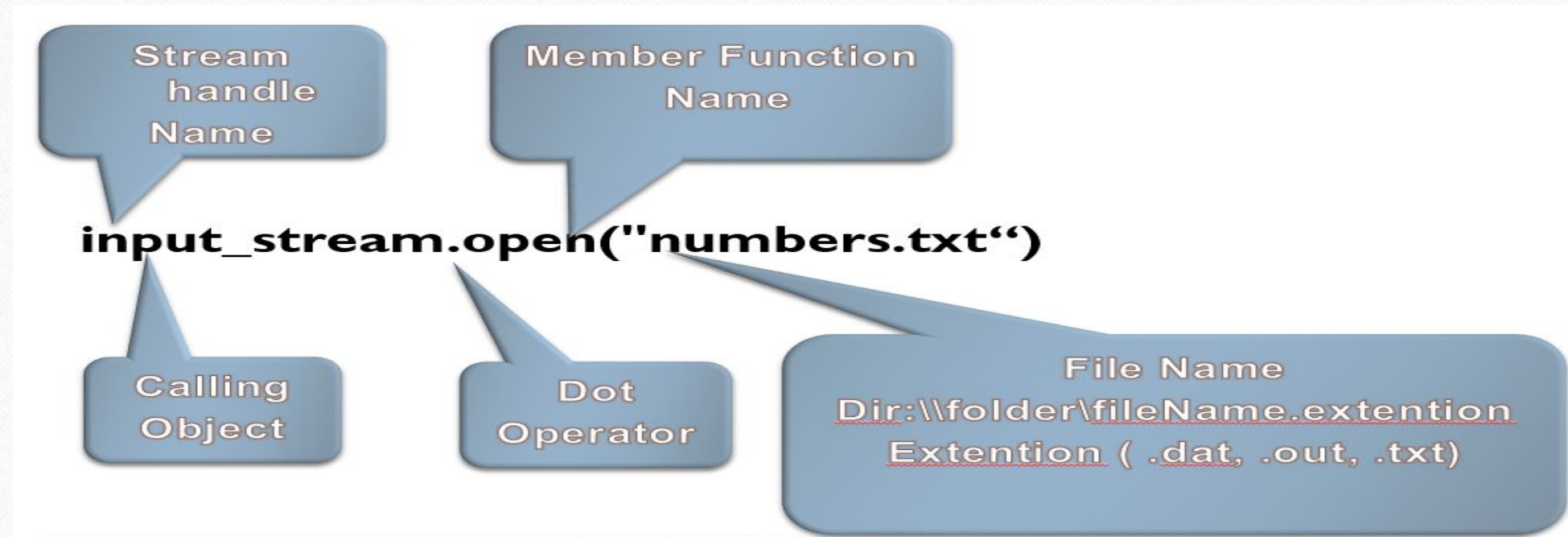
fsOut is an output
instance of ofstream

fsOut



memory

Object and Member Functions



Open()

- ❑ Opening a file associates a file stream variable declared in the program with a physical file at the source, such as a disk.
- ❑ In the case of an input file:
 - ❑ the file must exist before the open statement executes.
 - ❑ If the file does not exist, the open statement fails and the input stream enters the fail state
- ❑ An output file does not have to exist before it is opened;
 - ❑ if the output file does not exist, the computer prepares an empty file for output.
 - ❑ If the designated output file already exists, by default, the old contents are erased when the file is opened.

Validate the file before trying to access

Method 1:

By checking the stream variable;

```
If ( ! Mstream)
```

```
{
```

```
Cout << "Cannot open file.\n";
```

```
}
```

Method 2:

By using `bool is_open()` function.

```
If ( ! Mstream.is_open()) {
```

```
Cout << "File is not open.\n";
```

```
}
```


Input File-Related Functions

- **`fsin.get(char character)`**

extracts next character from the input stream `fsin` and places it in the character variable `character`.

- **`fsin.eof()`**

tests for the end-of-file condition.

File I/O Example: Reading

- Read char by char

```
#include <iostream>
#include <fstream>

int main()
{
    //Declare and open a text file
    ifstream openFile("data.txt");

    char ch;

    //do until the end of file
    while( ! OpenFile.eof() )
    {
        OpenFile.get(ch); // get one character
        cout << ch;      // display the character
    }

    OpenFile.close(); // close the file

    return 0;
}
```

- Read a line

```
#include <iostream>
#include <fstream>
#include <string>

int main()
{
    //Declare and open a text file
    ifstream openFile("data.txt");

    string line;

    while(!openFile.eof())
    {
        //fetch line from data.txt and put it in a string
        getline(openFile, line);

        cout << line;
    }

    openFile.close(); // close the file

    return 0; }
}
```


Output File-Related Functions

- **ofstream fsOut;**
- **fsOut.open(const char[] fname)**
connects stream fsOut to the external file fname.
- **fsOut.put(char character)**
inserts character character to the output stream fsOut.
- **fsOut.eof()**
tests for the end-of-file condition.

File I/O Example: Writing

- First Method (use the constructor)

```
#include <fstream>
using namespace std;
int main()
{
    /* declare and automatically open the file */
    ofstream outFile("fout.txt");

    //behave just like cout, put the word into the file
    outFile << "Hello World!";
    outFile.close();
    return 0;
}
```

- Second Method (use Open function)

```
#include <fstream>
using namespace std;
int main()
{
    // declare output file variable
    ofstream outFile;
    // open an exist file fout.txt
    outFile.open("fout.txt");

    //behave just like cout, put the word into the file
    outFile << "Hello World!";

    outFile.close();

    return 0;
}
```


File Open Mode

Name	Description
ios::in	Open file to read
ios::out	Open file to write
ios::app	All the data you write, is put at the end of the file. It calls ios::out
ios::ate	All the data you write, is put at the end of the file. It does not call ios::out
ios::trunc	Deletes all previous content in the file. (empties the file)
ios::nocreate	If the file does not exists, opening it with the open() function gets impossible.
ios::noreplace	If the file exists, trying to open it with the open() function, returns an error.
ios::binary	Opens the file in binary mode.

File Open Mode

```
#include <fstream>
int main(void)
{
    ofstream outFile("file1.txt", ios::out);
    outFile << "That's new!\n";
    outFile.close();
        Return 0;
}
```

If you want to set more than one open mode, just use the **OR** operator- |. This way:

ios::ate | ios::binary

File format

- In c++ files we (read from/ write to) them as a stream of characters
- What if I want to write or read numbers ?

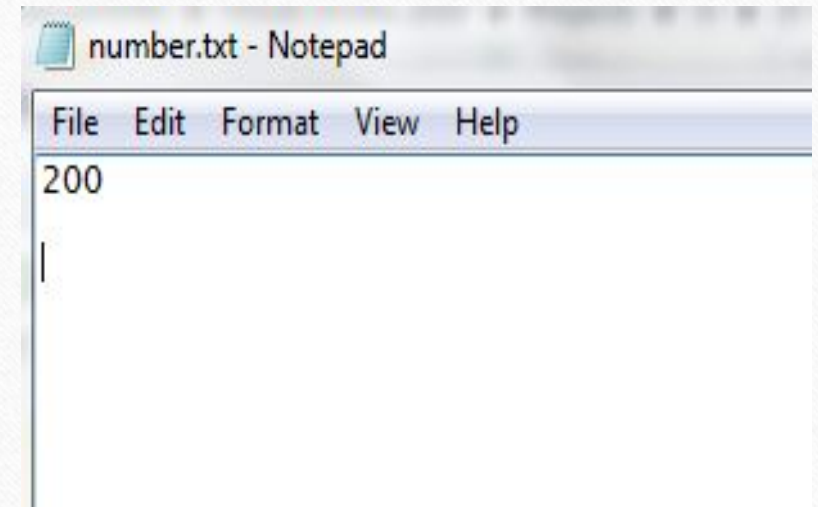
Example writing to file

```
#include <iostream>
#include <fstream>
using namespace std;
void main()
{
    ofstream outFile;
    // open an exist file fout.txt
    outFile.open("number.txt",ios::app);

    if (!outFile.is_open())
    { cout << " problem with opening the file ";}
    else
    {outFile <<200 <<endl ;
    cout << "done writing" <<endl;}

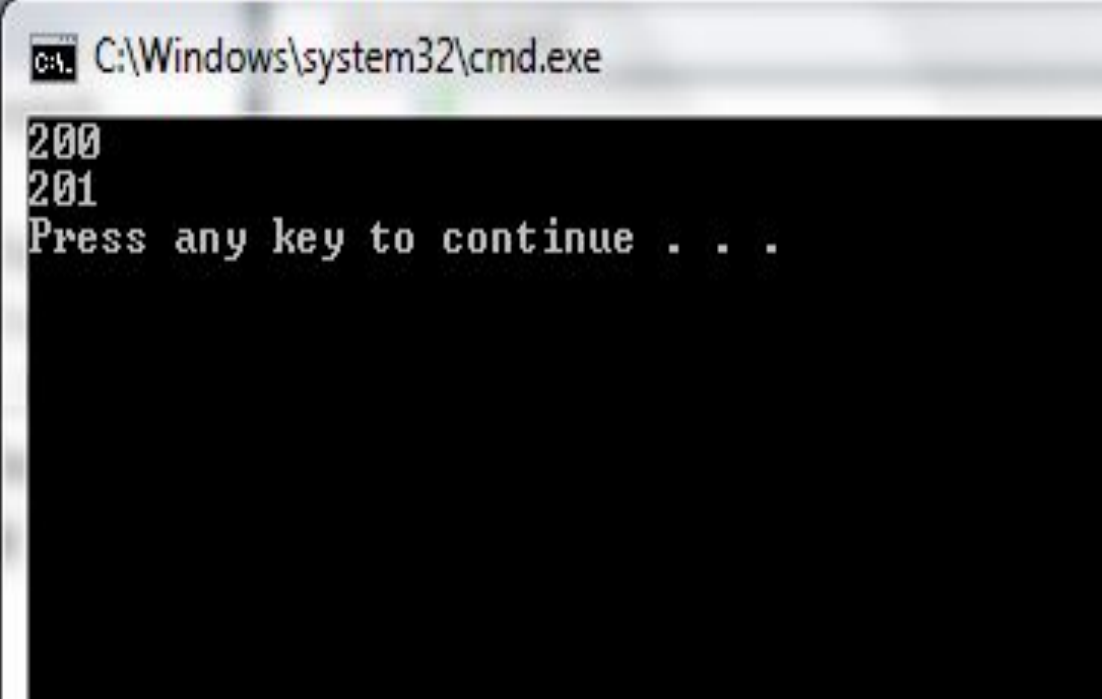
    outFile.close();

}
```



Example Reading from file

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
using namespace std;
void main()
{
    //Declare and open a text file
    ifstream INFile("number.txt");
    string line;
    int total=0;
    while(! INFile.eof())
    {
        getline(INFile, line);
        //converting line string to int
        stringstream(line) >> total;
        cout << line <<endl;
        cout <<total +1<<endl;}
    INFile.close(); // close the file
}
```



A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Windows\system32\cmd.exe'. The command prompt has a black background with white text. It displays the output of the C++ program: '200' on the first line, '201' on the second line, and 'Press any key to continue . . .' on the third line.

Writing OBJECTS to file

```
15 int main()
16 {
17     A a;
18     a.x = 10;
19     a.y = 20;
20
21     ofstream obj;
22
23     obj.open("C:/Users/basit.jasani/Desktop/abc.bin", ios::out | ios::binary | ios::trunc);
24     obj.write((char*)&a, sizeof(a));
25     obj.close();
26
27     ifstream obj2;
28
29     A b;
30
31     obj2.open("C:/Users/basit.jasani/Desktop/abc.bin", ios::in);
32     obj2.read((char*)&b, sizeof(b));
33
34     cout << b.x << " " << b.y << endl; //}
35     obj2.close();
36 }
```

```
8 class A
9 {
10     public:
11     int x;
12     int y;
13 };
```


Errors!

- In software industrial programming most of the programs contain bugs. Bigger the program greater number of bugs it contains.
- The following are mainly errors or bugs that occurred in any program:
 - **Logical error:**
In this error the logic implemented is incorrect. This error occurs due to less concentration of programmer or poor knowledge of programmer regarding program.
 - **Compilation error:**
This error is occurred due to use of wrong idiom, function or structure. This error is shown at compilation time of program.
 - **Run Time error:**
This error occurred at the run time. This error occurs when program crashes during run time..

Exception!

- An exception is a situation, which occurred by the runtime error. In other words, an exception is a runtime error. An exception may result in loss of data or an abnormal execution of program.
- It is a new feature that ANSI C++ included in it. Now almost all C++ compilers support this feature.

Exceptions!

- **Exceptions are different**, however. You can't eliminate exceptional circumstances; you can only prepare for them. Your users will run out of memory from time to time, and the only question is what you will do. Your choices are limited to these:
 - Crash the program.
 - Inform the user and exit gracefully.
 - Inform the user and allow the user to try to recover and continue.
 - Take corrective action and continue without disturbing the user.

Exceptions!

- Exception provides a method to control exceptional conditions (like run time error) or to control any crashed program by transferring control to some special functions called handler.
- We can use following the keywords or functions to control runtime error in C++:
 - `try {}`
 - `catch {}`
 - `throw()`

try {} block

- This block captures series of errors in any program at runtime and throw it to the catch block where user can customize the error message.
- Syntax:

```
try
{
//define program;
throw(variable);
}
```

catch {} block

- This block catches the error thrown by try block. This block contains method to customize error.

Syntax:

```
catch  
{  
//defines method to control error;  
}
```

throw function

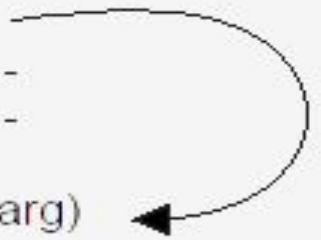
- This function is used to transfer the error from try block to catch block. This function plays major role to save program from crashing.

Syntax:

throw(variable);

Exception Flow!

```
try
{
    -----
    throw val;
    -----
}
catch(data-type arg)
{
    -----
    -----
    -----
}
```



throws
exception
value

The diagram illustrates the flow of an exception. A curved arrow originates from the `throw val;` statement within the `try` block and points to the `catch(data-type arg)` block. To the right of the arrow, the text "throws exception value" is written, indicating the action being performed.

try-blocks and if-else

- try-blocks are very similar to if-else statements
 - If everything is normal, the entire try-block is executed
 - else, if an exception is thrown, the catch-block is executed
- A big difference between try-blocks and if-else statements is the try-block's ability to send a message to one of its branches

Example of simple try-throw-catch

```
cout<<"\nEnter 1st number : ";
cin>>n1;

cout<<"\nEnter 2nd number : ";
cin>>n2;

try
{
    if(n2==0)
        throw n2;           //Statement 1
    else
    {
        result = n1 / n2;
        cout<<"\nThe result is : "<<result;
    }
}
catch(int x)
{
    cout<<"\nCan't divide by : "<<x;
}
```


Multiple Catches

```
int a = 3;  
  
try  
{  
    if(a==1)  
        throw a;           //Statement 1  
  
    else if (a==2)  
    {  
        throw 'A';  
    }  
  
    else if (a==3)  
    {  
        throw 1.5;  
    }  
}
```

```
catch(int x)  
{  
    cout<<"Integer exception caught";  
}  
  
catch(char a)  
{  
    cout<<"Char exception caught";  
}  
  
catch(double b)  
{  
    cout<<"Double exception caught";  
}
```

Catch all exceptions / Default Catch

```
int a = 2;  
  
try  
{  
    if(a==1)  
        throw a;           //Statement 1  
  
    else if (a==2)  
    {  
        throw 'A';  
    }  
  
    else if (a==3)  
    {  
        throw 1.5;  
    }  
}
```

```
catch(int x)  
{  
    cout<<"Integer exception caught";  
}  
  
catch(char a)  
{  
    cout<<"Char exception caught";  
}  
  
catch(...)  
{  
    cout<<"Exception caught";  
}
```


Nested Exception

```
int a = 0;

try
{
    try
    {
        throw(a);
    }
    catch(int a)
    {
        cout<<"X is integer"<<endl;
        throw(a);
    }
}
```

```
catch(int x)
{
    if(a>0)
    {
        cout<<"X is Positive"<<endl;
    }
}
catch(...)
{
    cout<<"X not an integer";
}
```


Exception not caught anywhere

```
main()
{
    int a = 0;

    try
    {
        throw a;
    }

    catch (char a)
    {
    }
}
```

C:\Users\basit.jasani\Downloads\code (1).exe

```
terminate called after throwing an instance of 'int'
-----
Process exited after 3.064 seconds with return value 3
Press any key to continue . . .
```

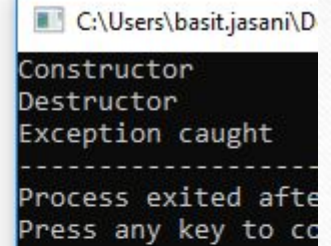
Concept!

- When an exception is thrown, all objects created inside the enclosing try block are destructed before the control is transferred to catch block.

Example!

```
class A{  
    public:  
    A()  
    {  
        cout<<"Constructor"<<endl;  
    }  
  
    ~A()  
    {  
        cout<<"Destructor"<<endl;  
    }  
};
```

```
main()  
{  
    int b = 0;  
  
    try  
    {  
        A a;  
        throw b;  
    }  
  
    catch (int a)  
    {  
        cout<<"Exception caught";  
    }  
}
```



```
C:\Users\basit.jasani\D  
Constructor  
Destructor  
Exception caught  
-----  
Process exited after  
Press any key to co
```