Dr. Vishwanath Karad
MIT WORLD PEACE UNIVERSITY | PUNE
TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

CS214 Object Oriented Programming

CS214 Object Oriented Programming 0-2-1

Course Objectives:

1. Understand basic concepts of Object Oriented Programming
2. Learn Inheritance, Polymorphism and Exception Handling features of Object Oriented Programming
3. Study concepts of Standard Template Library

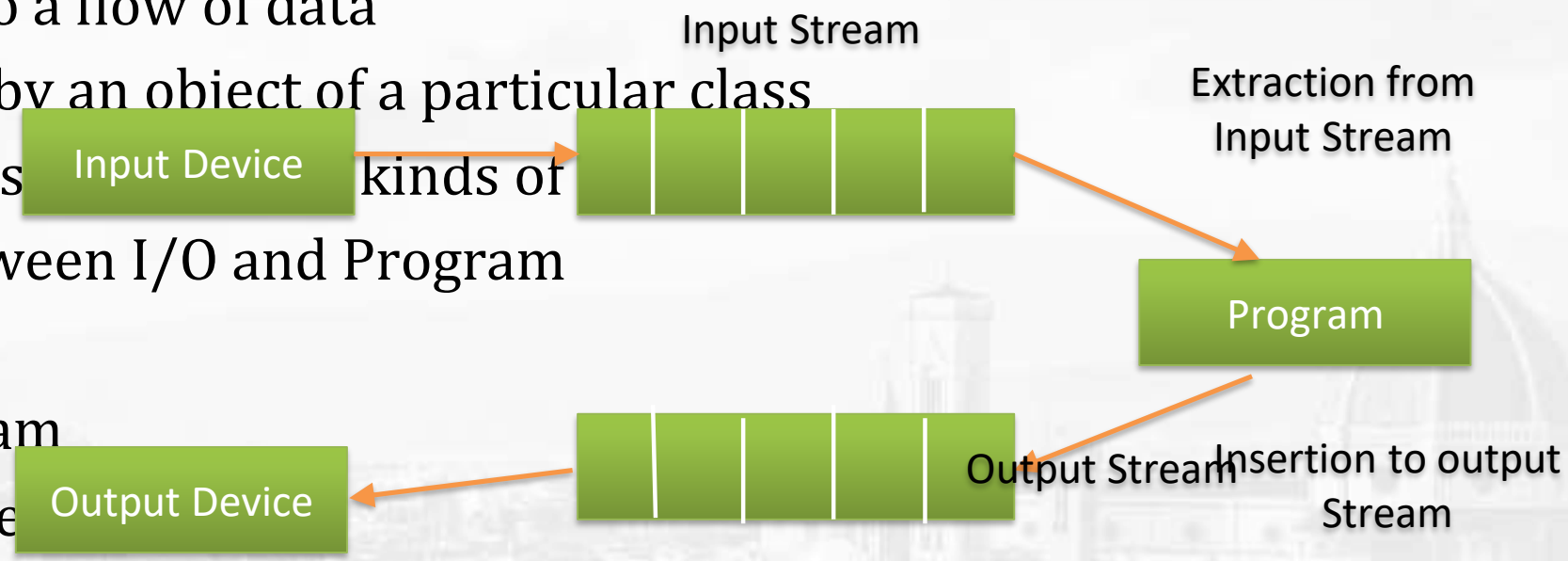
File Handling and Exception Handling

File Handling

- Steams and Files
- Stream classes & Stream Errors
- File Pointers
- File I/O with Member functions
- Formatted I/O and I/O manipulators
- Error handling during file operations
- Overloading of extraction and insertion operators

Streams

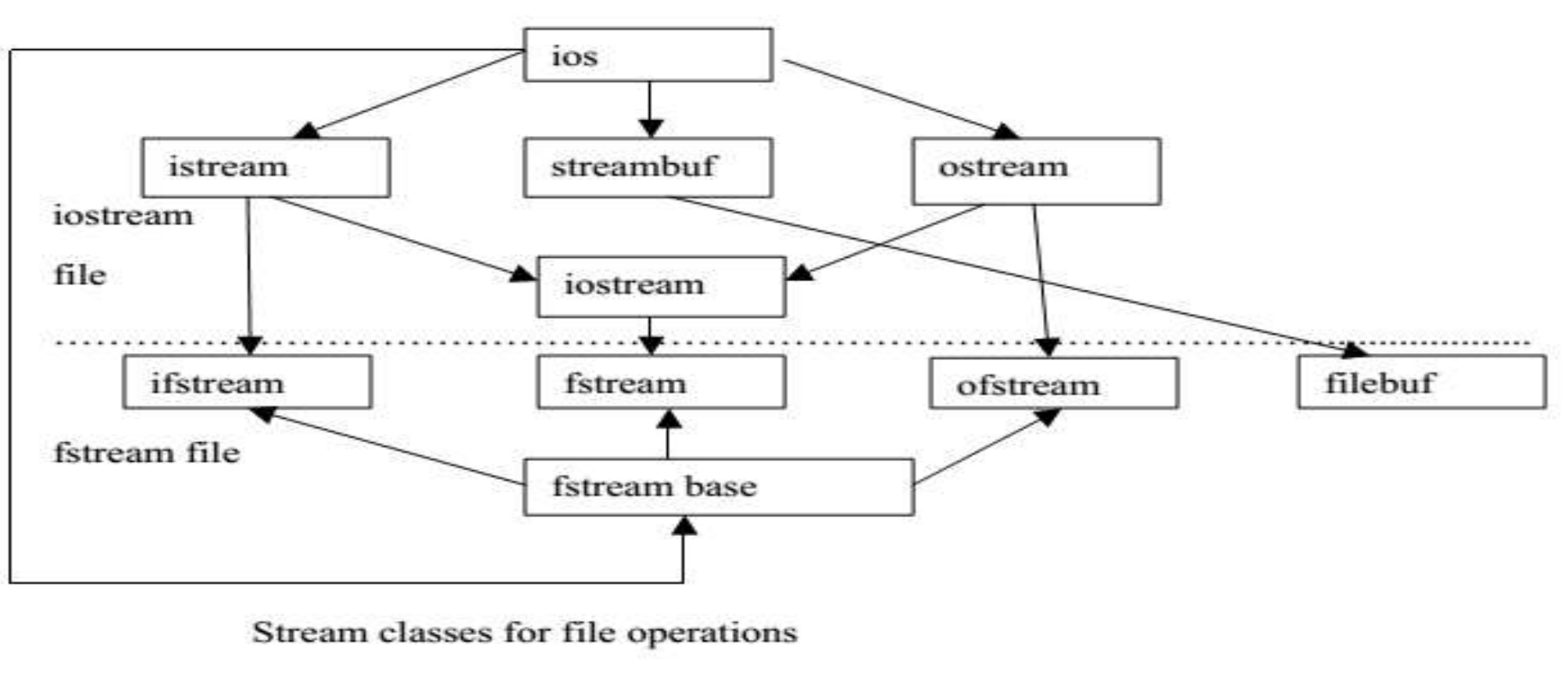
- Stream is sequence of bytes
- Name given to a flow of data
- Represented by an object of a particular class
- Used to repres kinds of
- Interface between I/O and Program
- Two streams
  - Input stream
  - Output stre



Input stream can come from keyboard or any other storage device.

Output stream can go to screen or any other storage device

Stream Classes



Stream classes for file operations

Stream Classes for I/O

Class name	Contents
ios (general input/output stream class)	▪ Basic facilities that are used by all other input-output classes
istream (input stream)	▪ Inherits properties of ios ▪ Has functions like get(), getline(), read() ▪ Contains operator >>
ostream (output stream)	▪ Inherits properties of ios ▪ put(), write() functions ▪ Contains operator <<
iostream (input/ output stream)	▪ Inherits properties of istream and ostream through multiple inheritance and thus contains all input and output functions

File

- A file is a collection of related data stored in a particular area on the disk .The data is stored in disk using the concept of file .
- The information / data stored under a specific name on a storage device, is called a file.
- Types of File

  1:ASCII Text File

  -Data Stored in ASCII character

  -Processed Sequentially in forward direction

  -Eof character is present at the end of file

  2:Binary File

  -Data Stored in binary format

  -Processed randomly
- Why to use File Handling

  -For permanent storage.

  -The transfer of input - data or output - data from one computer to another can be easily done by using files.

Stream Classes for File

Class name	Contents
fstream (general file input/output stream class)	▪ This is used to both read and write data from/to files ▪ Provides support for simultaneous input and output operations. ▪ Contains open() with default input mode. ▪ Inherits all the function from isteram and ostream classes through iostream
ifstream (file input stream)	▪ This is used to read data from files ▪ Provides input operations. ▪ Contains open() with default input mode. ▪ Inherits the functions **get()**, **getline()**, **read()**, **seekg()** and **tellg()** function from istream.
ofstream (file output stream)	▪ This is used to create a file and write data on files ▪ Provides output operations. ▪ Contains **open()** with default output mode. ▪ Inherits **put()**, **seekp()**, **teelp()** and **write()** function from ostream

Steps in File Handling

Following steps are followed in File I/O process:

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

Functions used in File Handling

Function	Operation
open( )	To open/create a file
close( )	To close an existing file
get( )	Read a single character from a file
put( )	write a single character in file.
read( )	Read data from file
write( )	Write data into file.

File open()

- Two ways to open file
  1) By Using Constructor
  2) By Using open() function

1) By using Constructor:

The file is open while creating object of stream classes.

Syntax:

file_stream_class stream_object("filename");

```
#include<fstream>
using namespace std;
int main()
 {
    ifstream fin("abc.txt");
    ofstream fout("xyz.txt");
 }
```

abc.txt open in input mode

abc.txt

```
Hi,
Welcome to MITWPU
```

xyz.txt open in output mode

xyz.txt

```
Hello
Welcome to CSE dept
```

File open()

- Two ways to open file
  1) By Using Constructor
  2) By Using open() function

2) By using open() function:

The function **open()** can be used to open multiple files that use the same stream object

Syntax:

file-stream-class stream-object;

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

```
#include<fstream>
int main()
 {
   ifstream fin;
   fin.open("abc.txt");
   ofstream fout;
   fout.open("xyz.txt");
   fstream file;
   file.open("pqr.txt");
  }
```

abc.txt open in input mode

abc.txt

Hi,
Welcome to MITWPU

xyz.txt open in output mode

xyz.txt

Hello
Welcome to CSE dept

pqr.txt open in input & output mode

pqr.txt

Object Oriented Programming

File Opening Modes

File mode parameter	Meaning
ios::app	Append to end of file
ios::ate	go to end of file on opening
ios::binary	file open in binary mode
ios::in	open file for reading only
ios::out	open file for writing only
ios::nocreate	open fails if the file does not exist
ios::noreplace	open fails if the file already exist
ios::trunc	delete the contents of the file if it exist

```
#include<fstream>
int main()
  {
      ifstream fin;
      fin.open("abc.txt",ios::in);
      ofstream fout;
      fout.open("xyz.txt",ios::out);
      fstream file;
      file.open("pqr.txt",ios::out|ios::app);
  }
```

abc.txt open in input mod

abc.txt

Hi,
Welcome to MITWPU

xyz.txt open in output mo

xyz.txt

Hello
Welcome to CSE dep

pqr.txt open in output & append mode

pqr.txt

Object Oriented Programming

File close()

- The function **close()** is used to close file. File must be closed after completing all operations on it

  Syntax: *stream_object.close();*

```
#include<fstream>
int main()
  {
       ifstream fin;
       fin.open("abc.txt",ios::in);
       ofstream fout;
       fout.open("xyz.txt",ios::out);
       fstream file;
       file.open("pqr.txt",ios::out|ios::app);
       fin.close();
       fout.close();
       file.close();
  }
```

abc.txt open in input mode

abc.txt

```
Hi,
Welcome to MITWPU
```

xyz.txt open in output mode

xyz.txt

```
Hello
Welcome to CSE dept
```

test .txt open in output & append mode

test.txt

```
Object Oriented Programming
```

bool is_open( )

- This function returns **true** if the **file is opened** and associated with this stream. Otherwise, it returns **false**:

Syntax*: fin.is_open();*

```
#include<fstream>
using namespace std;
int main()
{
        ifstream fin;
        fin.open("abc.txt");
        if (!fin.is_open())
         cout<<"Cannot open file!"<<endl;
        fin.close();
 }
```

return 1 if file is open else return 0

abc.txt

Hi, Welcome

Functions used in Input/output Operations

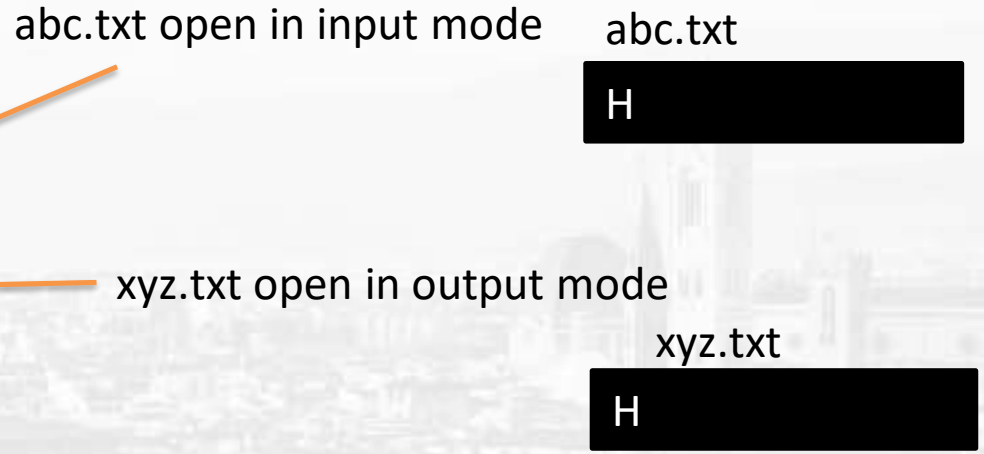
Function	Operation	Parameter
get( )	Read a single character from a file	ch=cin.get(); OR cin.get(ch); ch=fin.get(); OR fin.get(ch)
getline()	Read a line or text that ends with a newline character from a file	getline(cin, line) getline(cin, line,delimchar) OR getline(fin, line) getline(fin, line, delimchar)
put( )	write a single character in file.	cout.put(ch) OR fout.put(ch)
read( )	Read data from file	file . read ((char *)&V , sizeof (V));
write( )	Write data into file	file . write ((char *)&V , sizeof (V));
>> operator (Extraction)	Read data from file	cin>>ch OR fin>>ch
<< operator(Insertion)	Write data into file	cout<<ch OR fout<<ch

Reading and Writing into file using get() and put()

- get() function reads single character from file and put() function writes single character to file
- Syntax: *ch=fin.get();* OR *fin.get(ch)*
  fout.put(ch)

```
#include<fstream>
using namespace std;
int main() {
        ifstream fin;
        fin.open("abc.txt");
        ofstream fout;
        fout.open("xyz.txt");
        char ch='H';

        fin.get(ch);
        fout .put(ch);
        fin.close( );
        fout.close( );
        return 0;
```



getline()

- **getline**() function reads a line or text that ends with a newline character from a file

Syntax: **getline** (object, char* str, char delim = '\n');

getline(fin, line)
getline(fin, line, delimchar)

```
#include<fstream>
using namespace std;
int main() {
        string str;
        ifstream fin;
        fin.open("test.txt");

        getline(fin,str);
        cout<<"Data of Fileis"<<endl;
        cout<<str;

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

Reads a line from file

test.txt

```
Hi, Welcome to MITWPU
```

Output Screen

```
Data of file is
Hi, Welcome to MITWPU
```

Reading and Writing into file using >> & << operator

- Extraction >> operator reads data from file and Insertion << operator writes data to file

Syntax: *fin>>ch*

fout<<ch

```
#include<fstream>
using namespace std;
int main() {
        ifstream fin;
        fin.open("abc.txt");
        ofstream fout;
        fout.open("xyz.txt");
        char ch[30];
        fin>>ch;
        fout << ch;

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

Reads a data from file abc.txt

Writes a data to file xyz.txt

abc.txt

Hi, Welcome

xyz.txt

Hi, Welcome

```
class Student
 {
   int rollno;
   string name;
  public:
    void getdata()
     {
        cout<<"enter roll no & name of student";
                cin>>rollno>>name;
     }
   void putdata()
     {
       cout<<"Roll"<<rollno;
       cout<<"Name";
       cout<<name;
     }
 };
int main()
 {
      Student S1,S2;
       S1.getdata();
       fstream file;
       file.open("test.dat",ios::binary);
       file.write((char *)&S1,sizeof(S1));
       file.read((char *)&S2,sizeof(S2));
       S2.putdata() ;
       file.close();
       return 0;
  }
```

read() & write()

- **read**() function reads a data from file and **write**() function writes a data to file.

Syntax: **file . read ((char *)&V , sizeof (V));**
file . write ((char *)&V , sizeof (V));

test.txt

```
11
Abc
```

Output Screen

```
Roll 11
Name Abc
```

File Pointers and Manipulation Functions

- Each file have two associated pointers known as the file pointers.
- One of them is called the **Input Pointer** (or **get pointer**)
- The other is called the **Output Pointer** (or **put pointer**).
- The input pointer is used for reading the contents of a given file location and the output pointer is used for writing to a given file location.

To move file pointer to desired position use these function to manage the file pointers.

seekg() = moves get pointer (input) to a specified location

seekp() = moves put pointer (output) to a specified location

tellg() = gives the current position of the get pointer

tellp() = gives the current position of the put pointer

File Manipulation functions

Function	Operation
seekg(0, ios :: beg)	go to start
seekg(0, ios :: cur)	stay at current position
seekg(0, ios :: end)	go to the end of file
seekg(m, ios :: beg)	move to m+1 byte in the file
seekg(m, ios :: cur)	go forward by m bytes from the current position
seekg(-m, ios :: cur)	go backward by m bytes from the current position
seekg(-m, ios :: end)	go backward by m bytes from the end

Program to set file pointer

```
fstream file;
file.open("test.txt");
if(!file.is_open())
{cout << " Cannot open file!";  }
file << "This is the first line "<<endl;
file << "This is the second line" <<endl;
file.seekg(ios::beg);
for (int i = 0; i != 5; ++i) {

    cout << (char)file.get() << endl;
  }
char next = file.get();
cout << "The next character is " << (char)next ;
cout<<endl;
file.seekg(ios::beg);
char* str = new char[50];
file.getline(str, 50);
cout << str << endl;
file.ignore();
cout << "Peek " << (char) file.peek() << endl;
cout << "Current position is " << file.tellg() <<
endl;
```

open file test.txt for input and output

test.txt

```
This is the first line
This is the second line
```

reset position of the input

Output Screen

```
T
h
i
s

The next character is=i
This is the first line
Peek= h
Current position is= 24
```

reset position again

show the next character without extracti

get current position

Exception Handling Topics

- Exception Handling
- Exception Handling Mechanism
- Multiple Exceptions, re-throwing an exception
- Exception and Inheritance.

Exception Handling

- Exceptions
  - Indicate problems that occur during a program's execution
  - Occur infrequently
- Exception handling
  - Can resolve exceptions
    - Allow a program to continue executing or
    - Notify the user of the problem and
    - Terminate the program in a controlled manner
  - Makes programs robust and fault-tolerant
- Remove error-handling code from the program execution's "main line"

Exception Handling

- Programmers can handle any exceptions they choose
  - All exceptions
  - All exceptions of a certain type
  - All exceptions of a group of related types
- A standard mechanism for processing errors
  - Especially important when working on a project with a large team of programmers
- *C++* exception handling is much like Java's
- With exception handling, a program can continue executing (rather than terminating) after dealing with a problem.
- When no exceptions occur, there is no performance penalty

Exception Handling – try and catch blocks

```
try
{
```

Program statements
requires monitor for exceptions

```
}
```

```
catch( type  argument )
{
```

Program statements
handles for *Exception*

```
}
```

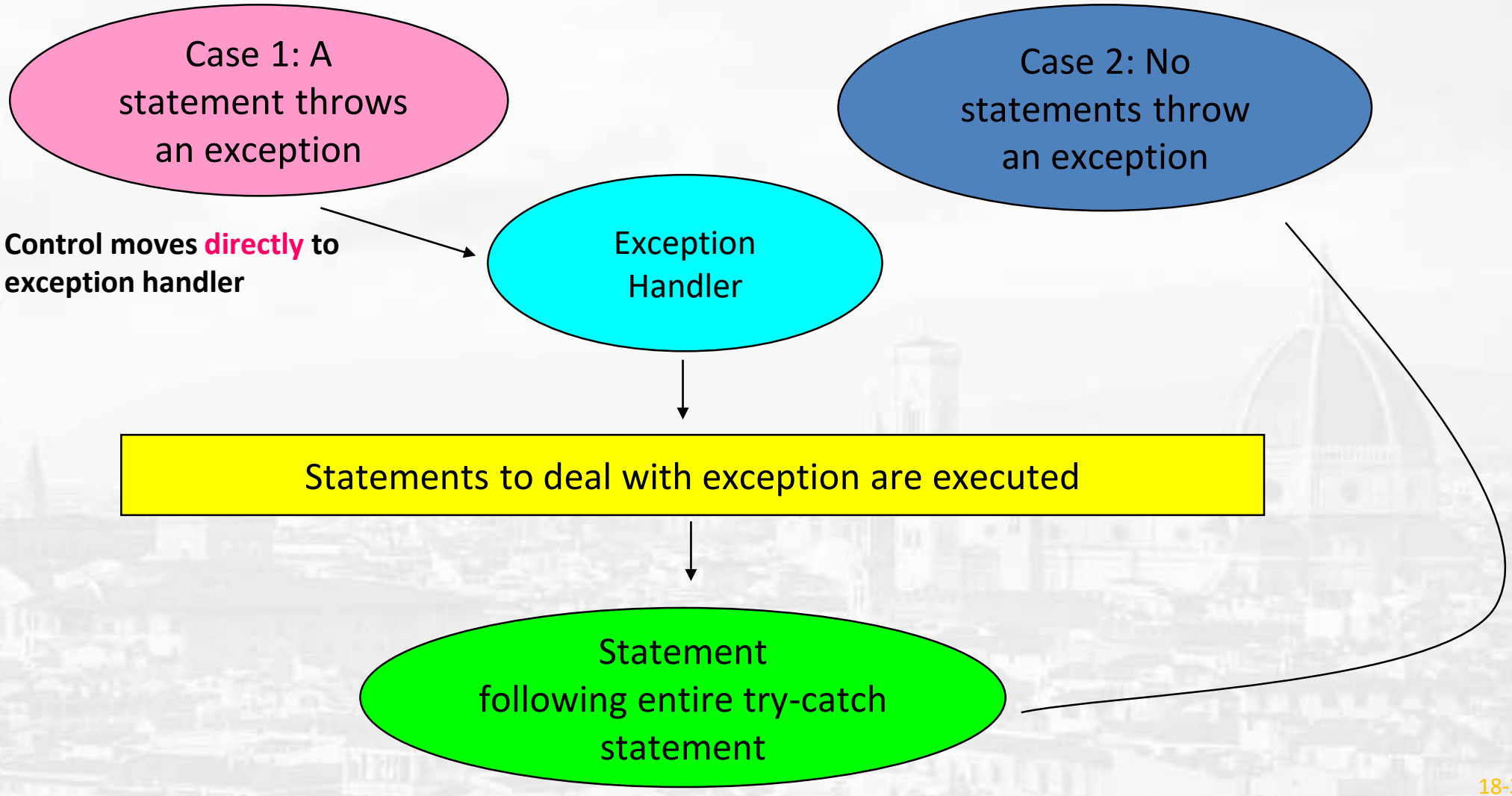
Example

```
try
{
    int d = 0;
    int a = 30 / d;
}
```


```
catch(int e )
{
    printf("Division by zero.");
}
```

```
try
{
    Program statements
    requires monitor for exceptions
}
catch( type1  argument )
{
    Prog
    han
}
    catch( type2  argument )
    {
        Program state
        handles for E
    }
        catch( typen  argument ){
            Program statements
            handles for Exception
        }
```

Execution of `try-catch`



C++ Exception Handlers

- **Exception Handlers Form:**

```
try {
-- code that is expected to raise an exception
}
catch (formal parameter) {
-- handler code
}
...
catch (formal parameter) {
-- handler code
}
```

Try-catch-throw Block

- **try**: Keyword **try** followed by braces (**{}**) Should enclose
  - Statements that might cause exceptions
  - Statements that should be skipped in case of an exception
- **catch**: Keyword **catch**
  - Immediately follow a **try** block
  - One or more **catch** handlers for each **try** block
  - Exception parameter enclosed in parentheses
  - Represents the type of exception to process
  - Executes if exception parameter type matches the exception thrown in
- **throw**: Keyword **throw** followed by an operand representing the type of exception
  - The **throw** operand can be of any type
  - If the **throw** operand is an object, it is called an **exception** object
  - The **throw** operand initializes the exception parameter in the matching **catch** handler, if one is found

```
try
 {
    Statement 1;
    Statement 2;
    throw x;
 } catch(m)
   {
        Error handler routine
   }
```

Flow of execution of try/catch blocks.

```
int main() {
  int x = -1;
  cout << "Before try  block\n";
  try {
            cout << "Inside try block \n";
            if (x < 0) {
                        throw x;
                        cout << "After throw (Never executed) \n";
            }
  }
  catch (int  x ) {
            cout << "Exception Caught \n";
  }
 cout << "After catch (Will be executed) \n";
 return 0;
}
```

O/P:

```
Before try  block
Inside try  block
Exception Caught
After catch (Will be executed)
```

Example: Exception divide by zero

```
#include<iostream>
using namespace std;
int main() {
	int opr1,opr2;    float res;
	cout << "Enter operand 1 and 2:-“;        cin>>opr1>>opr2;
	try {
		if (opr2 == 0) {
			throw (opr2);
		}
		else {
			res=(float)opr1/opr2;                cout << "Result is:" << res;
		}
	}
	catch (int i) {
			cout << "Answer is infinite because opr2 is:" << i;
	}
	return 0;
}
```

Ex: If exception is thrown and not caught then program terminates abnormally.

```
#include <iostream>
using namespace std;
int main()
{
    try{
        throw 'a';
    }
    catch(int x){
        cout << "Caught ";
    }
    return 0;
}
```

Catching All Exceptions:-

- If we use an ellipsis (...) as the parameter of catch, that handler will catch any exception doesn’t matter what the type of the throw exception is.
- try {

```
// code here}
catch (int param)
    { cout << "int exception"; }
catch (char param)
    { cout << "char exception"; }
catch (...)
    { cout << "default exception"; }
```

Exception Specification

- Also called **throw** lists
- Comma-separated list of exception classes in parentheses
- Indicates **someFunction** can **throw** types **ExceptionA**, **ExceptionB** and **ExceptionC**
- A function can throw only exceptions of types in its specification (or derived types)

  If a function throws a non-specification exception, function **unexpected** is called This normally terminates the program
- Absence of exception specification indicates that the function can **throw** any exception
- An empty exception specification, **throw()**, indicates the function *cannot* **throw** any exceptions

```
int someFunction( double value )
   throw ( ExceptionA, ExceptionB,
           ExceptionC )
{
   ...
}
```

Exception & Inheritance

- New exception classes can be defined to inherit from existing exception classes
- A **catch** handler for a particular exception class can also catch exceptions of classes derived from that class
    - Enables **catching** related errors with a concise notation

Standard Library Exception Hierarchy

- Base-class **exception**

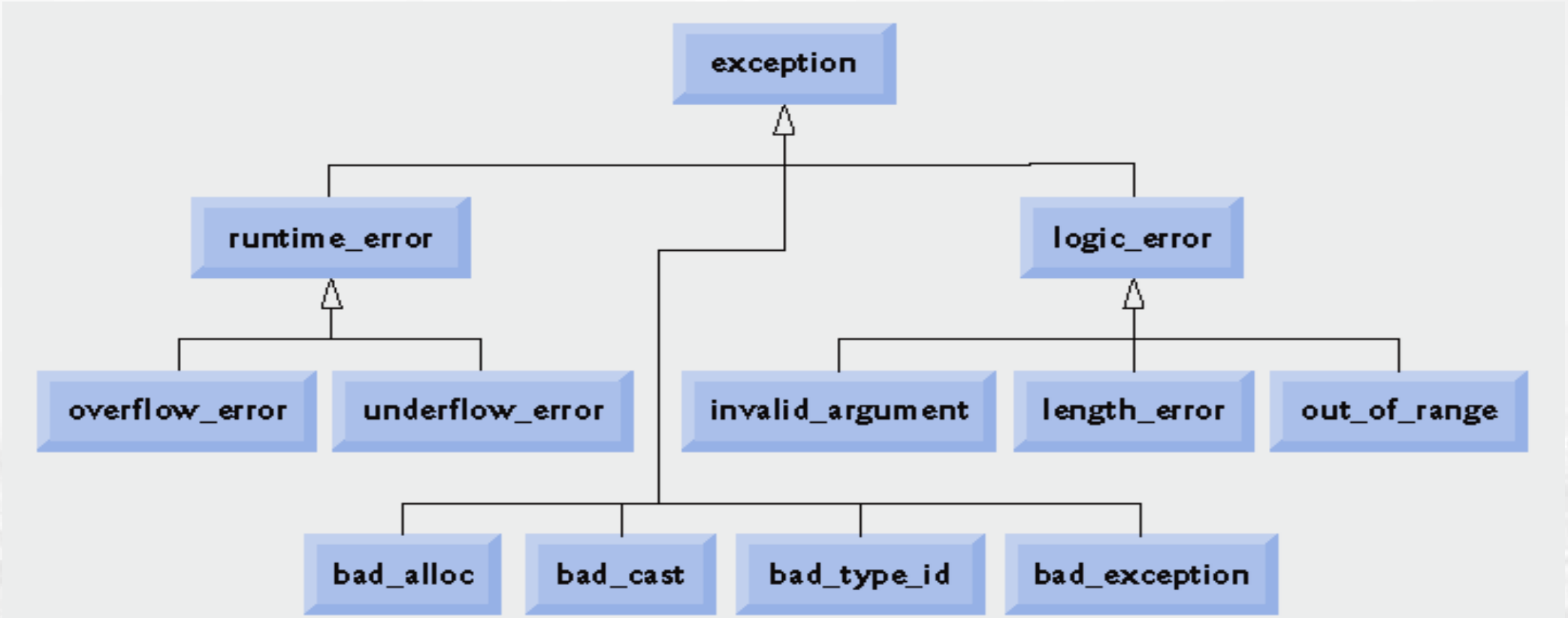
  Contains **virtual** function **what** for storing error messages

- Exception classes derived from **exception**

1. **bad_alloc** – thrown by **new**
2. **bad_cast** – thrown by **dynamic_cast**
3. **bad_typeid** – thrown by **typeid**
4. **bad_exception** – thrown by **unexpected**

Instead of terminating the program or calling the function specified by **set_unexpected** Used only if **bad_exception** is in the function's **throw** list

Standard Library Exceptions



Summary

Course Outcomes:

After completion of this course, students will be able to:

1. Apply the basic concepts of Object Oriented Programming in application development
2. Design and develop real world applications using inheritance, Polymorphism and Exception Handling features
3. Explore and use Standard Template Library to simplify programming

Reference

Books:

1. Herbert Schildt, 'C++ The Complete Reference', Fourth Edition, McGraw Hill Professional, 2011, ISBN-13: 978-0072226805
2. Robert Lafore, 'Object-Oriented Programming in C++', Fourth Edition, Sams Publishing, ISBN: 0672323087, ISBN-13: 978-8131722824
3. Bjarne Stroustrup, 'The C++ Programming language', Third Edition, Pearson Education. ISBN: 9788131705216
4. K. R. Venugopal, Rajkumar Buyya, T. Ravishankar, 'Mastering C++', Tata McGraw-Hill, ISBN 13: 9780074634547

Weblinks:

1. https://nptel.ac.in/courses/106105151/
2. http://nptel.ac.in/syllabus/106106110/
3. http://ocw.mit.edu

MOOCs:

https://onlinecourses.nptel.ac.in/noc18_cs32

Question Bank

- Theory Questions
- MCQ

Practice Assignments

1. Program to read from text file and display it
2. Program to count number of words
3. Program to copy contents of file to another file
4. Program to replace word in a file
5. Write a menu driven program that will create a data file containing the list of telephone numbers in the following form

   John 23456
   Ahmed 9876

   Use a class object to store each set of data, access the file created and implement the following tasks
   I. Determine the telephone number of specified person
   II. Determine the name if telephone number is known
   III. Update the telephone number, whenever there is a change.

Thank You!!