# Unit IV
# Input-Output, File Handling, Exception Handling and Templates in C++

Complete UG / BCA Level Descriptive Notes

Hindi + English Mix with Examples, Diagrams, Exam Questions and MCQs

Topics Covered: I/O Classes, File and Stream Classes, Char I/O, String I/O, Object I/O, File Pointers, Opening and Closing Files, Exception Handling, try-catch-throw/throws concept, Templates.

# 1. Introduction to Input-Output in C++

Input-Output यानी I/O का meaning है program में data लेना और result display करना. C++ में I/O operations stream concept पर based होते हैं. Stream means data का flow. जब data keyboard से program में आता है तो यह input stream कहलाता है. जब data program से monitor पर जाता है तो output stream कहलाता है.

- Input: keyboard, file, memory या other device से data लेना.
- Output: monitor, file, printer या other device पर data भेजना.
- C++ I/O is type-safe and uses classes and objects from header files like <iostream>, <fstream>, <sstream>.

## 1.1 Stream का Concept

Stream को पानी की धारा की तरह समझिए. Data bytes या characters के रूप में एक direction में flow करता है. Input stream data को source से program तक लाती है, output stream data को program से destination तक भेजती है.

```
Keyboard  --->  cin  --->  Program
Program   --->  cout --->  Monitor
File      --->  ifstream ---> Program
Program   --->  ofstream ---> File
```

## 1.2 Header Files for I/O

| Header File | Main Use | Important Classes/Objects |
|---|---|---|
| <iostream> | Standard input-output | cin, cout, cerr, clog, istream, ostream |
| <fstream> | File input-output | ifstream, ofstream, fstream |
| <sstream> | String-based stream input-output | istringstream, ostringstream, stringstream |
| <iomanip> | Formatting output | setw, setprecision, fixed, left, right |

# 2. I/O Classes in C++

C++ में I/O classes एक hierarchy बनाती हैं. इन classes की help से input और output operations perform किए जाते हैं. Most common classes are ios, istream, ostream, iostream, ifstream, ofstream and fstream.

## 2.1 Basic I/O Class Hierarchy

```
                ios
              /      \
         istream      ostream
             \          /
              iostream

For File Handling:
     istream  ---> ifstream
     ostream  ---> ofstream
     iostream ---> fstream
```

## 2.2 ios Class

ios class input-output system की base class है. यह formatting flags, error state, precision, width आदि को manage करती है. सभी stream classes indirectly ios से जुड़ी होती हैं.

- It stores stream state such as good, fail, bad and eof.
- It provides formatting facilities like width, precision and flags.
- It is a base class, so normally हम इसका object directly use नहीं करते.

### 2.3 istream Class

istream input stream class है. यह input लेने के लिए use होती है. cin object istream class का object है.

```
int age;
cin >> age;
```

### 2.4 ostream Class

ostream output stream class है. यह output display करने के लिए use होती है. cout object ostream class का object है.

```
cout << "Hello C++";
```

### 2.5 iostream Class

iostream class input और output दोनों operations support करती है. It is derived from both istream and ostream. इसका use उन streams में होता है जहाँ read और write दोनों चाहिए.

## 3. Standard I/O Objects

| Object | Class | Purpose | Example |
|---|---|---|---|
| cin | istream | Keyboard से input लेना | cin >> x; |
| cout | ostream | Normal output display करना | cout << x; |
| cerr | ostream | Error message display करना, unbuffered | cerr << "Error"; |
| clog | ostream | Log message display करना, buffered | clog << "Log"; |

### 3.1 cin and cout Example

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

int main() {
    int roll;
    string name;

    cout << "Enter roll number: ";
    cin >> roll;

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

    cout << "Roll No: " << roll << endl;
    cout << "Name: " << name << endl;

    return 0;
```

```
}
```

Explanation: cin extraction operator >> use करता है और cout insertion operator << use करता है.

# 4. File and Stream Classes

File handling में data को permanent storage में store किया जाता है. C++ में file handling के लिए <fstream> header use होता है. इसमें three important classes होती हैं: ifstream, ofstream, fstream.

| Class | Meaning | Use |
|---|---|---|
| ifstream | Input file stream | File से data read करना |
| ofstream | Output file stream | File में data write करना |
| fstream | File stream | File में read और write दोनों करना |

## 4.1 ofstream: Writing Data to File

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

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

    fout << "Roll No: 101" << endl;
    fout << "Name: Amit" << endl;

    fout.close();
    cout << "Data written successfully";

    return 0;
}
```

यह program student.txt file create करता है और उसमें data write करता है.

## 4.2 ifstream: Reading Data from File

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

int main() {
    ifstream fin;
    string line;

    fin.open("student.txt");

    while (getline(fin, line)) {
        cout << line << endl;
    }

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

यह program file से line by line data read करके screen पर display करता है.

### 4.3 fstream: Reading and Writing Both

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

int main() {
    fstream file;
    file.open("data.txt", ios::in | ios::out | ios::app);

    file << "New line added" << endl;

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

## 5. Opening and Closing Files

File open करने के दो common तरीके हैं: constructor method और open() function method.

### 5.1 Opening File using Constructor

```
ofstream fout("abc.txt");
ifstream fin("abc.txt");
fstream file("abc.txt", ios::in | ios::out);
```

### 5.2 Opening File using open() Function

```
ofstream fout;
fout.open("abc.txt");
```

### 5.3 Closing File

File use करने के बाद close करना good practice है. close() function file और program के बीच connection को release करता है.

```
fout.close();
fin.close();
file.close();
```

### 5.4 File Open Modes

| Mode | Meaning | Use |
|---|---|---|
| ios::in | Input mode | File read करने के लिए |
| ios::out | Output mode | File write करने के लिए |
| ios::app | Append mode | File के end में data जोड़ने के लिए |
| ios::ate | At end mode | Open होते ही pointer end पर जाता है |
| ios::trunc | Truncate mode | Existing file content delete हो जाता है |
| ios::binary | Binary mode | Binary file read/write करने के लिए |

### 5.5 Checking File Open Successfully

```
ifstream fin("student.txt");
```

```
if (!fin) {
    cout << "File could not be opened";
} else {
    cout << "File opened successfully";
}
```

# 6. Character I/O in File Handling

Character I/O में file से एक-एक character read/write किया जाता है. इसके लिए put() और get() functions use होते हैं.

## 6.1 Writing Character using put()

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

int main() {
    ofstream fout("char.txt");

    fout.put('A');
    fout.put('\n');
    fout.put('B');

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

## 6.2 Reading Character using get()

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

int main() {
    ifstream fin("char.txt");
    char ch;

    while (fin.get(ch)) {
        cout << ch;
    }

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

## 6.3 Use of eof()

eof() means end of file. यह true return करता है जब file का end reach हो जाता है. But best practice है कि while(fin.get(ch)) style use किया जाए because it checks reading success directly.

# 7. String I/O in C++

String I/O में हम complete line या text file read/write करते हैं. इसके लिए getline() बहुत important function है.

## 7.1 Writing String to File

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

int main() {
    ofstream fout("message.txt");

    string msg = "Welcome to C++ File Handling";
    fout << msg << endl;

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

## 7.2 Reading String using getline()

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

int main() {
    ifstream fin("message.txt");
    string line;

    while (getline(fin, line)) {
        cout << line << endl;
    }

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

## 7.3 cin vs getline

| cin | getline() |
|---|---|
| Space आने पर input stop कर देता है | Complete line read करता है |
| Single word input के लिए useful | Full name/address/message input के लिए useful |
| Example: cin >> name; | Example: getline(cin, name); |

# 8. Object I/O in C++

Object I/O का मतलब है class object को file में store करना और file से वापस read करना. Object I/O binary mode में अधिक useful होता है. इसके लिए write() और read() functions use होते हैं.

## 8.1 Writing Object to File

```
#include <iostream>
```

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

class Student {
public:
    int roll;
    char name[30];

    void getData() {
        cout << "Enter roll: ";
        cin >> roll;
        cout << "Enter name: ";
        cin >> name;
    }

    void showData() {
        cout << "Roll: " << roll << endl;
        cout << "Name: " << name << endl;
    }
};

int main() {
    Student s;
    s.getData();

    ofstream fout("student.dat", ios::binary);
    fout.write((char*)&s, sizeof(s));
    fout.close();

    return 0;
}
```

## 8.2 Reading Object from File

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

class Student {
public:
    int roll;
    char name[30];

    void showData() {
        cout << "Roll: " << roll << endl;
        cout << "Name: " << name << endl;
    }
};

int main() {
    Student s;

    ifstream fin("student.dat", ios::binary);
```

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

    s.showData();
    return 0;
}
```

### 8.3 Important Note on Object I/O

Object binary I/O simple classes के लिए useful है, लेकिन modern C++ में string, pointer, dynamic memory वाले classes को direct write/read करना risky हो सकता है. ऐसे cases में serialization techniques use करनी चाहिए.

## 9. File Pointer in C++

File pointer file के अंदर current position को indicate करता है. जब हम file read या write करते हैं, pointer automatically आगे move होता है. C++ में दो file pointers होते हैं: get pointer and put pointer.

### 9.1 Get Pointer and Put Pointer

| Pointer | Purpose | Functions |
|---|---|---|
| Get pointer | File से read करने की current position | seekg(), tellg() |
| Put pointer | File में write करने की current position | seekp(), tellp() |

### 9.2 tellg() and tellp()

tellg() current get pointer position बताता है. tellp() current put pointer position बताता है.

```
ifstream fin("data.txt");
cout << "Current read position: " << fin.tellg();
```

### 9.3 seekg() and seekp()

seekg() get pointer को move करता है. seekp() put pointer को move करता है.

```
fin.seekg(0);           // move read pointer to beginning
fout.seekp(0);          // move write pointer to beginning
fin.seekg(10, ios::beg); // move 10 bytes from beginning
fin.seekg(-5, ios::end); // move 5 bytes before end
```

### 9.4 File Pointer Direction Constants

| Constant | Meaning |
|---|---|
| ios::beg | Beginning of file |
| ios::cur | Current position |
| ios::end | End of file |

### 9.5 Example: Finding File Size

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

int main() {
    ifstream fin("data.txt", ios::binary);
```

```
    fin.seekg(0, ios::end);
    cout << "File size: " << fin.tellg() << " bytes";

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

# 10. Formatted I/O

Formatted I/O में output को proper width, precision, alignment में display किया जाता है. इसके लिए <iomanip> header file use होती है.

## 10.1 setw(), setprecision(), fixed

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

int main() {
    double marks = 89.56789;

    cout << setw(10) << "Marks" << endl;
    cout << fixed << setprecision(2) << marks;

    return 0;
}
```

# 11. Exception Handling in C++

Exception Handling एक mechanism है जिससे program runtime errors को handle कर सकता है और abnormal termination से बच सकता है. Exception means abnormal condition during program execution.

## 11.1 Definition

Exception handling is a technique used to handle runtime errors so that normal flow of the program can be maintained.

Hindi: Exception handling ऐसी technique है जिससे runtime error को safely handle करके program को crash होने से बचाया जाता है.

## 11.2 Need of Exception Handling

- Program को sudden crash होने से बचाता है.
- Runtime errors को proper message के साथ handle करता है.
- Code को clean और manageable बनाता है.
- Error handling logic और normal logic को अलग करता है.

## 11.3 Common Runtime Errors

| Error | Example |
|---|---|
| Divide by zero | a / b where b = 0 |

| | |
|---|---|
| File not found | Opening missing file |
| Invalid input | User enters wrong data |
| Memory allocation failure | new fails to allocate memory |
| Array index problem | Accessing out of range index |

# 12. Exception Handling Keywords

C++ exception handling में three main keywords use होते हैं: try, catch and throw. कुछ syllabi में throws keyword लिखा होता है; C++ में standard keyword throw है. Java में throws keyword commonly use होता है. C++ में exception specification पुराने versions में throw(type) के रूप में मिलती थी, but modern C++ में generally use नहीं करते.

| Keyword | Meaning | Use |
|---|---|---|
| try | Risky code block | जहाँ exception आ सकता है |
| throw | Exception generate करना | Error condition आने पर value/object throw करना |
| catch | Exception handle करना | Thrown exception को receive करके handle करना |
| throws | C++ keyword नहीं; Java concept | Syllabus में लिखा हो तो throw/exception specification की context में समझें |

## 12.1 Basic Syntax

```
try {
    // risky code
    throw value;
}
catch (type variable) {
    // handling code
}
```

## 12.2 Simple Example: Divide by Zero

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

int main() {
    int a = 10, b = 0;

    try {
        if (b == 0) {
            throw b;
        }
        cout << "Result: " << a / b;
    }
    catch (int x) {
        cout << "Error: Division by zero is not allowed";
    }

    return 0;
```

```
}
```

### 12.3 Explanation of try-catch-throw

- try block में risky code लिखा जाता है.
- जब error condition मिलती है, throw keyword exception throw करता है.
- catch block उस exception को receive करके proper action लेता है.
- अगर exception catch हो जाए तो program crash नहीं होता.

## 13. Multiple Catch Blocks

एक try block के साथ multiple catch blocks लिखे जा सकते हैं. अलग-अलग type के exceptions को अलग-अलग catch block handle कर सकते हैं.

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

int main() {
    try {
        throw 10.5;
    }
    catch (int x) {
        cout << "Integer exception";
    }
    catch (double d) {
        cout << "Double exception";
    }
    catch (char c) {
        cout << "Character exception";
    }

    return 0;
}
```

## 14. Catch All Handler

catch(...) सभी types के exceptions को catch कर सकता है. इसे catch-all handler कहते हैं.

```
try {
    throw "Unknown error";
}
catch (...) {
    cout << "Some exception occurred";
}
```

## 15. Nested try-catch

जब एक try block के अंदर दूसरा try-catch block होता है, तो इसे nested try-catch कहते हैं.

```
try {
    try {
```

```
            throw 5;
        }
        catch (int x) {
            cout << "Inner catch" << endl;
            throw;   // rethrow same exception
        }
    }
    catch (int x) {
        cout << "Outer catch" << endl;
    }
```

## 16. Exception Handling with Functions

Exception किसी function से भी throw हो सकती है और calling function में catch हो सकती है.

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

void checkAge(int age) {
    if (age < 18) {
        throw age;
    }
    cout << "Eligible";
}

int main() {
    try {
        checkAge(15);
    }
    catch (int age) {
        cout << "Not eligible. Age is below 18";
    }

    return 0;
}
```

## 17. Standard Exceptions

C++ में standard exception classes <exception> header में मिलती हैं. Base class std::exception है. what() function error message return करता है.

| Exception Class | Meaning |
|---|---|
| exception | Base class for standard exceptions |
| bad_alloc | Memory allocation failure |
| out_of_range | Index/range error |
| runtime_error | Runtime error |
| invalid_argument | Invalid argument passed |

### 17.1 Example using std::exception

```
#include <iostream>
```

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

int main() {
    try {
        int *p = new int[100000000000];
    }
    catch (exception &e) {
        cout << "Exception: " << e.what();
    }

    return 0;
}
```

## 18. User-Defined Exception

User-defined exception में हम अपनी class बनाकर exception throw कर सकते हैं.

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

class LowBalanceException {
public:
    void showMessage() {
        cout << "Error: Low balance in account";
    }
};

int main() {
    int balance = 500;
    int withdraw = 1000;

    try {
        if (withdraw > balance) {
            throw LowBalanceException();
        }
        balance = balance - withdraw;
    }
    catch (LowBalanceException e) {
        e.showMessage();
    }

    return 0;
}
```

## 19. Benefits and Limitations of Exception Handling

| Benefits | Limitations |
|---|---|
| Program crash होने से बच सकता है | हर error exception से handle नहीं होती |
| Code readable बनता है | गलत use करने पर program complex हो सकता है |
| Error handling centralized होता है | Performance overhead हो सकता है |

| | |
|---|---|
| Runtime error को graceful तरीके से handle करता है | Compile-time errors को handle नहीं करता |

# 20. Standard Template Library and Templates

Syllabus में Standard Template Library और Template दोनों दिए हैं. STL C++ की powerful library है जो generic classes और functions provide करती है. Template C++ की वह feature है जिससे generic programming possible होती है.

## 20.1 Template Definition

Template is a feature of C++ that allows writing generic functions and classes. Generic means same code can work with different data types.

Hindi: Template की help से हम ऐसा code लिख सकते हैं जो int, float, double, char जैसे different data types पर काम कर सके.

## 20.2 Need of Template

Without template हमें same logic के लिए अलग-अलग functions लिखने पड़ते हैं. Template code duplication को reduce करता है.

```
int add(int a, int b) { return a + b; }
float add(float a, float b) { return a + b; }
double add(double a, double b) { return a + b; }
```

Template से same logic एक बार लिखा जा सकता है.

# 21. Function Template

Function template generic function बनाता है. Compiler data type के according function generate करता है.

## 21.1 Syntax

```
template <class T>
T functionName(T a, T b) {
    // code
}
```

## 21.2 Function Template Example

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

template <class T>
T add(T a, T b) {
    return a + b;
}

int main() {
    cout << add(10, 20) << endl;
    cout << add(5.5, 2.5) << endl;
```

```
    return 0;
}
```

### 21.3 Template with Multiple Parameters

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

template <class T1, class T2>
void display(T1 a, T2 b) {
    cout << a << " " << b << endl;
}

int main() {
    display(10, "Amit");
    display(12.5, 'A');
    return 0;
}
```

## 22. Class Template

Class template generic class बनाता है. Same class different data types के साथ use की जा सकती है.

### 22.1 Class Template Example

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

template <class T>
class Box {
private:
    T value;

public:
    Box(T v) {
        value = v;
    }

    void show() {
        cout << "Value: " << value << endl;
    }
};

int main() {
    Box<int> b1(100);
    Box<string> b2("C++ Template");

    b1.show();
    b2.show();

    return 0;
}
```

# 23. Standard Template Library (STL)

STL is a collection of template classes and functions. It provides ready-made data structures and algorithms. STL reduces development time and improves reliability.

## 23.1 Main Components of STL

| Component | Meaning | Examples |
|---|---|---|
| Containers | Data store करने वाली template classes | vector, list, stack, queue, map |
| Algorithms | Data पर operations perform करते हैं | sort, search, reverse, count |
| Iterators | Containers के elements access करने के लिए pointer-like objects | begin(), end() |
| Function Objects | Function की तरह behave करने वाले objects | greater<int>() |

## 23.2 Common STL Containers

| Container | Use |
|---|---|
| vector | Dynamic array |
| list | Doubly linked list |
| stack | LIFO structure |
| queue | FIFO structure |
| map | Key-value pair storage |
| set | Unique sorted values |

## 23.3 Vector Example

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

int main() {
    vector<int> v;

    v.push_back(10);
    v.push_back(20);
    v.push_back(30);

    for (int x : v) {
        cout << x << " ";
    }

    return 0;
}
```

## 23.4 STL Algorithm Example: sort()

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

```
int main() {
    vector<int> v = {40, 10, 30, 20};

    sort(v.begin(), v.end());

    for (int x : v) {
        cout << x << " ";
    }

    return 0;
}
```

## 24. Template vs STL

| Basis | Template | STL |
|---|---|---|
| Meaning | Generic programming feature | Ready-made template-based library |
| Purpose | Generic function/class बनाना | Data structures and algorithms provide करना |
| Example | template <class T> | vector<int>, sort() |
| Created by | Programmer can create | C++ standard library provides |

## 25. Complete Program: File Handling + Exception Handling

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

int main() {
    ifstream fin;

    try {
        fin.open("student.txt");

        if (!fin) {
            throw "File not found";
        }

        string line;
        while (getline(fin, line)) {
            cout << line << endl;
        }
        fin.close();
    }
    catch (const char *msg) {
        cout << "Exception: " << msg;
    }

    return 0;
}
```

# 26. Important University Exam Definitions

**Stream:** Stream is a sequence of bytes or characters flowing between program and device. Hindi: Stream data flow को represent करती है.

**ifstream:** ifstream is a file input stream class used to read data from files.

**ofstream:** ofstream is a file output stream class used to write data into files.

**fstream:** fstream is used for both reading and writing files.

**File Pointer:** File pointer indicates current read or write position inside a file.

**Exception:** Exception is an abnormal condition that occurs during program execution.

**Exception Handling:** Exception handling is a mechanism to handle runtime errors using try, catch and throw.

**Template:** Template is a C++ feature used to write generic functions and classes.

**STL:** STL is a standard library of template-based containers, algorithms and iterators.

# 27. Important Long Answer Questions

## Q1. Explain file handling in C++ with file stream classes.

File handling in C++ is used to store data permanently in files. C++ provides <fstream> header for file handling. The main file stream classes are ifstream, ofstream and fstream. ifstream is used to read data from a file, ofstream is used to write data into a file, and fstream is used for both reading and writing. Files can be opened using constructor or open() function. After completing operations, files should be closed using close() function. Various file modes such as ios::in, ios::out, ios::app, ios::binary and ios::trunc are used according to requirement.

## Q2. Explain character I/O and string I/O in C++.

Character I/O means reading or writing one character at a time. In C++, put() is used to write a character into a file and get() is used to read a character from a file. String I/O means reading and writing complete text or lines. getline() is used to read a complete line including spaces. Character I/O is useful when processing file character by character, while string I/O is useful for line-by-line text processing.

## Q3. Explain file pointers in C++.

A file pointer represents the current position inside a file. C++ maintains two file pointers: get pointer and put pointer. The get pointer is used during reading operations, and the put pointer is used during writing operations. Functions tellg() and tellp() return the current position of get and put pointers respectively. Functions seekg() and seekp() are used to move these pointers to a specific position in the file. File pointer movement can be done from ios::beg, ios::cur and ios::end.

## Q4. Explain exception handling with try, catch and throw.

Exception handling is a mechanism to handle runtime errors. The try block contains risky code where an exception may occur. The throw keyword is used to throw an exception when an abnormal condition occurs. The catch block receives and handles the thrown exception. This mechanism prevents abnormal termination

of a program and maintains normal flow. Multiple catch blocks can be used to handle different types of exceptions.

### Q5. Explain templates in C++ with example.

Templates allow generic programming in C++. A template enables a function or class to work with different data types without rewriting the same code. Function templates are used to create generic functions, while class templates are used to create generic classes. Templates reduce code duplication and increase reusability. STL is based on templates and provides ready-made generic containers and algorithms.

## 28. Short Questions with Answers

**What is stream?** Stream is a flow of data between program and device.

**Which header is used for file handling?** <fstream> header is used for file handling.

**What is ifstream?** ifstream is used to read data from a file.

**What is ofstream?** ofstream is used to write data into a file.

**What is fstream?** fstream is used for both reading and writing files.

**Which function closes a file?** close() function closes a file.

**What is ios::app?** It opens file in append mode.

**What is seekg()?** seekg() moves the get/read pointer.

**What is seekp()?** seekp() moves the put/write pointer.

**What is exception?** Exception is an abnormal runtime condition.

**Name three exception handling keywords.** try, catch and throw.

**What is template?** Template is used to write generic code.

**What is STL?** STL is Standard Template Library providing containers, algorithms and iterators.

## 29. Important MCQs

1. Which header file is used for standard input-output in C++?

A. <iostream>

B. <fstream>

C. <string>

D. <math>

**Answer: A**

2. Which class is used to read data from file?

A. ofstream

B. ifstream

C. ostream

D. iomanip

**Answer: B**

3. Which class is used to write data into file?

A. ifstream

B. ofstream

C. istream

D. sstream

**Answer: B**

4. Which mode is used to append data into file?

A. ios::in

B. ios::out

C. ios::app

D. ios::binary

**Answer: C**

5. Which function is used to close a file?

A. open()

B. close()

C. exit()

D. end()

**Answer: B**

6. Which function reads one character from file?

A. put()

B. get()

C. readline()

D. writechar()

**Answer: B**

7. Which keyword is used to throw an exception?

A. try

B. catch

C. throw

D. error

**Answer: C**

8. Which block handles exception?

A. try

B. catch

C. throw

D. main

**Answer: B**

9. Runtime errors are handled by:

A. Loop

B. Array

C. Exception handling

D. Pointer only

**Answer: C**

10. Template is used for:

A. Generic programming

B. Only file handling

C. Only exception handling

D. Only inheritance

**Answer: A**

11. STL stands for:

A. Standard Type Library

B. Standard Template Library

C. System Template Library

D. Simple Template Logic

**Answer: B**

12. Which STL container behaves like dynamic array?

A. stack

B. queue

C. vector

D. map

**Answer: C**

## 30. Quick Revision Points

- C++ I/O is based on stream concept.
- cin is used for input and cout is used for output.
- <fstream> header provides file handling classes.
- ifstream reads from file; ofstream writes to file; fstream does both.
- File should be closed using close() after operations.
- put() and get() are used for character I/O.
- getline() is used for line/string input.
- Object I/O can be done using read() and write() in binary mode.
- File pointers are controlled by seekg(), seekp(), tellg() and tellp().
- Exception handling uses try, catch and throw.
- Templates support generic programming.
- STL provides containers, algorithms and iterators.

## 31. Best Exam Summary

Input-output and file handling in C++ are based on stream classes. Standard I/O uses cin and cout, while file I/O uses ifstream, ofstream and fstream. File handling allows permanent storage of data and supports character, string and object input-output. File pointers help in random access by moving read and write positions. Exception handling provides a safe way to handle runtime errors using try, catch and throw. Templates provide generic programming, and STL offers ready-made template-based containers and algorithms for efficient programming.