**Assignment 5: C++ Streams and File Handling**

1. What are streams in C++ and why are they important?

Ans Streams in C++ are sequences of data (bytes or characters) used for input and output operations. They provide a uniform interface to interact with various devices (e.g., console, files).

Importance:

* Enable consistent handling of I/O operations across different sources (console, files, strings).
* Support formatted and unformatted I/O.
* Simplify complex I/O operations with a high-level abstraction.

2. Explain the different types of streams in C++.

Ans The different types of streams in C++ are:

* Input Streams: Handle data input (e.g., cin, ifstream).
* Output Streams: Handle data output (e.g., cout, ofstream).
* Bidirectional Streams: Support both input and output (e.g., fstream, stringstream).
* File Streams: Manage file I/O (ifstream, ofstream, fstream).
* String Streams: Manipulate strings as streams (istringstream, ostringstream, stringstream).

3. How do input and output streams differ in C++?

Ans Difference:

| **Feature** | **Input Stream** | **Output Stream** |
| --- | --- | --- |
| **Purpose** | To receive data into the program | To send data out from the program |
| **Base Class** | istream | ostream |
| **Common Object** | cin | cout |
| **Operator Used** | >> (extraction operator) | << (insertion operator) |
| **Example Usage** | std::cin >> variable; | std::cout << "Text"; |
| **Data Direction** | From input device (keyboard, file) → program | Program → output device (screen, file) |
| **Error Handling Method** | cin.fail() | cout.fail() |

4. Describe the role of the iostream library in C++.

Ans The iostream library provides classes and objects for handling input and output operations:

Objects: cin (input), cout (output), cerr (error output), clog (log output).

Classes: istream (input operations), ostream (output operations), iostream (bidirectional).

Role: Facilitates console I/O, supports formatted/unformatted operations, and serves as the foundation for other stream libraries (e.g., fstream, sstream).

5. What is the difference between a stream and a file stream?

Ans Stream: A general abstraction for sequential data flow, used for any I/O (e.g., console, strings, files). Examples: cin, cout, stringstream.

File Stream: A specific type of stream for file I/O operations. Examples: ifstream (read), ofstream (write), fstream (read/write).

Difference: File streams are specialized for file operations, while streams can handle various data sources.

6. What is the purpose of the cin object in C++?

Ans The cin object, defined in the iostream library, is used for reading input from the standard input device (usually the keyboard). It supports formatted input using the extraction operator (>>).

7. How does the cin object handle input operations?

Ans cin reads input from the keyboard, parses it based on the target variable’s type, and stores it. It uses the >> operator and skips whitespace by default.

8. What is the purpose of the cout object in C++?

Ans The cout object, defined in the iostream library, is used for writing output to the standard output device (usually the console). It supports formatted output using the insertion operator (<<).

9. How does the cout object handle output operations?

Ans cout writes data to the console, converting variables to their string representation. It uses the << operator and supports manipulators for formatting.

10. Explain the use of the insertion (<<) and extraction (>>) operators in conjunction with cin and cout.

Ans Insertion Operator (<<): Used with cout to insert data into the output stream (e.g., cout << "Hello" << x;). Chains multiple outputs.

Extraction Operator (>>): Used with cin to extract data from the input stream into variables (e.g., cin >> x >> y;). Chains multiple inputs.

11. What are the main C++ stream classes and their purposes?

Ans istream: Handles input operations (e.g., cin).

ostream: Handles output operations (e.g., cout).

iostream: Supports bidirectional I/O (inherits from istream and ostream).

ifstream: Reads from files.

ofstream: Writes to files.

fstream: Reads and writes to files.

istringstream: Reads from strings.

ostringstream: Writes to strings.

stringstream: Reads and writes to strings.

12. Explain the hierarchy of C++ stream classes.

Ans The C++ stream classes form a hierarchy:

* ios\_base: Base class with common formatting and state flags.
* ios: Inherits from ios\_base, adds error handling and stream state.
* istream: Inherits from ios, handles input operations.
* ostream: Inherits from ios, handles output operations.
* iostream: Inherits from istream and ostream, supports bidirectional I/O.
* ifstream, ofstream, fstream: Inherit from istream, ostream, or iostream for file operations.
* istringstream, ostringstream, stringstream: Inherit similarly for string operations.

13. What is the role of the istream and ostream classes?

Ans istream: Provides methods for input operations (e.g., get(), getline(), >>). Used by cin, ifstream, etc.

ostream: Provides methods for output operations (e.g., put(), write(), <<). Used by cout, ofstream, etc.

Role: Serve as base classes for specific input/output stream classes.

14. Describe the functionality of the ifstream and ofstream classes.

Ans ifstream (Input File Stream): Reads data from files. Inherits from istream. Supports methods like get(), getline(), and read().

ofstream (Output File Stream): Writes data to files. Inherits from ostream. Supports methods like put(), write(), and <<.

15. How do the fstream and stringstream classes differ from other stream classes?

Ans fstream: Supports both reading and writing to files. Unlike ifstream (read-only) or ofstream (write-only), it allows bidirectional file operations.

stringstream: Reads and writes to strings instead of files or console. Unlike file/console streams, it manipulates in-memory strings.

Difference: fstream is file-specific, stringstream is string-specific, while other streams focus on single-direction I/O or console.

16. What is unformatted I/O in C++?

Ans Unformatted I/O involves reading or writing raw data without applying formatting (e.g., no whitespace skipping, no type conversion). It deals with characters or bytes directly.

Examples: get(), put(), read(), write().

17. Provide examples of unformatted I/O functions.

Answer:

#include <iostream>

using namespace std;

int main() {

// Unformatted input

char ch;

cout << "Enter a character: ";

cin.get(ch); // Reads a single character, including whitespace

cout << "You entered: " << ch << endl;

// Unformatted output

cout.put('A'); // Writes a single character

cout.put('\n');

return 0;

}

18. What is formatted I/O in C++?

Ans Formatted I/O involves reading or writing data with specific formatting, such as type conversion, alignment, or precision. It uses operators (>>, <<) and manipulators (e.g., setw, setprecision).

Example: cout << setw(10) << 42; formats output with width.

19. How do you use manipulators to perform formatted I/O in C++?

Ans Manipulators are functions or objects that modify stream behavior (e.g., formatting, alignment). They are used with << or >> and often require <iomanip>. Detail in 23 answer.

20. Explain the difference between unformatted and formatted I/O operations.

Ans Here is a tabular difference between unformatted and formatted I/O operations in C++:

| **Feature** | **Unformatted I/O** | **Formatted I/O** |
| --- | --- | --- |
| **Definition** | Raw input/output without specific formatting | Input/output with format control |
| **Operators/Functions Used** | get(), put(), getline(), read(), write() | >>, << |
| **Data Type Handling** | Handles data as characters or bytes | Handles data as specific types (int, float, etc.) |
| **Formatting Control** | No control over formatting | Allows control over width, precision, alignment, etc. |
| **Complexity** | Simple and low-level | More user-friendly and readable |
| **Example** | cin.get(ch);, cout.put(ch); | cin >> x;, cout << "Value: " << x; |
| **Use Case** | Reading/writing single characters or blocks | Displaying variables with readable formatting |

21. What are manipulators in C++?

Ans Manipulators are functions or objects that modify the behavior of streams, such as formatting output (e.g., width, precision) or controlling input. They are used with << or >>.

Examples: setw, setprecision, fixed, endl.

22. How do manipulators modify the behavior of I/O operations?

Ans Manipulators change stream state or formatting:

Width: setw(n) sets field width for the next output.

Precision: setprecision(n) sets decimal places for floating-point numbers.

Flags: fixed, scientific control number format.

Others: endl adds a newline and flushes the stream.They are inserted into the stream using << (e.g., cout << setw(5) << x;).

23. Provide examples of commonly used manipulators in C++.

Answer:

#include <iostream>

#include <iomanip>

using namespace std;

int main() {

// setw: Set field width

cout << setw(10) << "Hello" << endl; // Right-aligned in 10 spaces

// setprecision: Set decimal precision

double pi = 3.14159;

cout << setprecision(3) << pi << endl; // Prints 3.14

// fixed: Fixed-point notation

cout << fixed << setprecision(2) << pi << endl; // Prints 3.14

// endl: Newline and flush

cout << "End of line" << endl;

return 0;

}

24. Explain the use of the setw, setprecision, and fixed manipulators.

Ans setw(n): Sets the width of the next output field to n characters. Defined in <iomanip>.

setprecision(n): Sets the number of digits for floating-point output (total or decimal places, depending on format). Defined in <iomanip>.

fixed: Forces floating-point numbers to use fixed-point notation (e.g., 123.45 instead of scientific). Defined in <ios>.

25. How do you create custom manipulators in C++?

Ans Custom manipulators are functions that take a stream as an argument and modify its behavior. They return the stream for chaining.

Example:

#include <iostream>

using namespace std;

ostream& myManip(ostream& os) {

os << "==Custom==";

return os;

}

int main() {

cout << myManip << " Hello" << endl; // Prints ==Custom== Hello

return 0;

}

26. What is a file stream in C++ and how is it used?

Ans A file stream is a stream object used for file input/output operations. Classes include ifstream (read), ofstream (write), and fstream (read/write).It is use to open a file, perform read/write operations, and close the file.

27. Explain the process of opening and closing files using file streams.

Ans Opening: Use the stream constructor or open() method with the filename and mode (e.g., ios::in, ios::out).

Closing: Call close() to release resources, or it happens automatically when the stream object is destroyed.

Example:

#include <fstream>

using namespace std;

int main() {

ofstream outFile;

outFile.open("example.txt"); // Open file

outFile << "Hello";

outFile.close(); // Close file

return 0;

}

28. Describe the different modes in which a file can be opened.

Ans File modes (in std::ios):

ios::in: Open for reading.

ios::out: Open for writing.

ios::app: Append data to the end.

ios::trunc: Truncate file if it exists.

ios::ate: Seek to the end after opening.

ios::binary: Open in binary mode.Note: Combine modes with | (e.g., ios::in | ios::out).

29. How do you read from and write to files using file streams?

Ans Reading: Use ifstream or fstream with getline(), get(), read(), or >>.

Writing: Use ofstream or fstream with put(), write(), or <<.

30. Provide an example of using file streams to copy the contents of one file to another.

Answer:

#include <iostream>

#include <fstream>

using namespace std;

int main() {

ifstream inFile("source.txt"); // Open source file

ofstream outFile("destination.txt"); // Open destination file

if (!inFile || !outFile) {

cout << "Error opening files!" << endl;

return 1;

}

char ch;

while (inFile.get(ch)) {

outFile.put(ch); // Copy character by character

}

cout << "File copied successfully!" << endl;

inFile.close();

outFile.close();

return 0;

}

31. What are the main C++ file stream classes and their purposes?

Ans The main C++ file stream classes and their purposes are:

ifstream: Reads data from files (input).

ofstream: Writes data to files (output).

fstream: Supports both reading and writing to files.Purpose: Provide a high-level interface for file I/O operations.

32. Explain the role of the ifstream, ofstream, and fstream classes.

Ans Their roles are:

ifstream: Inherits from istream, used for reading file contents (e.g., getline(), read()).

ofstream: Inherits from ostream, used for writing to files (e.g., write(), <<).

fstream: Inherits from iostream, supports both reading and writing (e.g., simultaneous file access).

33. How do you use the ifstream class to read data from a file?

Ans Open the file with ifstream and use methods like getline(), get(), or >> to read data.

Example:

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

int main() {

ifstream inFile("data.txt");

if (!inFile) {

cout << "Error opening file!" << endl;

return 1;

}

string line;

while (getline(inFile, line)) {

cout << line << endl;

}

inFile.close();

return 0;

}

34. How do you use the ofstream class to write data to a file?

Ans Open the file with ofstream and use <<, put(), or write() to write data.

Example:

#include <iostream>

#include <fstream>

using namespace std;

int main() {

ofstream outFile("output.txt");

if (!outFile) {

cout << "Error opening file!" << endl;

return 1;

}

outFile << "Hello, File!" << endl;

outFile.close();

return 0;

}

35. Describe the functionality of the fstream class for both input and output operations.

Answer:The fstream class supports simultaneous reading and writing to a file. It requires appropriate modes (ios::in, ios::out) and supports methods like read(), write(), getline(), and <<.Example:

#include <iostream>

#include <fstream>

using namespace std;

int main() {

fstream file("data.txt", ios::in | ios::out | ios::trunc);

if (!file) {

cout << "Error opening file!" << endl;

return 1;

}

file << "Test Data";

file.seekg(0, ios::beg);

string line;

getline(file, line);

cout << "Read: " << line << endl;

file.close();

return 0;

}

36. What are file management functions in C++?

Ans File management functions manipulate files or file pointers:

remove(): Deletes a file.

rename(): Renames or moves a file.

seekg(): Sets the read pointer position.

seekp(): Sets the write pointer position.

tellg(): Gets the read pointer position.

tellp(): Gets the write pointer position.

37. How do you use the remove and rename functions to manage files?

Ans remove(const char filename)\*: Deletes the specified file.

rename(const char oldname, const char newname)\*\*: Renames a file.

Example:

#include <iostream>

#include <cstdio>

using namespace std;

int main() {

if (rename("old.txt", "new.txt") == 0) {

cout << "File renamed!" << endl;

} else {

cout << "Rename failed!" << endl;

}

if (remove("new.txt") == 0) {

cout << "File deleted!" << endl;

} else {

cout << "Delete failed!" << endl;

}

return 0;

}

38. Explain the purpose of the seekg and seekp functions in file management.

Ans seekg(offset, direction): Moves the input (read) pointer to a specified position.

seekp(offset, direction): Moves the output (write) pointer to a specified position.

Purpose: Enable random access by positioning the file pointer for reading or writing.Directions: ios::beg (beginning), ios::cur (current), ios::end (end).

39. Provide examples of using file management functions to manipulate file pointers.

Answer:

#include <iostream>

#include <fstream>

using namespace std;

int main() {

fstream file("test.txt", ios::in | ios::out | ios::trunc);

if (!file) {

cout << "Error opening file!" << endl;

return 1;

}

file << "Hello, World!";

file.seekg(7, ios::beg); // Move read pointer to 'W'

char ch;

file.get(ch);

cout << "Character: " << ch << endl; // Prints W

file.seekp(7, ios::beg); // Move write pointer to 'W'

file << "C++";

file.seekg(0, ios::beg);

string line;

getline(file, line);

cout << "Updated: " << line << endl; // Prints Hello, C++ld!

file.close();

return 0;

}

40. What are file modes in C++?

Ans File modes specify how a file is opened and used (e.g., read, write, append). They are flags passed to file stream constructors or open().

41. Describe the different file modes available in C++.

Ans the different file modes available in C++ are:

ios::in: Open for reading.

ios::out: Open for writing.

ios::app: Append data to the end.

ios::trunc: Truncate file if it exists.

ios::ate: Seek to the end after opening.

ios::binary: Open in binary mode (no text conversions).

Note: Combine modes with | (e.g., ios::in | ios::out).

42. How do you specify a file mode when opening a file?

Ans Pass the mode as the second argument to the constructor or open() method.

Example:

ifstream inFile("file.txt", ios::in);

ofstream outFile("file.txt", ios::out | ios::trunc);

fstream file("file.txt", ios::in | ios::out | ios::binary);

43. Explain the difference between binary and text file modes.

Ans Diffrence:

| **Feature** | **Text File Mode** | **Binary File Mode** |
| --- | --- | --- |
| **Storage Format** | Stores data as human-readable text | Stores data in raw binary format |
| **File Mode Flag** | "r", "w", "a" (or use "ios::in" etc.) | "rb", "wb", "ab" (or use "ios::binary") |
| **Data Size** | Data size may change (e.g., newline conversion) | Exact data size preserved |
| **Use Case** | For reading/writing text (e.g., .txt) | For images, audio, executables, etc. |
| **Efficiency** | Slower for large data | Faster and more efficient |

44. Provide examples of opening files in different modes using file streams.

Answer:

#include <iostream>

#include <fstream>

using namespace std;

int main() {

// Write in text mode, truncate

ofstream outFile("text.txt", ios::out | ios::trunc);

outFile << "Text data" << endl;

outFile.close();

// Append in text mode

ofstream appendFile("text.txt", ios::out | ios::app);

appendFile << "More data" << endl;

appendFile.close();

// Read in text mode

ifstream inFile("text.txt", ios::in);

string line;

getline(inFile, line);

cout << line << endl;

inFile.close();

// Write in binary mode

ofstream binFile("data.bin", ios::out | ios::binary);

int x = 12345;

binFile.write(reinterpret\_cast<char\*>(&x), sizeof(x));

binFile.close();

return 0;

}

45. What are binary files in C++ and how do they differ from text files?

Ans Binary Files: Store data as raw bytes without formatting or conversions. Used for non-text data (e.g., numbers, structs).

Text Files: Store human-readable text with platform-specific conversions (e.g., newline handling).Differences:

Binary files are compact and precise but not readable.

Text files are readable but may include formatting overhead.

46. Explain the process of reading from and writing to binary files.

Ans Writing: Use write() to store raw bytes of variables or structs in binary mode.

Reading: Use read() to retrieve bytes and cast to the original type.

Example:

#include <iostream>

#include <fstream>

using namespace std;

struct Data {

int id;

char name[20];

};

int main() {

// Write binary

ofstream outFile("data.bin", ios::out | ios::binary);

Data d1 = {1, "Alice"};

outFile.write(reinterpret\_cast<char\*>(&d1), sizeof(d1));

outFile.close();

// Read binary

ifstream inFile("data.bin", ios::in | ios::binary);

Data d2;

inFile.read(reinterpret\_cast<char\*>(&d2), sizeof(d2));

cout << "ID: " << d2.id << ", Name: " << d2.name << endl;

inFile.close();

return 0;

}

47. What are random access files in C++?

Ans Random access files allow reading or writing at any position in the file, not just sequentially. This is achieved using seekg() (read) and seekp() (write) to move the file pointer.

48. How do you perform random access operations on files?

Ans Open the file with fstream or appropriate stream, using ios::in, ios::out, or both.

Use seekg() or seekp() to move the file pointer to the desired position.

Read or write using read(), write(), or stream operators.

49. Provide examples of using file streams to implement random access in binary files.

Answer:

#include <iostream>

#include <fstream>

using namespace std;

struct Record {

int id;

char name[20];

};

int main() {

fstream file("records.bin", ios::in | ios::out | ios::binary | ios::trunc);

if (!file) {

cout << "Error opening file!" << endl;

return 1;

}

// Write two records

Record r1 = {1, "Alice"};

Record r2 = {2, "Bob"};

file.write(reinterpret\_cast<char\*>(&r1), sizeof(r1));

file.write(reinterpret\_cast<char\*>(&r2), sizeof(r2));

// Random access: Read second record

file.seekg(sizeof(r1), ios::beg);

Record r3;

file.read(reinterpret\_cast<char\*>(&r3), sizeof(r3));

cout << "ID: " << r3.id << ", Name: " << r3.name << endl; // Prints Bob

// Random access: Update first record

file.seekp(0, ios::beg);

Record r4 = {3, "Charlie"};

file.write(reinterpret\_cast<char\*>(&r4), sizeof(r4));

// Verify update

file.seekg(0, ios::beg);

file.read(reinterpret\_cast<char\*>(&r3), sizeof(r3));

cout << "Updated ID: " << r3.id << ", Name: " << r3.name << endl; // Prints Charlie

file.close();

return 0;

}