Great! Let's proceed to the topic of Input/Output in C++.

VI. Input/Output in C++

- **Streams:**
  - **Explanation:** In C++, I/O operations are performed using streams. A stream is an abstraction that represents a flow of data. Streams can be associated with various sources and destinations, such as the console (keyboard and screen), files, network connections, and memory buffers. The <iostream> header provides the basic I/O stream objects.
  - **Stream Classes:**
    - **std::istream:** Base class for input streams (e.g., reading from cin or a file).
    - **std::ostream:** Base class for output streams (e.g., writing to cout or a file).
    - **std::iostream:** Base class for streams that can perform both input and output.
    - **std::ifstream:** For reading from files.
    - **std::ofstream:** For writing to files.

        - **std::fstream:** For both reading from and writing to files.
        - **std::stringstream:** For I/O operations on strings in memory.
        - **std::istringstream:** For reading from strings in memory.
        - **std::ostringstream:** For writing to strings in memory.
- **Standard Streams:**
    - **std::cin:** Standard input stream, usually connected to the keyboard. Used for reading input from the user.

C++
```cpp
#include <iostream>
#include <string>

int main() {
    int age;
    std::cout << "Enter your age: ";
    std::cin >> age;
    std::cout << "You entered: " << age << std::endl;

    std::string name;
    std::cout << "Enter your name: ";
    std::cin >> name; // Reads only up to the first whitespace
    std::cout << "Your name is: " << name << std::endl;

    // To read a whole line including spaces:
    std::string fullName;
    std::cout << "Enter your full name: ";
    std::cin.ignore(); // Clear the newline character left in the buffer
```

```
  std::getline(std::cin, fullName);
  std::cout << "Your full name is: " << fullName << std::endl;

  return 0;
}
```

- **std::cout:** Standard output stream, usually connected to the console (screen). Used for displaying output to the user.

C++
```
#include <iostream>

int main() {
  int number = 123;
  double pi = 3.14159;
  std::string message = "Hello, C++!";

  std::cout << "The number is: " << number << std::endl;
  std::cout << "The value of pi is: " << pi << std::endl;
  std::cout << message << std::endl;

  return 0;
}
```

- **std::cerr:** Standard error stream, usually connected to the console. Used for displaying error messages. It is typically unbuffered, meaning output to cerr is flushed immediately.

C++
```
#include <iostream>
#include <stdexcept>

int main() {
  try {
    throw std::runtime_error("An error occurred!");
    } catch (const std::runtime_error& e) {
        std::cerr << "Error: " << e.what() << std::endl;
    }
    return 0;
}
```

- **std::clog:** Standard log stream, usually connected to the console. Also used for displaying error messages or logging information, but unlike cerr, it is typically buffered.

```C++
#include <iostream>

int main() {
    std::clog << "This is a log message." << std::endl;
    return 0;
}
```

- **Formatting Output:**
  - **Manipulators (from <iomanip> header):** Functions that modify the state of the output stream.
    - std::setw(n): Sets the width of the next output field to n characters.
    - std::setprecision(n): Sets the precision (number of digits after the decimal point) for floating-point numbers.
    - std::fixed: Forces floating-point numbers to

be displayed in fixed-point notation.

  - std::scientific: Forces floating-point numbers to be displayed in scientific notation.
  - std::left, std::right, std::internal: Adjust the alignment of output within the field[1] width.
  - std::setfill(c): Sets the fill character for padding within the field width.
  - std::hex, std::oct, std::dec: Set the base for integer output (hexadecimal, octal, decimal).
  - std::showbase, std::noshowbase: Display or hide the base prefix (e.g., 0x for hexadecimal).
  - std::uppercase, std::nouppercase: Display hexadecimal digits and scientific notation in uppercase or lowercase.
  - std::endl: Inserts a newline character and flushes the output stream.
  - std::flush: Flushes the output stream.
  - std::boolalpha, std::noboolalpha: Display boolean values as "true"/"false" or 1/0.

- **Example:**

C++
```
#include <iostream>
#include <iomanip>
```

```
int main() {
    double pi = 3.1415926535;
    int num = 42;

    std::cout << "Default pi: " << pi << std::endl;
    std::cout << "Pi with precision 3: " << std::setprecision(3) << pi << std::endl;
    std::cout << "Pi in fixed format with precision 5: " << std::fixed << std::setprecision(5)
<< pi << std::endl;
    std::cout << "Pi in scientific format: " << std::scientific << pi << std::endl;

    std::cout << std::endl;

    std::cout << "Number in decimal: " << std::dec << num << std::endl;
    std::cout << "Number in hexadecimal: " << std::hex << num << std::endl;
    std::cout << "Number in octal: " << std::oct << num << std::endl;

    std::cout << std::endl;

    std::cout << std::setw(10) << std::left << "Hello" << std::setw(10) << std::right << 123
<< std::endl;
    std::cout << std::setw(15) << std::setfill('*') << std::internal << -456 << std::endl;

    std::cout << std::endl;

    bool value = true;
    std::cout << "Boolean value (default): " << value << std::endl;
    std::cout << "Boolean value (alphabetic): " << std::boolalpha << value << std::endl;
    std::cout << "Boolean value (numeric): " << std::noboolalpha << value << std::endl;

    return 0;
}
```

- **File I/O (fstream, ofstream, ifstream):**
    - **Explanation:** Allows you to read from and write to files. You need to include the <fstream> header.

- ## std::ofstream (for writing to files):

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

int main() {
    std::ofstream outputFile("example.txt"); // Creates or overwrites the file

    if (outputFile.is_open()) {
        outputFile << "This is the first line." << std::endl;
        outputFile << "This is the second line." << std::endl;
        int number = 789;
        outputFile << "The number is: " << number << std::endl;
        outputFile.close();
        std::cout << "Data written to example.txt" << std::endl;
    } else {
        std::cerr << "Unable to open file for writing." << std::endl;
    }

    // Appending to a file:
    std::ofstream appendFile("example.txt", std::ios::app);
    if (appendFile.is_open()) {
        appendFile << "This line is appended." << std::endl;
        appendFile.close();
        std::cout << "Data appended to example.txt" << std::endl;
    } else {
        std::cerr << "Unable to open file for appending." << std::endl;
    }

    return 0;
}
```

- ## std::ifstream (for reading from files):

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

```
int main() {
  std::ifstream inputFile("example.txt");
  std::string line;

  if (inputFile.is_open()) {
    while (std::getline(inputFile, line)) {
      std::cout << "Read from file: " << line << std::endl;
    }
    inputFile.close();
  } else {
    std::cerr << "Unable to open file for reading." << std::endl;
  }

  return 0;
}
```

- ### std::fstream (for both reading and writing):

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

int main() {
  std::fstream file("data.txt", std::ios::in | std::ios::out | std::ios::trunc); // Open for read/write,
truncate if exists

  if (file.is_open()) {
    file << "Initial data." << std::endl; // Write to the file
    file.seekg(0); // Go back to the beginning of the file
    std::string line;
    std::getline(file, line);
    std::cout << "Read from file: " << line << std::endl;
    file.close();
  } else {
    std::cerr << "Unable to open file for read/write." << std::endl;
  }

  return 0;
}
```

  - **File Opening Modes (defined as constants in std::ios):**
    - std::ios::in: Open for input (read operation).
    - std::ios::out: Open for output (write operation). If the file exists, its contents are discarded.
    - std::ios::app: Open for append. Output operations are performed at the end of the file.
    - std::ios::binary: Open in binary mode (data is treated as a sequence of bytes).
    - std::ios::trunc: Truncate the file to zero length if it exists. Used with std::ios::out by default.
    - std::ios::ate: Open and seek to the end of the file.
    - You can combine these modes using the bitwise OR operator (|).
- **String Streams (stringstream, istringstream, ostringstream):**
  - **Explanation:** Allow you to perform I/O operations on strings in memory, as if they were input or output streams. You need to

include the <sstream> header.

- **std::ostringstream (for writing to strings):**

```C++
#include <iostream>
#include <sstream>
#include <string>

int main() {
    std::ostringstream oss;
    std::string name = "John";
    int age = 30;

    oss << "Name: " << name << ", Age: " << age;
    std::string result = oss.str(); // Get the string from the stream
    std::cout << "Formatted string: " << result << std::endl;

    return 0;
}
```

- **std::istringstream (for reading from strings):**

```C++
#include <iostream>
#include <sstream>
#include <string>

int main() {
    std::string data = "123 3.14 Hello";
    std::istringstream iss(data);
    int number;
    double pi;
    std::string message;

    iss >> number >> pi >> message;
    std::cout << "Number: " << number << std::endl;
    std::cout << "Pi: " << pi << std::endl;
    std::cout << "Message: " << message << std::endl;
    return 0;
}
```

- ### std::stringstream (for both reading from and writing to strings):

```
C++
#include <iostream>
#include <sstream>
#include <string>

int main() {
    std::stringstream ss;
    ss << "Value: " << 456;
    std::cout << ss.str() << std::endl;

    int value;
    ss >> value; // This will fail as "Value:" is not an integer
    std::cout << "Extracted value (attempt 1): " << value << std::endl;

    ss.clear(); // Clear error flags
    ss.str("");  // Clear the string content
    ss << 789;
    ss >> value;
    std::cout << "Extracted value (attempt 2): " << value << std::endl;

    return 0;
}
```

This concludes the section on Input/Output in C++. We've covered the basics of streams, standard streams, formatting, file I/O, and string streams. Next, we will move on to Miscellaneous C++ Features. Let me know if you have any questions about these I/O concepts.

Sources
1. https://cppscripts.com/stdexcept-cpp
2. https://cppscripts.com/cpp-fstream-read
3. https://codelucky.com/cpp-file-io/