cin and cout objects

C++ I/O operation is using the stream concept. Stream is the sequence of bytes or flow of data. It makes the performance fast.

If bytes flow from main memory to device like printer, display screen, or a network connection, etc, this is called as **output operation.**

If bytes flow from device like printer, display screen, or a network connection, etc to main memory, this is called as **input operation.**

Standard input stream (cin)

The cin is a predefined object of istream class. It is connected with the standard input device, which is usually a keyboard. The cin is used in conjunction with stream extraction operator (>>) to read the input from a console.

example of standard input stream (cin):

```
#include <iostream>
using namespace std;
int main( ) {
  int age;
   cout << "Enter your age: ";
   cin >> age;
   cout << "Your age is: " << age << endl;
}
```

Output:

```
Enter your age: 22
Your age is: 22
```

Standard output stream (cout)

The cout is a predefined object of ostream class. It is connected with the standard output device, which is usually a display screen. The cout is used in conjunction with stream insertion operator (<<) to display the output on a console

example of standard output stream (cout):

```
#include <iostream>
using namespace std;
int main( ) {
   char ary[] = "Welcome to SNS";
   cout << "Value of ary is: " << ary << endl;
}
```

Output:

```
Value of ary is: Welcome to SNS
```

Unformatted and formatted I/O

Unformatted Input/Output: Unformatted I/O deals with raw data, reading or writing characters, bytes, or blocks of data without any specific formatting. Functions like **read()** and **write()** are typical examples of unformatted I/O operations in C++.

For instance, using **read()** and **write()** for unformatted I/O:

```
#include <iostream>
#include <fstream>
int main() {
   char data[100];
   // Reading from a file using unformatted input
   std::ifstream inFile("input.txt", std::ios::binary);
   inFile.read(data, 100);

   // Writing to a file using unformatted output
   std::ofstream outFile("output.txt", std::ios::binary);
   outFile.write(data, 100);

   inFile.close();
   outFile.close();

   return 0;
}
```

Formatted Input/Output: Formatted I/O is concerned with presenting data in a specific format (like text, numbers, etc.) using functions like **printf()** and **scanf()** (from C) or **cout** and **cin** in C++.

Using **cout** and **cin** for formatted I/O:

```
#include <iostream>
#include <iomanip>
int main() {
   int num = 10;
   double pi = 3.14159;
   // Formatted output using cout
   std::cout << "Number: " << num << std::endl;
   std::cout << std::fixed << std::setprecision(2) << "Pi: " << pi << std::endl;

   // Formatted input using cin
   int userInput;
   std::cout << "Enter a number: ";
   std::cin >> userInput;
   std::cout << "You entered: " << userInput << std::endl;

   return 0;
}
```

Formatted I/O offers more control over how data is presented or parsed, making it easier to display information in a readable and meaningful way, while unformatted I/O deals with raw data directly without any specific structure or formatting.

Manipulators:

Manipulators are helping functions that can modify the input/output stream. It does not mean that we change the value of a variable, it only modifies the I/O stream using insertion (<<) and extraction (>>) operators.

- Manipulators are special functions that can be included in the I/O statement to alter the format parameters of a stream.
- Manipulators are operators that are used to format the data display.
- To access manipulators, the file iomanip.h should be included in the program.

For example, if we want to print the hexadecimal value of 100 then we can print it as:
cout<<setbase(16)<<100

Types of Manipulators There are various types of manipulators:

1. **Manipulators without arguments**: The most important manipulators defined by the **IOStream library** are provided below.
   - **endl**: It is defined in ostream. It is used to enter a new line and after entering a new line it flushes (i.e. it forces all the output written on the screen or in the file) the output stream.
   - **ws**: It is defined in istream and is used to ignore the whitespaces in the string sequence.
   - **ends**: It is also defined in ostream and it inserts a null character into the output stream. It typically works with std::ostrstream, when the associated output buffer needs to be null-terminated to be processed as a C string.
   - **flush**: It is also defined in ostream and it flushes the output stream, i.e. it forces all the output written on the screen or in the file. Without flush, the output would be the same, but may not appear in real-time.
   -

File management functions

In C++, file management is typically handled through the <fstream> library, which provides classes like ofstream (output file stream), ifstream (input file stream), and fstream (file stream for both input and output). These classes allow you to perform various file operations like opening, reading, writing, and closing files.

Creating user-defined file management functions promotes code reusability, readability, and maintenance by encapsulating file operations into modular, reusable components. These functions can be expanded to include error handling, more complex file operations, and additional functionalities

file management functions in C++:

Opening File

```
#include <fstream>
#include <iostream>
int main() {
   std::ofstream outFile; // Creating an output file stream object
   outFile.open("example.txt"); // Opening/creating a file named example.txt
```

```
    if (outFile.is_open()) {
        // File opened successfully
        // Perform operations like writing to the file
        outFile << "Hello, File!";

        outFile.close(); // Close the file when done
    } else {
        std::cout << "Unable to open file!";
    }
    return 0;
}
```

Writing to a File

```
#include <fstream>
#include <iostream>
int main() {
    std::ofstream outFile("example.txt"); // Open file directly in constructor
    if (outFile.is_open()) {
        outFile << "Writing data to a file.\n";
        outFile << 12345 << std::endl; // Writing integers

        outFile.close(); // Close the file
    } else {
        std::cout << "Unable to open file!";
    }
    return 0;
}
```

Reading from file

```
#include <fstream>
#include <iostream>
#include <string>
int main() {
    std::ifstream inFile("example.txt"); // Open file for reading

    if (inFile.is_open()) {
        std::string line;
        while (std::getline(inFile, line)) {
            std::cout << line << std::endl; // Print each line
        }

        inFile.close(); // Close the file
    } else {
        std::cout << "Unable to open file!";
    }

    return 0;
}
```

Checking End-of-File (EOF):

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

int main() {
    std::ifstream inFile("example.txt");

    if (inFile.is_open()) {
        char ch;
        while (!inFile.eof()) {
            inFile.get(ch); // Read character by character
            std::cout << ch;
        }

        inFile.close();
    } else {
        std::cout << "Unable to open file!";
    }

    return 0;
}
```