Lecture 2 Streams

#CS106L

- **Overview**
- **Stringstream**
- **State Bits**
- **Input/Output Streams**
- **Manipulators**

---

device:console,keyboard,file,pipeline,socket

2 part:type conversion,write to console

stringstream position function `` keywords that change the behavior of the stream

standard iostreams:

- `cin`:Standard input stream
- `cout`:Standard output stream (buffered)
- `cerr`:Standard error stream (unbuffered)
- `clog`:Standard error stream (buffered)

Key Takeaways

• `>>` extracts the next variable of a certain type, up to the next whitespace.

- The `>>` and `<<` operators return a reference to the stream

itself, so in each instance the stream is the left-hand operand.

- `stringstream`: Reading and writing simultaneously can often lead to subtle bugs, be careful.

- `eofbit` and `failbit` are normally the ones you will be checking.

ostringstream

```
#include<iostream>

#include<sstream>

using namespace std;

int main(){

    ostringstream oss("hello world",ostringstream::ate);

    //the version of without ostringstream::ate

    cout << oss.str() << endl;

    oss << "fuck the world";

    cout << oss.str() << endl;


    return 0;

}
```

istringstream

```
#include<iostream>

#include<sstream>

using namespace std;

int main(){

    istringstream iss("3.1415926 hello world");
```

```
    int num;

    string unit;

    iss >> num >> unit;

        //guess the result

    cout << num / 2 << " " << unit;

    return 0;

}
```

Q1:Why my code in vscode is invalid? I mean the hello.txt doesn't seem to change?

There are a few potential reasons why your code is not changing the contents of the "hello.txt" file:

1. The input file "hello.txt" may not exist, or it may not be in the correct directory. You should check if the file exists and make sure that the path to the file is correct.
2. You may not have the necessary permissions to write to the file. In this case, you should check the file's permissions and make sure that you have write access to the file.
3. You may be reading the data from the file, but you are not modifying it or writing anything new to it. In the code provided, you are only reading an integer value from the file using the `>>` operator and storing it in the variable `n`, but you are not writing anything new to the file. If you want to modify the contents of the file, you will need to open it with an `ofstream` object and use the `<<` operator or the `write()` method to write new data to the file.
4. If you are trying to write a new value to the file but not just reading from it, you have to open the file in "trunc" mode to clear the existing data in the file, like this:

```
ofstream out("hello.txt", ios::trunc);`
```

5. Another potential reason is that you may not have closed the file after writing to it, this will prevent the changes from being saved, so make sure you close the file properly using the `close()` method of the stream object.

It's recommended to check the existence of the file and the state of the stream to make sure it's open correctly, also check if you have the necessary permissions to access the file and do the necessary operation.

It's also recommended to check the file contents before and after running the code to see what changes have been made.

Those are the main reasons why the code might not be modifying the contents of the "hello.txt" file. However, it's always possible that there may be other issues or bugs in your code that are causing the problem. I would recommend checking for any other errors or warnings that the compiler or the IDE might be displaying and carefully reviewing the code to make sure it's functioning as expected.

Additionally, you can use debugging tools, such as breakpoints, watches, and step-by-step execution to gain more insight into how the code is running and what might be causing the issue.

Another option is to try to isolate the problem by running a simple test that just opens the file and writes a single line of text. This will help you to find out whether the problem is with the opening and writing process itself, or somewhere else in the code.

If you continue to have difficulty resolving the issue, you can provide more details about the problem and the error messages you are receiving and I would be happy to help you troubleshoot further.

Q2:How should my code change? Can you write a few specific changes to the code?

```
#include<iostream>

#include<fstream>

using namespace std;

int main(){
```

```
        ifstream mystream("hello.txt");

        int n;

        mystream >> n;

        return 0;

}
```

Here's an example of how you could modify the code to correctly modify the contents of the "hello.txt" file:

```
#include<iostream>

#include<fstream>

using namespace std;

int main(){

    ofstream outfile("hello.txt",ios::trunc); // open the file in truncate mode

    if(!outfile.good()){

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

        return 1;

    }

    outfile << "5201314" << endl;  // write new data to the file

    outfile.close();  // close the file

    return 0;

}
```

In this example:

- I created an `ofstream` object and passed the file name "hello.txt" to the constructor and also passed the ios::trunc flag, this will open the file in truncate mode which will overwrite any existing data in the file.

- I used the `good()` method of the stream to check if the file is opened correctly, and if not it will print an error message.
- I used the `<<` operator to write new data to the file, in this case, "Hello World".
- Finally, I close the file using the `close()` method of the stream, this is important to make sure that the new data is saved to the file.

Additionally, You can include a check to see if the file already exist, if it doesn't then create it with the initial data, you can use `ifstream` to check the existence, like this:

```
        ifstream infile("hello.txt");

    if(!infile.good()){

    ofstream outfile("hello.txt",ios::trunc);

    outfile << "Hello World" << endl;

    outfile.close();

    cout << "File created" << endl;

    }

    else {

    cout << "File already exist" << endl;

    infile.close();

    // you can also write your data to the file here.

}
```

You could also consider adding error handling and validation checks for the input values before writing them to the file. I hope this gives you a better idea of how you can modify the code to correctly change the contents of the "hello.txt" file. Remember to close the file after writing to it, and check if the file is open correctly, also remember to check if the file exists and if you have the necessary permissions to access it.

In addition, don't forget that you can use the `ios::app` flag to open a file in append mode, which will add new data to an existing file without

overwriting its contents.

You can also use other built-in methods to control the behavior of the stream and the file, such as `open()`, `close()`, `good()`, `fail()` and `bad()` for opening, closing and checking the state of the file.

Remember that when working with files it's important to handle errors and exceptions properly, such as missing files, insufficient permissions, and unexpected situations.

Q3:What exact does `>>` do?

In C++, `>>` is the stream extraction operator. When used in conjunction with input stream objects, such as `cin`, it is used to extract or read data from an input stream, typically from the keyboard or a file, and store it in a variable.

The `>>` operator reads data from an input stream one character at a time until it encounters whitespace (such as a space or newline character). Once it encounters whitespace, it stops reading and assigns the read data to the variable on the left-hand side of the operator.

For example, consider the following code:

```
        int x;

    cin >> x;
```

This code will prompt the user to enter an integer value, read the value from the keyboard, and store it in the variable `x`. It can be used to extract multiple values at once.

```
        int a,b,c;

    cin>>a>>b>>c;
```

This will prompt the user to enter 3 integers and store the value in a,b and c variables respectively.

It can also used with string variables.

```
        string name;

        cin>>name;
```

It will read the first word entered by the user, then assign it to the name variable. It should be noted that `>>` and `<<` operator have the same functionality but they work in opposite direction. `>>` is used for extraction while `<<` is used for insertion.

Yes, `>>` operator also has similar behavior when used in conjunction with other types of stream objects, such as `istringstream` and `ifstream`. For example, when used with an `istringstream` object, it can be used to read data from a string rather than from the keyboard or a file.

```
        istringstream iss("Hello World");

    string s;

    iss >> s;

    cout << s << endl;
```

This code will extract the first word "Hello" from the stringstream object and assign it to the `s` variable and then print it.

It should be noted that the `>>` operator does not automatically check for error conditions such as reaching the end of the stream or reading an invalid data type. If the `>>` operator encounters an error, it will set the stream's "fail" or "bad" bit, which can be checked using the `fail()` or `bad()` methods of the stream object. It's a good practice to check the status of the stream after using the `>>` operator to ensure that the operation was successful before proceeding with further operations on the data.

```
        if (cin >> x)

    cout << "x is: " << x << endl;

    else cout << "input error" << endl;
```

This will check the status of the cin after reading value of x, if it failed to read, it will print "input error" otherwise it will print the value of x.

Q4:what are the concept of buffer and flush in C++?

In C++, a buffer is a temporary storage area where data is held before it is moved to its final destination. When you perform operations such as writing

to a file or sending data over a network, the data is first written to a buffer, and then it is moved to its final destination. The use of a buffer can improve performance by reducing the number of writes to the final destination, for example, when you write to a file instead of writing to the disk every time, the system will write to a buffer in memory, and only when the buffer is full or a flush command is issued, the data will be written to the disk.

A flush command is used to force the data in the buffer to be written to its final destination. For example, when you use the `cout` object to output data to the screen, the data is first written to a buffer, and then it is displayed on the screen. If you want to make sure that the data is displayed on the screen immediately, you can use the `flush` method of the `cout` object, like this:

```
        cout << "Hello World!" << flush;
```

In the case of file streams, `ofstream` and `fstream`, you can use the `flush()` method of the stream object to flush the contents of the buffer to the file. This is useful when you want to make sure that the data is written to the file immediately, even if the buffer is not full.

```
        ofstream outfile("hello.txt",ios::trunc);

        outfile << "Hello World!" << endl;

        outfile.flush(); // flushes the buffer to the file

        outfile.close();
```

In addition to manual flush, the buffer will be automatically flushed in the following cases:

- When the buffer becomes full
- When the program terminates
- When the buffer is associated with an input stream and the user enters an end-of-file condition
- When the buffer is associated with an output stream and the program calls the `endl` manipulator.

It's worth noting that flushing the buffer will cause a write operation that may decrease performance, but it's necessary to ensure that the data is written to the file/destination, and to avoid data loss.

Q5:how to use the ifstream and ofstream in c++ in vscode?

In Visual Studio Code, you can use the `ifstream` and `ofstream` classes to read from and write to files just like you would in a regular C++ program. Here is an example that shows how to use `ifstream` to read a text file and `ofstream` to write a text file:

```
#include <iostream>

#include <fstream>

using namespace std;



int main() {

    // Reading from a file

    ifstream input_file("input.txt");

    if(!input_file.good()) {

        cout << "Error opening the input file." << endl;

        return 1;

    }


    string buffer;

    while(getline(input_file, buffer)) {

        cout << buffer << endl;

    }

    input_file.close();
```

```
    // Writing to a file

    ofstream output_file("output.txt");

    if(!output_file.good()) {

        cout << "Error opening the output file." << endl;

        return 1;

    }


    output_file << "Hello World!" << endl;

    output_file.close();


    return 0;

}
```

In this example, it first creates an `ifstream` object and opens the file "input.txt", it then reads the data from the file line by line and prints the read text to the console. Then it creates an `ofstream` object and opens the file "output.txt", it then writes the text "Hello World!" to the file and finally closes the file.

In VSCode, you can simply open a file by right-clicking on the file in the explorer and then select "Open with Code", or you can open the file by running the `code` command in the terminal and then passing the file path as an argument.

You can also run the c++ program in VSCode by using the integrated terminal and running the compiler command. The exact command to run will depend on the compiler you are using. For example, if you are using GCC, you can open the integrated terminal by pressing `ctrl`+`backtick` or from the menu `View` -> `Terminal`, then you can run the following command:

```
g++ main.cpp -o main
```

This will compile the file `main.cpp` into an executable file named `main`. You can then run the program by executing:

```
./main
```

Alternatively, you can use a VSCode extension such as "Code Runner" to build and run the C++ program directly from within the editor, you just need to press `Ctrl+Alt+N` to run the code, or you can use the 'Run Code' button on the top right corner of the editor.

Make sure that the file(s) that you read or write are in the same directory of the c++ file or you have to specify the path to it otherwise it will cause an error.

It's also good to keep in mind that if you are writing to a file and the file already exists, the contents of the file will be overwritten. If you want to add new data to an existing file without overwriting its contents, you can open the file in "appending" mode by passing an additional argument to the `ofstream` constructor, like this:

```
ofstream output_file("output.txt", ios::app);
```

In short, You can use the `ifstream` and `ofstream` classes to read from and write to files in VSCode just like in any other C++ program, and you can use various built-in.

Q6:Is there a stringstream that can you can both insert and extract?

Yes, in C++ there is a class called `stringstream` which can be used for both inserting and extracting data from a string. The `stringstream` class is part of the C++ Standard Template Library (STL), and it is defined in the `<sstream>` header file.

The `stringstream` class is derived from both the `istream` and `ostream` classes and it can read from and write to a string using the same syntax as reading from and writing to a file. The `stringstream` class has an internal string buffer that it reads from and writes to, so you can use the `<<` operator to insert data into the buffer and the `>>` operator to extract data from the buffer.

For example, to insert data into a `stringstream` object you could do:

```
stringstream ss;
```

```
        ss << "Hello" << " World" << endl;
```

And to extract the data from the `stringstream` object you could do:

```
        string s;

        ss >> s;

        cout << s << endl;
```

In the above example, it first insert the string "Hello World" into the buffer and then it extracts the first word "Hello" and assigns it to the `s` variable and then prints it.

It also has a `str()` method which allows you to access the underlying string buffer, this can be useful for further manipulation of the data in the string.

`stringstream` class also provides convenient ways to convert between the different fundamental types, such as int, double, and string. It uses the `<<` and `>>` operator to insert and extract values in the string buffer and can parse the data from string. This makes it a useful class for parsing data from a file, network connection, or user input, and for serializing data to be sent over a network or stored in a file.

there are a few other things to note about the `stringstream` class:

- The `stringstream` class provides the same methods as the `istream` and `ostream` classes, such as `good()`, `eof()`, and `fail()`, which can be used to check the status of the stream and detect errors.
- The `stringstream` class also provides member functions for changing the formatting flags, such as `width()` or `precision()`, this can be used to format the data in the stream.
- One of the power of `stringstream` is that you can use the `str()` method to access the underlying string buffer and manipulate it directly, this can be useful if you need to access a substring or extract a specific part of the string.
- A common use of `stringstream` is parsing CSV(Comma Separated Values) files or other structured text data, where you can read in a line of text, split it into fields using `getline()`, and then extract the values using the `>>` operator.

- `stringstream` is thread-safe in C++11 and later, meaning that it can be used safely in multi-threaded environments without additional synchronization.

`stringstream` is a powerful and versatile class that provides an easy way to work with strings as input and output streams, as well as perform type conversions on data. It's widely used in many types of applications such as parsing, formatting and data conversion . And it's an important part of STL, providing a convenient way to work with strings and other data types.

Q7:how the stream read from the file into the buffer in c++?

In C++, you can use the `ifstream` class to read data from a file into a buffer. The `ifstream` class is part of the C++ Standard Template Library (STL) and it is defined in the `<fstream>` header file.

The basic process for reading data from a file using an `ifstream` object involves the following steps:

1. Create an `ifstream` object and associate it with the file you want to read from by calling the `open()` method, passing the file's name as an argument.

```
ifstream input_file;

input_file.open("data.txt");
```

2. Check if the file was opened successfully. If the `open()` method failed, the `fail()` or `bad()` method of the `ifstream` object will return `true`, indicating that there was an error opening the file.

```
if(!input_file.good()) cout << "Error opening the file." << endl;
```

3. Once the file is opened, you can read the data into a buffer by using the `>>` operator. You can read one value at a time by using the `>>` operator, or you can use the `getline()` method to read an entire line of text into a string.

```
string buffer;

while(getline(input_file, buffer)) {
```

```
    cout << buffer << endl;

}
```

4. Once you are done reading the data, you should close the file by calling the `close()` method of the `ifstream` object.

```
        input_file.close();
```

It's also worth noting that ifstream provides various other member functions like `read()` which can be used to read binary data from the file, `seekg()` method that allows to move the read pointer to a specific position and `tellg()` method returns the current position of the read pointer.

This is a basic example of how to read data from a file using an `ifstream` object and a buffer in C++. There are many other ways to read data from a file depending on the context and the type of data you are working with.

It's also important to keep in mind that when reading large files, it may be more efficient to read the data in smaller chunks and process them incrementally, instead of reading the entire file into memory at once.

It's also worth to note that if you are reading a large file in which you only need a specific portion, it's better to use `mmap()` instead, which maps the file directly into memory and allows random access to it, this will improve the performance, and lower the memory usage.

Always consider the use case and the type of data, and apply the best method for that situation. Also, it's important to handle the cases of errors and unexpected situations in the reading process like missing files, insufficient permissions, and etc.