Advanced C

Input/Output Library

Th.S Nguyen Minh Anh

Phenikaa University

Last Update: 27th February 2023

Recap

We have learned how to handle console input and output:

- The `std::cout` to print out console
- The `std::cin` to input from keyboard

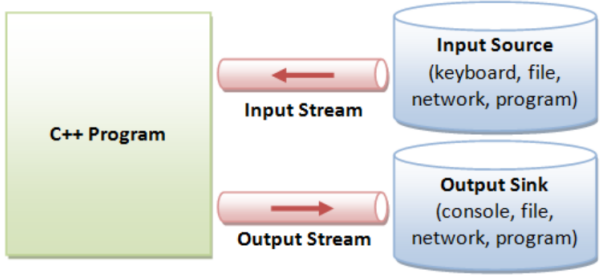
What if our data from file instead of keyboard? or we want to store output into a file instead of just printing it out?

Outline

Definition

stream

- Sequence of bytes **flowing in and out** of the programs
- Think of streams as a **pipeline of data** (just like water and oil flowing through a pipe).
- Streams acts as an interface between the programs and the actual IO devices
- Streams convert between *data* and the *string representation of data*.



Two ways to classify streams

By Direction:

- **Input streams**: Used for reading data (ex. `std::istream`, `std::cin`)
- **Output streams**: Used for writing data (ex. `std::ostream`, `std::cout`)
- **Input/Output streams**: Used for both reading and writing data (ex. `std::iostream`, `std::stringstream`)

By Source or Destination:

- **Console streams**: Read/write to console (ex. std::cout, std::cin)
- **File streams**: Read/write to files (ex. std::fstream, std::ifstream, std::ofstream)
- **String streams**: Read/write to strings (ex. std::stringstream, std::istringstream, std::ostringstream)

Output Streams

An output stream lets you take data from your program and output it to a source (like the console, a file, etc.).

- Can only **send** data to the stream.
- Send data using stream **insertion operator**: $<<$
- Converts data into string and sends to the stream.

The `std::cout` stream is an example of an output stream.

```
std::cout << 5 << std::endl;
// converts int value 5 to string '5'
// sends '5' to the console output stream
```

Input Streams

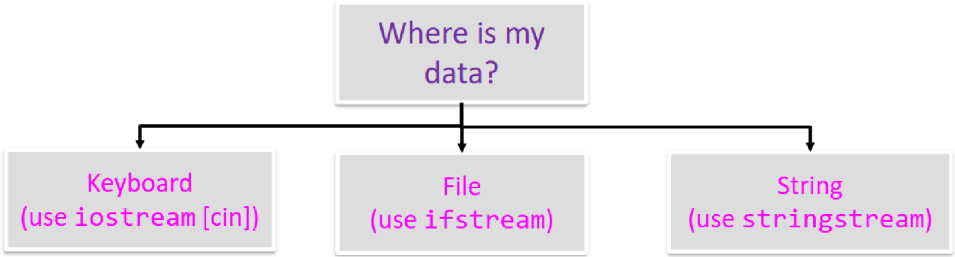
An input stream lets you get data from a source (like user input, a file, a webpage, etc.) and read it in your program.

- Can only **receive** data.
- Pull out data using stream **extraction operator**: >>
- Extraction receives data from stream as a string and converts it into the appropriate type.

The `std::cin` stream is an example of an input stream.

```
int x;
string str;
std::cin >> x >> str;
//reads exactly one int then one string from console
```

I/O Decision Tree



Outline

Outline

Reading in a File

Let say we want to read the `data.txt` file

```
1 4 67 2 9 1 6 2 7
11 12 56 10 9 50 26
11 99 122 45 23 2 4
2 88 17 21 56 77 17 0
```

Step 1: Open the file

Remember to include the fstream header to use the fstream type

```
#include<fstream>
```

Before we can use an input stream in a program, we must **create** stream object:

```
ifstream inFile;
```

To **connect** the ifstream `inFile` to the file "data.txt", we use the following statement:

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

Checking for Failure with File Commands

The either of following commands is used to check if the stream is valid

```
inFile.is_open()
inFile.fail()
```

```
int main()
{
   ifstream inFile;

   inFile.open("Lecture4");
   if (inFile.fail())
   {
      cout << "Unable to open file!\n";
      exit(0);
   }
   ...
```

Step 2: Read the file - One word at a time

Read one word at a time using extraction operator >>

```
int num;
vector<int> vec;
int total = 0;
while(inFile >> num) {
   // do something with num
   vec.push_back(num);
   total += num;
}
```

Step 3: Close the file

To **disconnect** connect the ifstream "`inFile`" to whatever file it is connected to, we write:

```
inFile.close();
inFile.open("other_data.txt");  // open another file
```

Outline

Step 1: Open the file

Before we can use an output stream in a program, we must **create** stream object:

```
ofstream outFile;
```

To **connect** the ofstream `outFile` to the file "result.txt", we use the following statement:

```
outFile.open("result.txt");
```

Step 2+3: Write the file, then close

```
int main(){
   ...
   outFile << "List of numbers: ";
   for (int num : vec)
      outFile << num << " ";
   outFile << "\n";
   outFile << "Sum = " << total << endl;
   outFile.close();
```

That Looks Familiar...

- If file-writing syntax seems similar to printing to the console, that's because it is!
  - `cin` is an `ifstream`; `cout` is an `ofstream`

Here's the problem

What if the input file looks like this ...

```
1,4,67,2,9,1,6,2,7
11,12,56,10,9,50,26
11,99,122,45,23,2,4
2,88,17,21,56,77,17,0
```

Here's the problem

What if the input file looks like this ...

```
1,4,67,2,9,1,6,2,7
11,12,56,10,9,50,26
11,99,122,45,23,2,4
2,88,17,21,56,77,17,0
```

or this ...

```
Folarin Balogun,Reims,15
Alexandre Lacazette,Lyon,14
Kylian Mbappe,PSG,13
Neymar,PSG,12
```

getline() and stringstreams

A file has a **certain format** where you know related data is on a single line of text but aren't sure how many data items will be on that line

```
1,4,67,2,9,1,6,2,7
11,12,56,10,9,50,26
11,99,122,45,23,2,4
2,88,17,21,56,77,17,0
```

```
Folarin Balogun,Reims,15
Alexandre Lacazette,Lyon,14
Kylian Mbappe,PSG,13
Neymar,PSG,12
```

- Can't use >>
- We can use `getline()` to get the whole line as a string, then a `stringstream` with >> to parse out the pieces (**Tokenizing a string**)

Outline

stringstream

Sometimes we want to be able to treat a string like a stream.
Useful scenarios:

- Converting between data types
- Tokenizing a string

Include header file `sstream` to use stringstream type:

```
#include <sstream>
```

istringstream

An **istringstream** lets you tokenize a string.
Let's extract data from the line **"Balogun Reims 15"**

```
string line = "Balogun Reims 15";
...
...
...
...
...
...
```

istringstream

An **istringstream** lets you tokenize a string.

Let's extract data from the line **"Balogun Reims 15"**

```
string line = "Balogun Reims 15";
istringstream iss(line);
string name;
string club;
int numGoal;
iss >> name >> club >> numGoal;
cout << name << "-" << club << "-" << numGoal << endl;
```

istringstream - using Delimeters

How about we change the line a bit:

"Balogun Reims 15" $\rightarrow$ **"Balogun, Reims, 15"**

```
string line = "Balogun, Reims, 15";
...
...
...
...
...
...
...
...
```

std::getline()

Used to read a line from an input stream

```
istream& getline(istream& is, string& str, char delim);
```

- **is**: the stream getline reads from
- **str**: where it stores output in
- **delim**: Stops when read ('newline' = default)

How it works:

- Clears contents in `str`
- Extracts chars from `is` and stores them in `str` until:
  - End of file reached
  - Next char in `is` is `delim`, extracts but does not store `delim`

istringstream - using Delimeters

How about we change the line a bit:

"Balogun Reims 15" $\rightarrow$ **"Balogun, Reims, 15"**

```
string line = "Balogun, Reims, 15";
istringstream iss(line);
string token;
getline(iss, token, ',');
cout << token << endl; // token = Balogun
getline(iss, token, ',');
cout << token << endl; // token = Reims
getline(iss, token, ',');
cout << token << endl; // token = 15
```

ostringstream

An `ostringstream` lets you write output into a string buffer.
- Use the `str` method to extract the string that was built.

```
// produce a formatted string of output
int age = 42, iq = 95;
ostringstream oss;
oss << "Zoidberg's age is " << age << endl;
oss << " and his IQ is " << iq << "!" << endl;
string result = oss.str();
// result = "Zoidberg's age is 42\nand his IQ is 95!\n"
```

Choosing an I/O Strategy

- **Is my data delimited by particular characters?**
  - Yes, stop on newlines: Use getline()
  - Yes, stop on other character: User getline() with optional 3rd character
  - No, Use >> to skip all whitespaces and convert to a different data type (int, double, etc.)
- **If "yes" above, do I need to break data into smaller pieces (vs. just wanting one large string)**
  - Yes, create a stringstream and extract using >>
  - No, just keep the string returned by getline()
- **Is the number of items you need to read known as a constant or a variable read in earlier?**
  - Yes, Use a loop and extract (>>) values placing them in array or vector
  - No, Loop while extraction doesn't fail placing them in vector

Remember: getline() always gives text/string. To convert to other types it is easiest to use >>