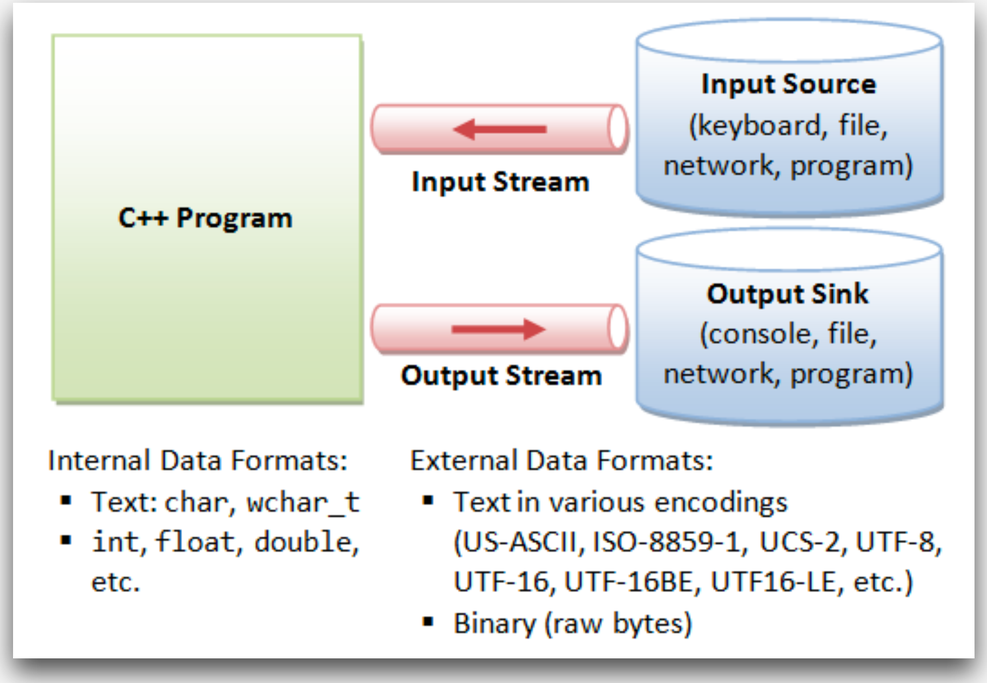
Chapter 7

File I/O and Streams

Streams

C++ input and output are managed by ***streams*** - a sequence of bytes or characters flowing in and out of a program.

Using streams allows for a consistent interface with processing different mediums.

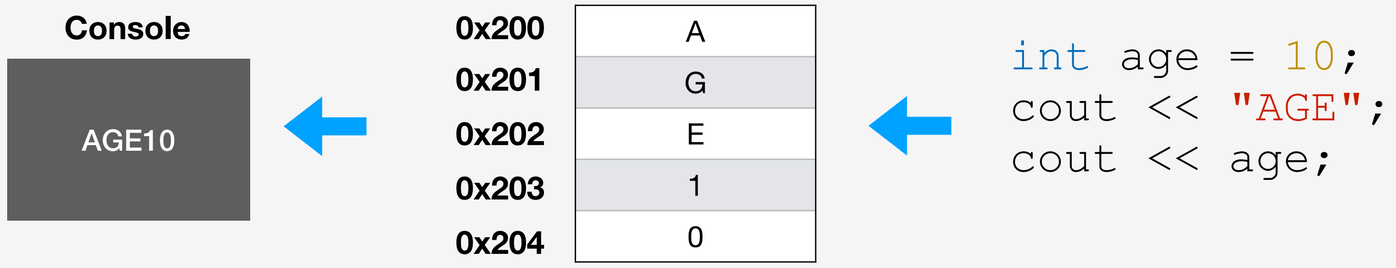


3 Different Mediums

- Console Input / Output
  - ***cin, cout***
- Stringstreams Input / Output
  - ***istringstream, ostringstream***
- File Input / Output
  - ***ifstream, ofstream***

cout

- ***cout*** is an ***ostream*** object that supports output to the console
- Uses **<< -** the insertion operator
- The insertions operator converts different types of data into a sequence of characters.



Note

Use ***endl*** or ***flush*** on debugging print statements so that you can ensure they are being printed to the console before an error occurs.

cin

- ***cin*** is an ***istream*** object that supports input from the console
- Uses **>>** **-** the extraction operator
- The extraction operator to extract data from a buffer and feed it into different types of variables in the program.

1. **Skips whitespace**
2. **Extracts characters**
3. **Stops at next whitespace**
4. **Converts extracted characters to targets variable type**
5. **Stores result into variable.**

NOTE: Remember the nuance of using ***getline***() and ***cin***

Handling ***cin*** errors

Good programs do not assume that users will always enter the correct data.

- For Example: Ask a user for their age (as an ***int***) and the user enters “incorrect”.

We can test for the error state of ***cin*** and clear the erroneous data from the streaming buffer to re-prompt!

```
// clears error state
cin.clear();
// Ignore characters in stream until newline
cin.ignore(numeric_limits<streamsize>::max(), '\n');
```

example: cin_error.cpp

Output formatting

Manipulators are functions and operators that can be used alongside the insertion operator to adjust / control the way data appears on the output medium

- Manipulators affect any medium
- Available from a variety of libraries, namely: `<iomanip> <ios>`
- Affects all subsequent output because the manipulator is set on the buffer

```
double myFloat = 12.223;
cout << setprecision(3) << myFloat << endl;
```

12.2

example: manipulators.cpp

Output formatting

Table 7.3.1: Floating point manipulators.

Manipulator	Description	Example
fixed	Use fixed-point notation. From <ios>	12.34
scientific	Use scientific notation. From <ios>	1.234e+01
setprecision(p)	If stream has not been manipulated to fixed or scientific: Sets max number of digits in number	p=3 yields 12.3 p=5 yields 12.34
	If stream has been manipulated to fixed or scientific: Sets max number of digits in fraction only (after the decimal point). From <iomanip>	fixed: p=1 yields 12.3 scientific: p=1 yields 1.2e+01
showpoint	Even if fraction is 0, show decimal point and trailing 0s. Opposite is noshowpoint. From <ios>	For 99.0 with precision=2 and fixed: 99 (default or noshowpoint) 99.00 (showpoint)

Output formatting

Table 7.3.2: Certain text manipulators help align output.

Manipulator	Description	Example (for item "Amy")
setw(n)	Sets the number of characters for the next output item only (does not persist, in contrast to other manipulators). By default, the item will be right-aligned, and filled with spaces. From <iomanip>	For n=7: " Amy"
setfill(c)	Sets the fill to character c. From <iomanip>	For c='*': "****Amy"
left	Changes to left alignment. From <ios>	"Amy "
right	Changes back to right alignment. From <ios>	" Amy"

example: more_manipulators.cpp

Additional Manipulators

- ***endl -*** insert a newline character, then flush the buffer
- ***flush -*** flushes buffer
- ***ends*** - ends the use of manipulators

```
setprecision(3); // does not affect output
cout << setprecision(3);
```

Remember

Manipulators must be called on a streaming object. Once they do, they affect all subsequent output.

C++ stringstreams

Streaming data from a string

istringstreams

stringstreams are a way for a programmer to treat strings as a streaming medium (works as both input, and output).

- ***istringstreams -*** input string stream. Used with the extraction operator and ***getline();***
- ***ostringstream -*** output string stream. Used with the insertion operator.
- Both are provided by the ***#include <sstream>*** library

Pro Tip:

String Streams are extremely helpful with parsing data!

Using stringstreams

- Stringstreams as input:

```
istringstream inSS(“”);
```

- Stringstreams as output:

```
ostringstream oSS;
oss << "Hello"
    << " World";
```

- Can also use a ***stringstream*** object to be used with both input and output.

```
stringstream stream;
```

example: word_count.cpp

when using stringstreams…

- Make sure you clear your stringstream before using it again!

```
stringstream stream1;
// Do something with stream1
stream1.clear();
stream1.str(“”);
```

- Or just use a new stringstream object

```
stringstream stream1;
// Do something with stream1
stringstream stream2;
// Do something with a new stream
```

Why use stringstreams?

- Can be used to format strings for use later (remember manipulators)
- Really useful for parsing data

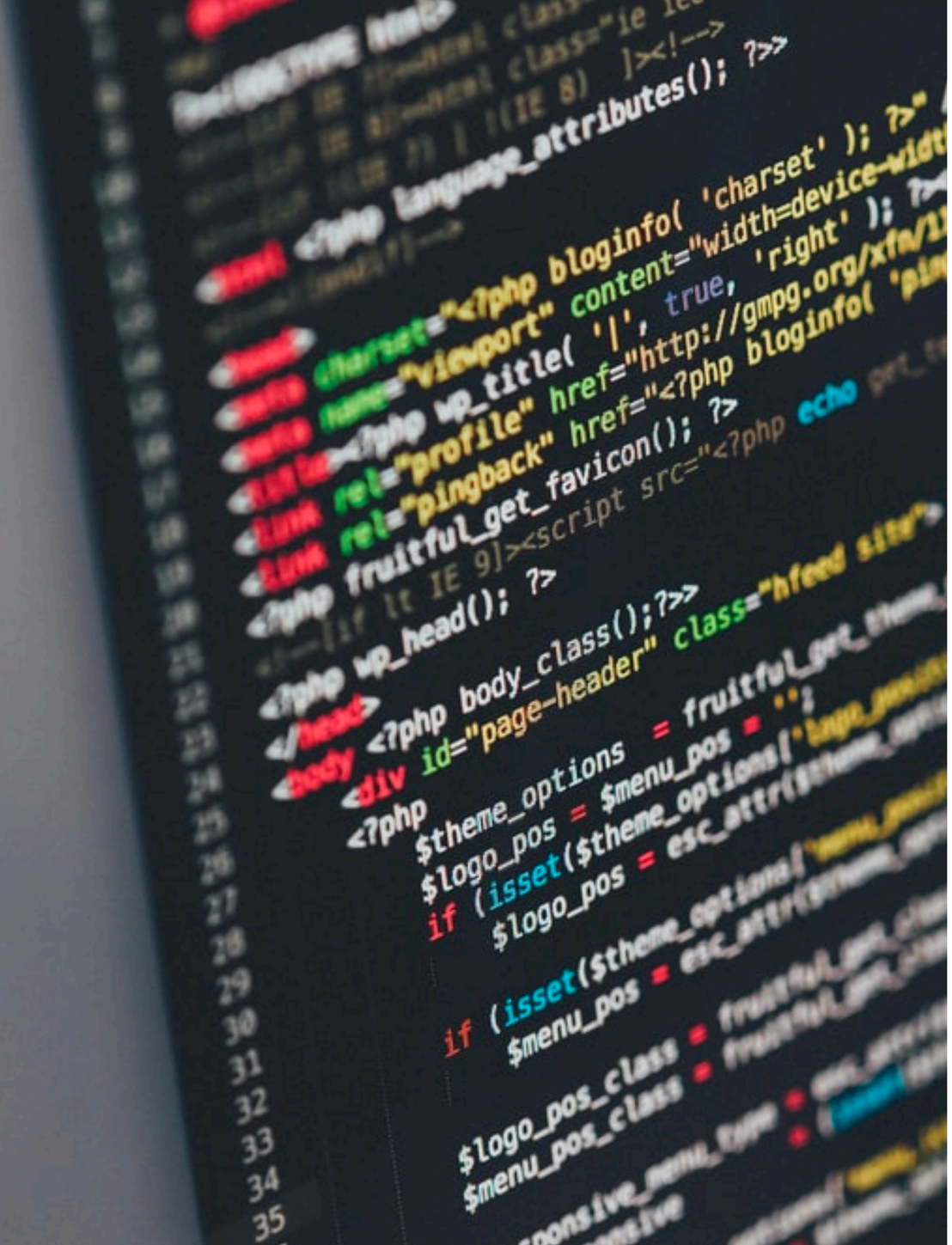
```
istringstream inSS("Erik Gabrielsen");
inSS >> firstName;
inSS >> lastName;
```

- Can be used for data type conversion!

```
istringstream inSS("12345");

int number;
inSS >> number;  // converted to int!
```

example: conversion.cpp



Parse a csv string!

Write a function that takes a string of comma separated values, and places the values in an array.

examples: csv_line.cpp

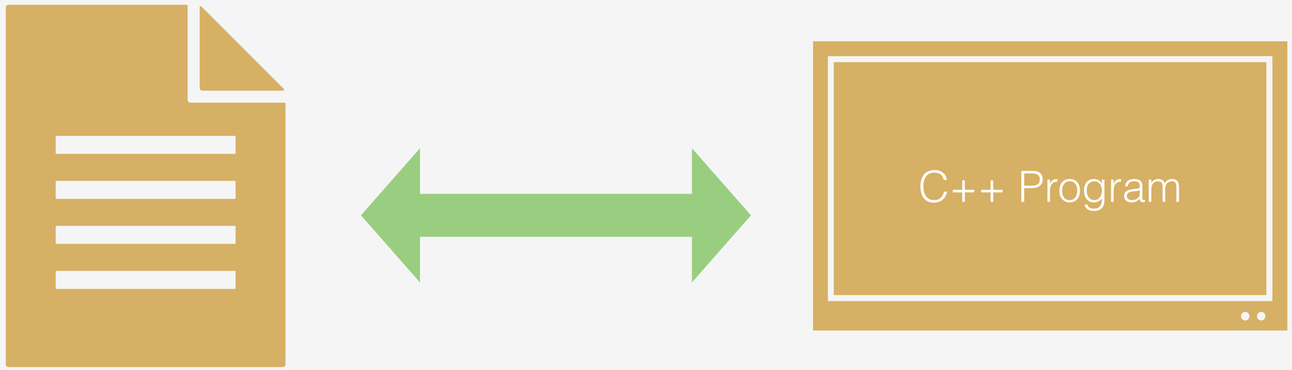
Files

Using C++ fstreams

Why use File I/O?

File Input and Output is an essential part of programming.

- Most software involves persisting or storing data when a program is not in use!
- In C++ we can accomplish this by using files (writing data to disk).
- This is how we can persist data even after a program is terminated



File I/O Basics

In order to communicate with a file, we need to establish a ***file stream object.***

- `#include <fstream>`
- `ofstream file1;`
  - Creates a file stream object that will be used for output
- `ifstream file2;`
  - Creates a file stream object that will be used for input
- Before we can read or write from a file, we must first ***open*** the file
  - `file1.open("filename.txt");`

examples: file_input.cpp
file_output.cpp

File I/O - Handling File Objects

- Opening a file does not always work. What if the file does not exist or is corrupt?
  - Handle file errors appropriately

```
ifstream file1;
file1.open("invalidFile.txt");
if (!file1) cout << "Could not open file!" << endl;
```

After you are done with any File streaming object, we must close the file so that we unload it from our program

- `file1.close();`

example: file_check.cpp

Relative Path vs Absolute Path

Relative Path - The path to a specific file, starting from whatever current directory you are in.

- In Clion, We open a file using:

```
ifstream file1;
file1.open(“../input.txt"); // Note the ..
```

Use Relative Path on Program 2!!

Absolute Path - The full path to a file or directory, starting from the root directory of the machine. For Mac the root directory is “/“. For Windows the root directory starts at “C://“

```
ifstream file1;
file1.open("/Users/egabrielsen/Desktop/cs1342/input.txt");
```

Best Practices!

When parsing information from files it is often preferred to use fstream AND sstring!

- ***First,*** read in each line from a file using an fstream object and ***getline().***

```
string line;
getline(file1, line);
```

- ***Second,*** using the string that holds each line of text, convert it into a stringstream

```
stringstream lineSS(line);
```

- ***Third,*** parse each individual attribute from the string stream object, into a string or whatever datatype is needed

```
string firstName, lastName, age;
getline(lineSS, firstName, ',');
getline(lineSS, lastName, ',');
getline(lineSS, age);
```

example: users.cpp