# CS 102
# Introduction to Programming Using C++

## Chapter 8

Files and Streams

# Homework

- Written homework
- R8.3, 6, 7, 8, 9, 11 to 13

- Programs
- p. 379. Choose one of P8.1, 2, and 4.
- Also, choose one of P8.14, 15, and 16.

# Writing to a File

- This is very similar to reading
- You declare a variable of type ofstream

```
ofstream out_file;
```

- Then you open the file

```
out_file.open ("\\CS102\\programs\\datafile");
```

- When you want to write to the file, you use

```
out_file << data_to_write << endl;
```

  - This is just like cout
- You should close the file at the end

# A Sample File

- Let's write some numeric data to a file
- We have the same problem we had back in Chapter 6 for arrays
  - We need a separator
- The difference is now it's not just for a person to read, it's for the computer
- It's easy to write the data, but…

# Reading That File

- What if you wanted to read in the file?
- Because of that, you can't just write the data to the file
- You might want to line up the data
- To do that you use I/O manipulators
- We saw these before, in Chapter 2
- They control how the data looks when it's printed (even if it's "printed" to a file)

# I/O Manipulators: Things to Remember

- The manipulators are used in a cout statement
  - These ideas can now be applied to file output
- Everything very similar to what we did before
- To use these manipulators, you need

  #inlcude <iomanip>
- Some I/O manipulators are listed on the next slide
- See iomanip.cpp for examples

# I/O Manipulators

- setw (5)
  - Sets the field width
  - Only refers to the next item in the cout statement
- setfill ('0')
  - Prepends 0s to integers that are smaller than the field width
- left, right
  - Controls data alignment in the field
  - The default is right
- fixed
  - Sets non-scientific notation mode
- setprecision (2)
  - Sets the number of digits after the decimal point
  - Should be used in conjunction with fixed

# Using left and right

- General use of left and right
  - We usually use left for strings
  - We usually use right for numbers
    - Of course, if you are expecting numbers with decimal points, you need to line up the decimal points
- Why?
  - That's how we look at those kinds of data

# String Streams

- This is a bit different
- You can create a string that is a stream
  - Why would you want to do that?
- It's for parsing a string
  - Parsing something means to break it into pieces according to some rule
- Of course, we do that to analyze the text

# An Example from the Textbook

```
istringstream strm;
strm.str("January 24, 2019");
```

- This sets strm to "January 24, 2019"
- Then you can parse it, based on the spaces:

```
string month;
int day;
string comma;
int year;
strm >> month  >>  day  >>  comma
        >>  year;
```

- Note that, to use a stringstream, you need

```
#include <sstream>
```

# A Real Timesaver

- Imagine doing this yourself
  - Take a few minutes to try this
  - Write some pseudocode or actual code
- You have to do two things
  - You have to split the string
  - You have to convert the numbers manually
- This will take much more coding time than just using a stringstream

# Creating a String from Data

- This is the reverse of the previous idea
- Suppose you have

  string month = "January";

  int day = 24;

  int year = 2019;
- You want to put them into a single string variable in the usual format
- It's similar, but you use an ostringstream

# The Code

```
ostringstream strm;
string month = "January";
int day = 24;
int year = 1973;
strm << month << " " << day << ", " << year;
```

- You can then print this with
  `cout << strm.str() << endl;`

# Running Programs from the Command Prompt

- An alternative to an IDE is the command prompt
- There are two advantages to using the command prompt
- The first is the idea of command line arguments
  - These are also called switches
- Windows switches are preceded by /
- Linux switches are preceded by –
- These arguments alter the operation of a command

# Some Arguments

- The dir command will list all files in the current folder
  - Linux uses ls
  - Its commands are somewhat different
- Some versions of the dir command are

  dir /w

  dir /od (Also dir /o-d)

  dir /ah

  dir /b

  dir /p

# Using Arguments with Our Own Programs

- To get the arguments, you have to call main () differently

  int main (int argc, char* argv [ ])

- argc tells how many arguments are used
- argv holds the arguments
  - This is similar to any array
  - argv [0] is the program name
  - The actual arguments are argv [1] , …, argv [argc-1]

# A Simple Program with Command Line Arguments

```
#include <iostream>
#include <iomanip>
using namespace std;

int main (int argc, char*argv[])
{
   for (int i=0;  i<argc;  i++)
        cout <<  "Argument "  <<  i  <<  ": "  <<  argv [i]
                 <<  endl;
}
```

# Command Line Arguments in Practice

- Command line arguments are used primarily for two things
- First, they are used to specify options
  - A program can have several options
- Second, they are used for files
  - If a program uses files, it's very common for the file name to be an argument on the command line

# The Textbook's Example

- The textbook has an example of an encryption/decryption program
- The program assumes you want to encrypt a file
  - This is the default
- If you want to decrypt a file, you use the –d argument
- Here are two examples

  caesar input.txt encrypt.txt

  caesar –d encrypt.txt output.txt

# A Caesar Cipher

- There are many was to encrypt data
- One way is the Caesar cipher
- It replaces every letter with the letter k letters after it
  - For example, suppose k = 5
  - Then it changes a to f, b to g, c to h, etc.
- When you reach the end of the alphabet, you start over
  - So, in the example the cipher changes u to z, v to a, w to b, etc.
- You can think of the letters as being on two cylinders, side by side, and you rotate the cylinders
- This is a very simple code
- It is very easy to decipher
- It is not really used

# The Textbook's Version

- In the textbook's version, they don't treat the alphabet as being on a cylinder
- They just subtract 3 from each letter's ASCII code
- In ASCII, 'A' is 65, 'B' = 66, etc.
- Also, 'a' = 97, 'b' = 98, etc.
- For example,
  - 't' becomes 'q', etc.
  - The end-of-line character(s) become unprintable
- You can change the value of k in the program

# Other Notes about the Program

- You should always pay attention to, and add these ideas to your code
- Important checks
  - After opening the input file, they check for errors
  - After opening the output file, they again check for errors
  - They also check that the number of arguments is correct
- Notice that the encryption/decryption is done by a function
  - This makes it easy
  - Well-designed programs are always modular

# The put Command

- You can use the put command to write a single character
  - For example

    ```
    char ch = 'x';
    cout.put (ch);
    ```
- This is more efficient than just cout << ch
- It is a holdover from C itself
- It is often used with files and screen output

# The get command

- There is a counterpart for reading input
- It is the get command
- Its syntax is get (ch)
- The character read is stored in ch
- get is frequently and commonly used for files
- The program caesar.cpp is on p.366

# Another Advantage of the Command Prompt

- The command prompt can also be used to redirect input and output
- You can send your output to a file instead of the screen
- You can get input from a file instead of the keyboard
- There is a third option, allowing the output of one program to be the input of the next

# Redirecting Output

- To redirect output, use >
- For example
  - the dir command lists the files in the current directory
- You can redirect the output to a file
- To send the output to a file instead of the screen, use

  dir >dirlist
- You can now use the file dirlist like any other file

# Redirecting Input

- To redirect input, use <
- For example
  - the more command echoes a program
  - You can get input from a file using more
  - This is useful for viewing a file a screenful at a time
- The command

  more <dirlist
- displays the dirlist file one screenful at a time

# The Pipe

- The last way to redirect input and output is to use the output of one program as input to another
- This is accomplished using the vertical bar, the "pipe": |
- If you have a folder with many files, you can use the dir command to list them
- All that output goes by too fast
  - You can "pipe" this to the more command to see the list a screenful at a time
  - To do this, use

    dir | more

# Connecting this to C++

- By using cout and cin, you enable this
- For example, here is a program

```
#include <iostream>
#include <fstream>
using namespace std;
int main ()
{
    string input_line;
    while (getline (cin, input_line))
        cout << input_line  <<  endl;
    return 0;
}
```

# Using the Program

- Here are several examples
- To print a file on the screen

  a <filename
- To type something and store it into a file

  a >filename
- To copy one file to another

  a <firstfile >secondfile
- You can come up with other examples

# Questions

- Are there any questions?