Topic 1 – CS1A Review – P5 – Small Topics

Header Files
Files
Stream Variables
Random Number Generators

User Defined Header Files

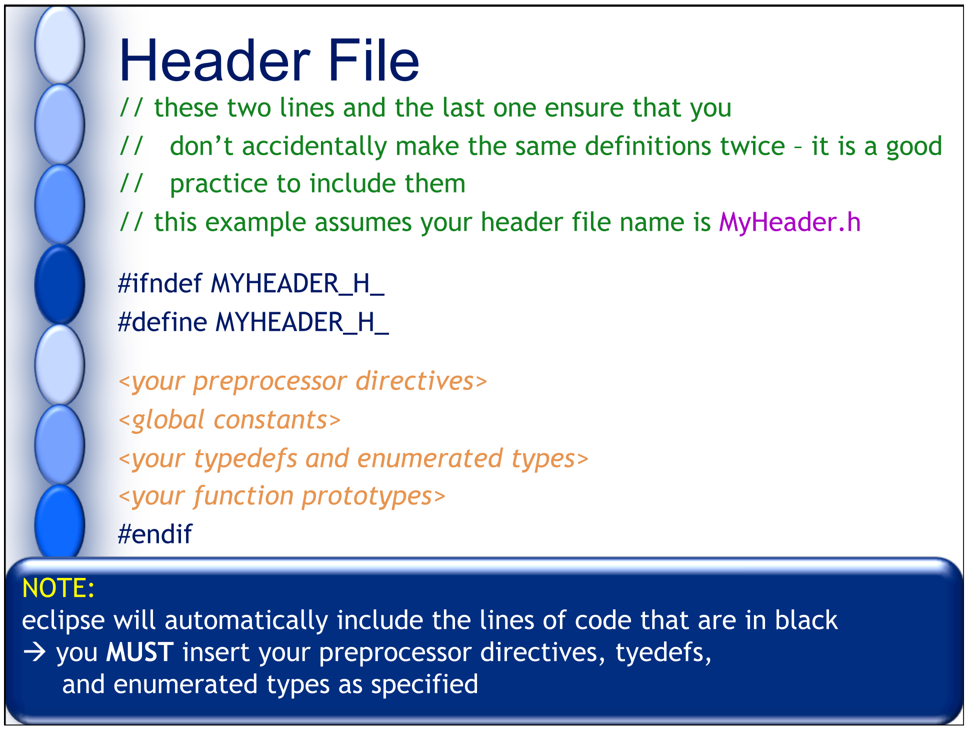
Header files

- So far we've worked with several header files
  - files that follow #include
  - <iostream>
  - <iomanip>
  - <fstream>
  - <string>

- We include these to be able to access certain predefined functions, classes, or variables in C++

Creating our own

- It is often convenient to create your own header files
- To do this we need to
  - create the file
  - Include it in our source code
- Creating the file
  - create a new file *filename.h*
    - end it with .h
- Including the file
  - #include "*filename*.h"



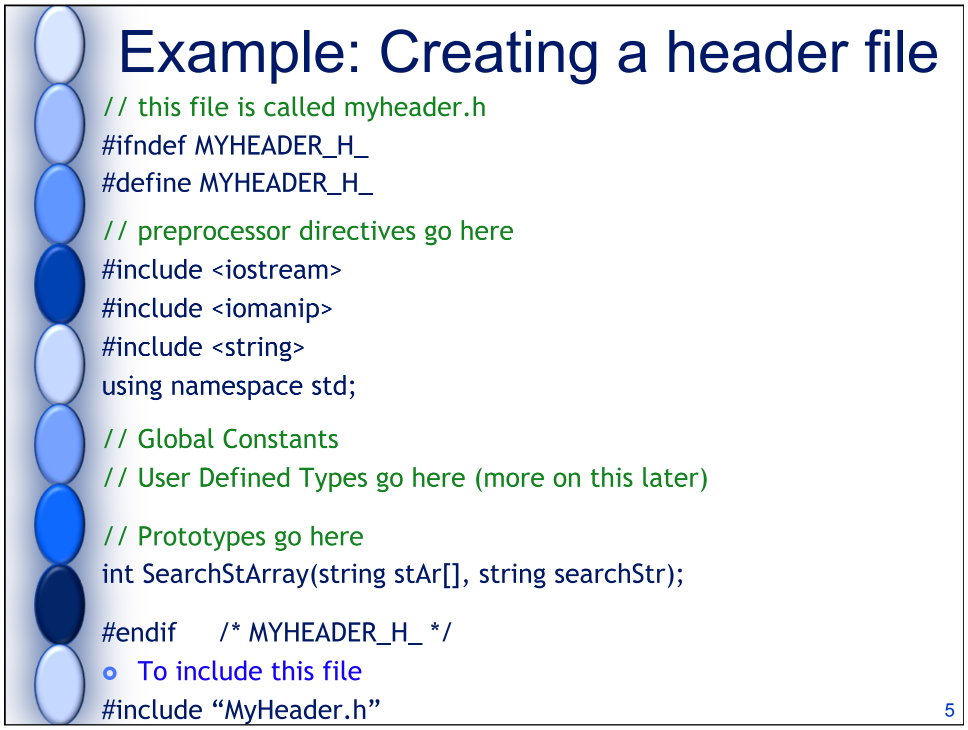
Header File

```
// these two lines and the last one ensure that you
//   don't accidentally make the same definitions twice – it is a good
//   practice to include them
// this example assumes your header file name is MyHeader.h

#ifndef MYHEADER_H_
#define MYHEADER_H_

<your preprocessor directives>
<global constants>
<your typedefs and enumerated types>
<your function prototypes>
#endif
```

NOTE:
eclipse will automatically include the lines of code that are in black
→ you **MUST** insert your preprocessor directives, tyedefs,
and enumerated types as specified



Example: Creating a header file

```
// this file is called myheader.h
#ifndef MYHEADER_H_
#define MYHEADER_H_
// preprocessor directives go here
#include <iostream>
#include <iomanip>
#include <string>
using namespace std;
// Global Constants
// User Defined Types go here (more on this later)
// Prototypes go here
int SearchStArray(string stAr[], string searchStr);
#endif     /* MYHEADER_H_ */
```

- To include this file

```
#include “MyHeader.h”
```

5

Some points to mention

- you must use quotes in your header file
  - "MyHeader.h" → NOT <MyHeader.h>
- the file must be located in your project folder
  - otherwise C++ can't find it

Common Errors

- Make sure your files are all in the same folder
- Make sure that you have your preprocessor directives BEFORE your prototypes
  - ORDER MATTERS
    - 1 – preprocessor directives
      - **# includes & namespace**
    - 2 – global constants
    - 3 – typedefs and enumerated types
    - 4 – prototypes
- You can't have code in the header file
- You can have code in a separate file
- You can only have 1 int main()

Using Input / Output Files

I/O Files

- Instead of using keyboard as input and the screen as output, we can use files

File I/O is a 5-step process

1. Include the header file fstream
2. Declare the file stream variables
3. Associate the file stream variables with the I/O sources
4. Use the file stream variables with >>, << or other I/O functions
5. Close the files

File I/O - Details

- Include the *fstream* headerfile
  - #include <fstream>
- Declare the file stream variables
  - ifstream *inFile*; ← declares the input file stream
  - ofstream *outFile*; ← declares the output file stream
- Open the files
  - *inFile*.open("*inFileName.txt*"); ← opens the input file
  - *outFile*.open("*outFileName.txt*"); ← opens the output file
- Close the files (when you are done with them)
  - *inFile*.close(); ← closes the input file
  - *outFile*.close(); ← closes the output file

EXAMPLE

```
#include <fstream>
int main()
{  …
    ifstream  inFile;
    ofstream outFile;

    // opens the file named InputFile.txt as an input file
    inFile.open("InputFile.txt");
    // opens the file named OutputFile.txt as an output file
    outFile.open("OutputFile.txt");

    // reads a name in from inFile and puts the data in the variable name
    getline(inFile,name);
    inFile >> id;
    // outputs the variable payrate to outData
    outFile << payRate <<  endl;
    // don't forget to close your files
     inFile.close();
     outFile.close();
}
```

NOTE: Output manipulators can be used with files too

Dynamically Naming a File

- To dynamically identify your input file
  (take the filename in as input)
  - The string must be null terminated
  - Data type *string* is not null terminated

- 2 options
  - Declare a c-string
    - char fileName[25];
  - Convert the string to a c-string
    (i.e. make it null terminated) with .c_str()
    - string fileName;
    - fileName.c_str()

Dynamically Naming a File (2)

Given:

```
#include <fstream>
…
ifstream iFile;
```

Example – using a c-string

```
char inFileName[25];
cout << “Enter an Input File Name: ”
getline(cin, inFileName);
iFile.open(inFileName);
```

Example – using a string – THIS WAY IS BETTER → WHY?

```
string inFileName;
cout << “Enter an Input File Name: ”
getline(cin, inFileName);
iFile.open(inFileName.c_str());
```

Create Your Input File First

- Go to File → New → File
- Make sure the files are in your project folder
  - Output files will auto generate
  - Input files won't
- Eclipse doesn't need these files to exist
  - →BUT if you want it to read input you need to identify it somewhere does need the input file

Passing Files

- If you need to use an input file in two functions you need to pass as a parameter
  - You can't just open and close the file
  - Must be passed by reference (use the &)

EXAMPLE

```
void PrintHeaderToFile(ofstream &oFile,// IN/OUT - output file
                  string asName,        // IN - assignment Name
                  char asType,          // IN -  assignment type
                                        //     (LAB or ASSIGNMENT)
                  int asNum);           // IN - assignment number
int main ()
{

       ofstream outFile;                // OUT - Output File
       outFile.open("output.txt");

       // output header for this lab
       PrintHeaderToFile(outFile, "Functions", 'A', 14);

       outFile << "I can output from here now too";
       outFile.close();
       return 0;
}
```

Including code in another file

- Create a .cpp file
- Ensure it is contained in the same folder
- Include whatever preprocessor directives you need for the functions in that file to run

This can be placed in a separate file

```
#include <string>
#include <iostream>
#include <iomanip>
#include <fstream>
using namespace std;
void PrintHeaderToFile(ofstream &oFile,   // IN/OUT - output file
                       string    asName,  // IN - assignment Name
                       char      asType,  // IN - assignment type
                       int       asNum)   // IN - assignment number
{
  oFile << left;
  oFile << "***************************************************\n";
  oFile << "*  PROGRAMMED BY : Michele Rousseau \n";
  oFile << "*  " << setw(14) << "STUDENT ID"  << ": 7502312\n";
  oFile << "*  " << setw(14) << "CLASS" << ": CS1B --> MW - 6p-7:30p\n";
  oFile << "*  ";

  if (toupper(asType) == 'L')
  {
     oFile << "LAB #" << setw(9);
  }
  else
  {
     oFile << "ASSIGNMENT #" << setw(2);
  }
  oFile << asNum << ": " << asName << endl;
  oFile << "***************************************************\n\n";
  oFile << right;
}
```

Ostream & Ostringstream

Ostream & ostringstream

When we think about our print header function

```
void PrintHeaderToFile(ofstream &oFile, // IN/OUT - output file
                       string    asName, // IN - assignment Name
                       char      asType, // IN - assignment type
                       int       asNum)  // IN - assignment number

{
   oFile << left;
   oFile << "***************************************************\n";
   oFile << "*  Programmed by : Michele Rousseau \n";
   oFile << "\n*  " << setw(14) << "Student ID"  << ": 7502312";
   …
```

What is different between this and when we output our function to the screen?

- `oFile <<   vs  cout <<`
- Remember oFile and cout are variables
  - Why can't we pass them in as arguments?

BECAUSE THEY ARE DIFFERENT DATA TYPES!

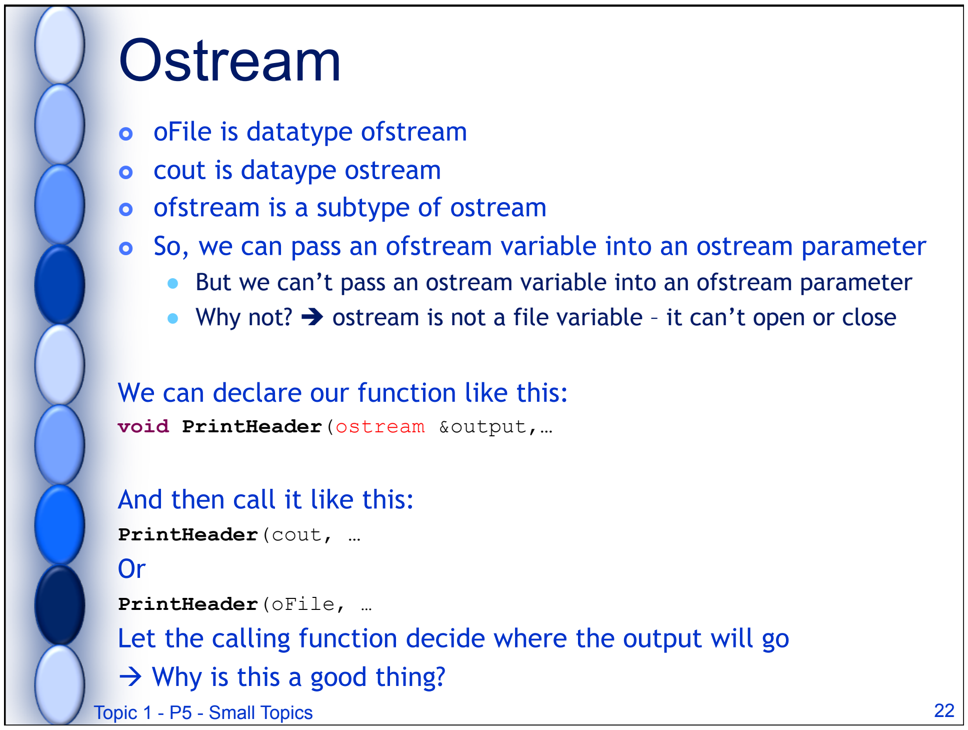
- There are two solutions to writing 2 separate functions
  - ostream or
  - ostringstream

Output Stream

- Output stream datatype can be used to represent different types of output objects such as files, console and output string

$$\begin{tikzcd}
\boxed{\text{ios\_base}} & \boxed{\text{ios}} \arrow[l] & \boxed{\text{ostream}} \arrow[l] & \boxed{\text{iostream}} \arrow[l] \\
 & & & \boxed{\text{ofstream}} \arrow[ul] \\
 & & & \boxed{\text{ostringstream}} \arrow[uul]
\end{tikzcd}$$

- We can use an ostream datatype to allow a function to output either to a file or to console (cout)



Ostream

- oFile is datatype ofstream
- cout is dataype ostream
- ofstream is a subtype of ostream
- So, we can pass an ofstream variable into an ostream parameter
  - But we can't pass an ostream variable into an ofstream parameter
  - Why not? ➔ ostream is not a file variable – it can't open or close

We can declare our function like this:

```
void PrintHeader(ostream &output,…
```

And then call it like this:

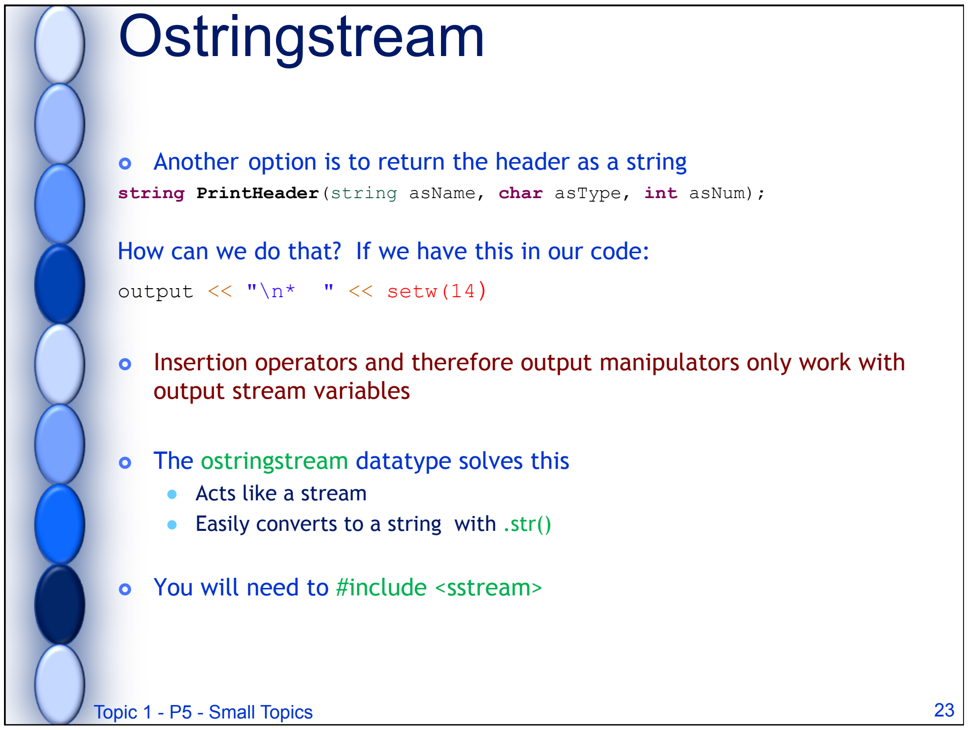
```
PrintHeader(cout, …
```

Or

```
PrintHeader(oFile, …
```

Let the calling function decide where the output will go

→ Why is this a good thing?



Ostringstream

- Another option is to return the header as a string

```
string PrintHeader(string asName, char asType, int asNum);
```

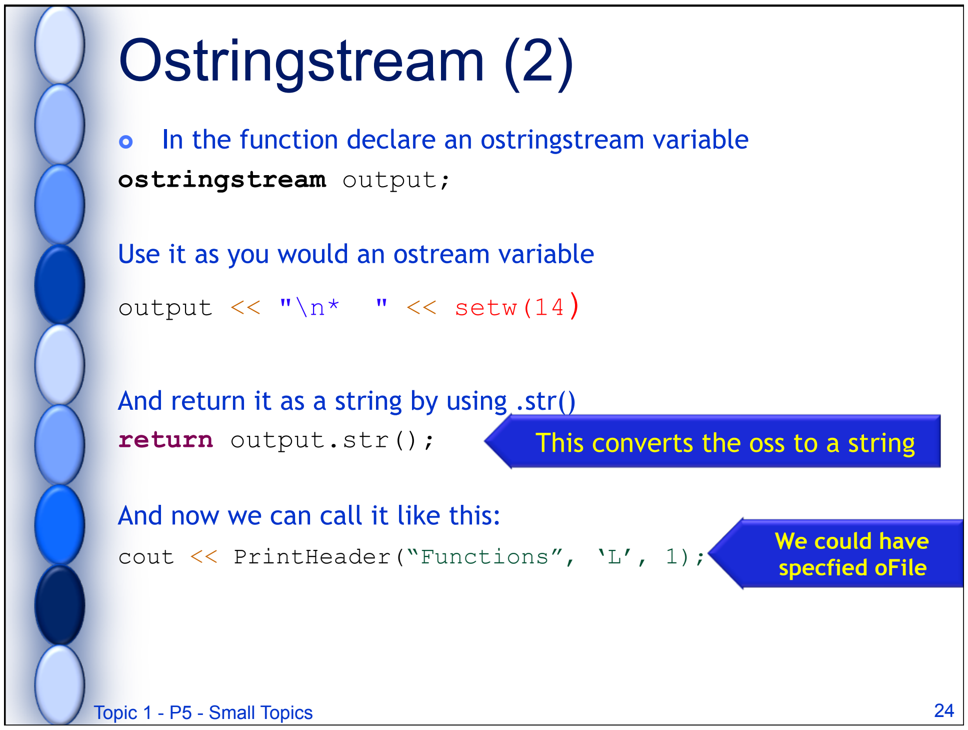
How can we do that? If we have this in our code:

```
output << "\n*   " << setw(14)
```

- Insertion operators and therefore output manipulators only work with output stream variables

- The ostringstream datatype solves this
  - Acts like a stream
  - Easily converts to a string with .str()

- You will need to #include <sstream>



Ostringstream (2)

- In the function declare an ostringstream variable

```
ostringstream output;
```

Use it as you would an ostream variable

```
output << "\n*   " << setw(14)
```

And return it as a string by using .str()

```
return output.str();
```

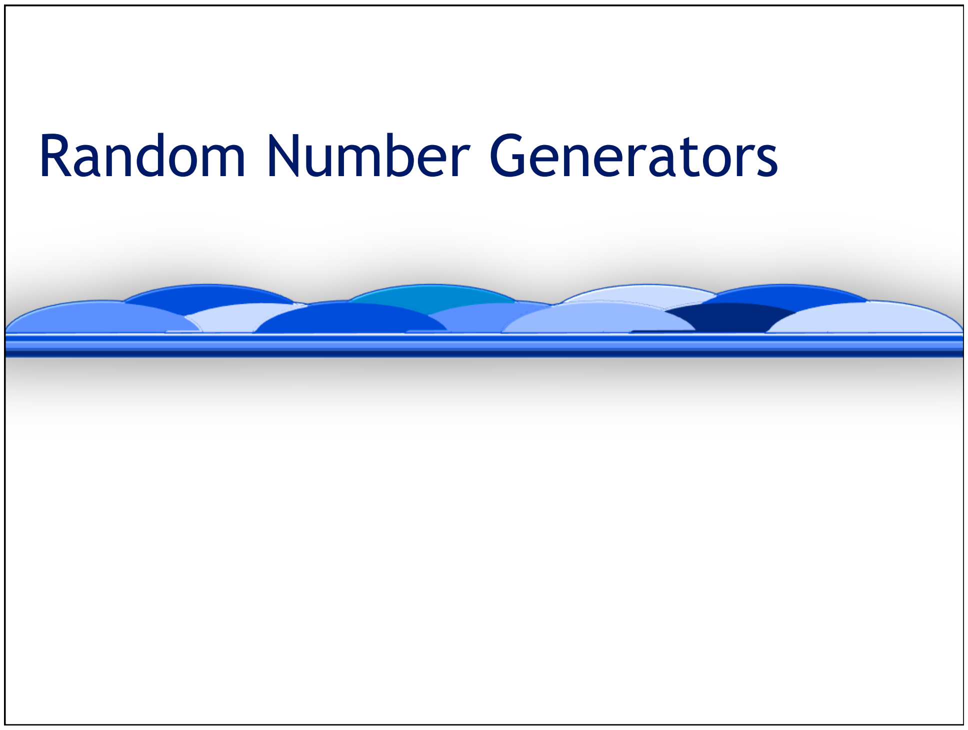
This converts the oss to a string

And now we can call it like this:

```
cout << PrintHeader("Functions", 'L', 1);
```

We could have specfied oFile

Topic 1 - P5 - Small Topics 24



Random Number Generators

Setting the seed for a random value

- To get a random value we need a seed
  - The seed value can be sets the starting value for the random values

Syntax
srand(seed);

  - We will use time as a seed since the time will provide a unique runtime value

Syntax
time(NULL)

This goes in main()

  - So to set the seed based off of the time we write

```
srand(time(NULL));
```

  - The seed should only be set 1X
    - otherwise it will start the set of random values over again
    - meaning it will produce the same value every time

Getting a Random Value

- Finally - when you want a random value

Syntax
rand()

- This will return a random integer from 0 to RAND_MAX

```
myRandomValue = rand();
```

- Use the mod function to get values within a specific range
  rand() % 25 - will give you values from 0 - 24
- For example if I want a random number from 1 to 25

```
myRandomValue = rand() % 25 + 1;
```

You will need to include the following two header files

```
#include <stdlib.h>          /* for srand, rand */
#include <time.h>            /* for time */
```