String Data type & User Defined Files

CS1A

- Strings
  - Declaring them
  - Using them
- User Defined Files
  - Why do we need them?
  - 5 steps to using files
  - Dynamically naming a file
  - Passing files into functions

String Data Type

- Problems with C-strings
  - They are static ← size can't be changed at runtime
  - We have to know how long the string will be ahead of time
    - Or be optimistic
- String fixes these issues

  It handles memory allocation, and makes copying, or assigning values to strings easier

Using Strings

- To use it you must the header file
  - #include <string>
- Declaring a string

```
string name1;
string name2;
```

- Assigning values

```
name1 = "Jean";
name2 = name1;
```

- You can easily concatenate strings using +

```
name2 = "Rousseau";
name1 = name1 + " " + name2 ⇔ name1 = name1 + ' ' + name2;
cout << name1;        ← OUTPUT: Jean Rousseau
```

- You can specify a value of a specific element of a string using the subscript operator []

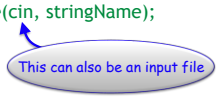
```
cout << name1[3];   ← OUTPUT: n
```

Additional String Functions

- Length & Size
  - Do the same thing → get the length of the string
    - Note this function RETURNS A VALUE so have a place to put it
  - doesn't include \0 but does include spaces

  cout << name1.length(); ← OUTPUT: 13

  cout << name1.size(); ← OUTPUT: 13
- Instead of cin.getline()
  - For strings use getline(cin, stringName);



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: "
cin.getline(inFileName, 25);
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
                                 //      ('L' = Lab,
                                 //       'A' = 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 << "\n*  " << setw(14) << "Student ID"  << ": 7502312";
  oFile << "\n*  " << setw(14) << "Class" << ": CS1B --> MW - 6p-7:30p";
  if (toupper(asType) == 'L')
  {
     oFile << "LAB #" << setw(9);
  }
  else
  {
     oFile << "ASSIGNMENT #" << setw(2);
  }
  oFile << asNum << ": " << asName;
  oFile << "\n************************************************\n\n";
  oFile << right;
```