

String Data type & User Defined Files

CS1A

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

© Michele Rousseau

Strings & Files

1

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

© Michele Rousseau

Strings & Files

2

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`

© Michele Rousseau

Strings & Files

3

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);`

This can also be an input file

© Michele Rousseau

Strings & Files

4

Using Input / Output Files

CS1A

© Michele Rousseau

Strings & Files

5

I/O Files in 5-Steps

- 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

© Michele Rousseau

Strings & Files

6

File I/O - Details

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

© Michele Rousseau

Strings & Files

7

EXAMPLE

```
#include <fstream>
int main()
{
    ...
    ifstream fin;
    ofstream fout;

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

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

NOTE: Output manipulators can be used with files too

© Michele Rousseau

Strings & Files

8

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()`

© Michele Rousseau

Strings & Files

9

Dynamically Naming a File (2)

Given:

```
#include <fstream>
...
ifstream inFile;
```

Example - using a c-string

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

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

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

© Michele Rousseau

Strings & Files

10

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

© Michele Rousseau

Strings & Files

11

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 `&`)

© Michele Rousseau

Strings & Files

12

EXAMPLE

```
void PrintHeaderToFile(ofstream &fout, // IN/OUT - output file
                     string  asName,
                     int     asNum,
                     char    asType = 'L');

int main ()
{
    ofstream oFile;           // OUT - Output File
    oFile.open("output.txt");

    // OUTPUT - class heading
    PrintHeaderToFile(oFile, "Functions", 14, 'A');

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

© Michele Rousseau

Strings & Files

13

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

© Michele Rousseau

Strings & Files

14

```
void PrintHeaderToFile(ofstream &fout, // IN/OUT - output file
                     string  asName, // IN - assignment Name
                     int     asNum,  // IN - assignment number
                     char    asType) // IN - assignment type
// ('L' = Lab, 'A' = Assignment)
{
    fout << left;
    fout << "*****\n";
    fout << "PROGRAMMED BY : Michele Rousseau\n";
    fout << " " << setw(14) << "STUDENT ID" << " : 7502312\n";
    fout << " " << setw(14) << "CLASS" << " : CS1A - MW - 6p-7:30p\n";
    fout << " " ;

    // PROCESSING - This will adjust setws and format appropriately
    // based on if this is a lab 'L' or assignment
    if (toupper(asType) == 'L')
    {
        fout << "LAB #" << setw(9);
    }
    else
    {
        fout << "ASSIGNMENT #" << setw(2);
    }

    fout << asNum << " : " << asName << endl;
    fout << "*****\n";
    fout << right;
}
```

Pre-Processor Directives
#include <string>
#include <iostream>
#include <iomanip>
#include <fstream>
using namespace std;

This should be placed in a separate file

© Michele Rousseau

15