Workshop #6: Classes and resources, IO operators

- Version 1.0

In this workshop, you will implement classes with resources that follow the rule of 3 to be safely copied and assigned.

Learning Outcomes

Upon successful completion of this workshop, you will have demonstrated the abilities to:

- define and create copy constructors
- define and create copy assignment
- Prevent copying and assignment in a class
- Overload insertion operator so the class can be printed using ostream
- Overload extraction operator so the class can be read using istream
- Do simple file IO using ifstream and ofstream
- use the C++ string class to extract an unknown number of characters from the input.

Submission Policy

The workshop is divided into one coding part and one non-coding part:

- Part 1 (**LAB**): A step-by-step guided workshop, worth 100% of the workshop's total mark

  Please note that the part 1 section is **not to be started in your first session of the week**. You should start it on your own before the day of your class and join the first session of the week to ask for help and correct your mistakes (if there are any).

- Part 2 (reflection): non-coding part. The reflection doesn't have marks associated with it but can incur a **penalty of max 40% of the whole workshop's mark** if your professor deems it insufficient (you make your marks from the code, but you can lose some on the reflection).

Due Dates

Part 1 (lab) is due 2 days after your lab day and Part 2 (Reflection) is due 2 days after your lab day.

The Due dates depend on your section. Please choose the "-due" option of the submitter program to see the exact due date of your section:

> Note that the submission usually opens by the end of Monday.
>
> ```
> ~profname.proflastname/submit 2??/wX/pY_sss -due<ENTER>
> ```

- Replace **??** with your subject code (`00 or 44`)
- Replace **X** with Workshop number: [`1 to 10`]
- Replace **Y** with the part number: [`1 or 2`]
- Replace **sss** with the section: [`naa, nbb, nra, zaa, etc...`]

Late penalties

You are allowed to submit your work up to 2 days after the due date with a 50% penalty for each day. After that, the submission will be closed and the mark will be zero.

Citation

Every file that you submit must contain (as a comment) at the top:
your name, **your Seneca email**, **Seneca Student ID** and the **date** when you completed the work.

For work that is done entirely by you (ONLY YOU)

If the file contains only your work or the work provided to you by your professor, add the following message as a comment at the top of the file:

> I have done all the coding by myself and only copied the code that my professor provided to complete my workshops and assignments.

For work that is done partially by you.

If the file contains work that is not yours (you found it online or somebody provided it to you), **write exactly which part of the assignment is given to you as help, who gave it to you, or which source you received it from.** By doing this you will only lose the mark for the parts you got help for, and the person helping you will be clear of any wrongdoing.

> - Add the citation to the file in which you have the borrowed code
> - In the 'reflect.txt` submission of part 2 (Reflection), add exactly what is added to which file and from where (or whom).
>
> :warning: This Submission Policy only applies to the workshops. All other assessments in this subject have their own submission policies.

If you have helped someone with your code

If you have helped someone with your code. Let them know of these regulations and in your 'reflect.txt' of part 2 (Reflection), write exactly which part of your code was copied and who was the recipient of this code.
By doing this you will be clear of any wrongdoing if the recipient of the code does not honour these regulations.

Compiling and Testing Your Program

All your code should be compiled using this command on `matrix`:

```
g++ -Wall -std=c++11 -g -o ws file1.cpp file2.cpp ...
```

- `-Wall`: the compiler will report all warnings
- `-std=c++11`: the code will be compiled using the C++11 standard
- `-g`: the executable file will contain debugging symbols, allowing *valgrind* to create better reports
- `-o ws`: the compiled application will be named `ws`

After compiling and testing your code, run your program as follows to check for possible memory leaks (assuming your executable name is `ws`):

```
valgrind --show-error-list=yes --leak-check=full --show-leak-kinds=all --track-origins=yes ws
```

- `--show-error-list=yes`: show the list of detected errors
- `--leak-check=full`: check for all types of memory problems
- `--show-leak-kinds=all`: show all types of memory leaks identified (enabled by the previous flag)
- `--track-origins=yes`: tracks the origin of uninitialized values (`g++` must use `-g` flag for compilation, so the information displayed here is meaningful).

To check the output, use a program that can compare text files. Search online for such a program for your platform, or use *diff* available on `matrix`.

> Note: All the code written in workshops and the project must be implemented in the **seneca** namespace, unless instructed otherwise.

Custom code submission

If you have any additional custom code, (i.e. functions, classes etc) that you want to reuse in the workshop save them under a module called Utils (`Utils.cpp and Utils.h`) and submit them with your workshop using the instructions in the "Submitting Utils Module" section.

Part 1 - LAB (100%) TextFile module

Your task for this part of the workshop is to complete the implementation of the **TextFile** module. This module attaches itself to a text file on the hard drive and loads the content of the text file into an array of **Line**s.

This module is then capable of displaying the text file page by page on the screen or give the user program read-only access to the lines of the text file as an array of C-strings.

A **Textfile** can be safely copied, and the copying will also result in the creation of a copy of the attached file on the hard drive.

A **TextFile** can be safely assigned to another **TextFile**, and the assignment will also overwrite the contents of the target file on the hard drive with the content of the source file.

Unlike all the other modules that you have created till now (containing the implementation of only one class), the **TextFile** module contains two classes; **Line** and **TextFile**.

The **Line** class encapsulates a single line of a text file. The **TextFile** encapsulates a text file on the hard drive and it is a dynamic array of **Lines**. This means an instance of **Line** class should not be able to exist outside of a **TextFile** class.

To enforce this, make the **Line** class fully private and make the **TextFile** class a friend of the **Line** class. Doing so, only within the scope of the **TextFile** class, a **Line** can be instantiated and accessed:

```
      class TextFile; // forward declaration
class Line{  // fully private, no public member
    ........
    friend class TextFile;
};
class TextFile{
    public:
    .......
};
```

The Line Class

The **Line** class is fully private and should only be accessible by the **TextFile** class.

```
      class Line {
   char* m_value;
   Line();
   ~Line();
   operator const char* ()const;
   Line& operator=(const char* lineContent);
   // incomplete...

};
```

Attributes (Member variables)

char* m_value

holds the address of the dynamically allocated Cstring (to hold a line of the text file)

Methods (Member functions)

operator const char* ()const;

Returns the address held in the **m_value** attribute.

Operator=(const char*) overload

Dynamically allocates memory in **m_value** and copies the Cstring pointed by **lineContent** into it.

default constructor

Initializes the **m_value** attribute to **nullptr**.

destructor

Makes sure all the allocated memory is freed.

> Make sure **Line** can not be copied or assigned to another **Line**.

The TextFile Class

```
   class TextFile {
   Line* m_textLines;
   char* m_filename;
   unsigned m_noOfLines;
```

```
      unsigned m_pageSize;
      void setFilename(const char* fname, bool isCopy = false);
      void setNoOfLines();
      void loadText();
      void saveAs(const char *fileName)const;
      void setEmpty();
   public:
      TextFile(unsigned pageSize = 15);
      TextFile(const char* filename, unsigned pageSize = 15);
      TextFile(const TextFile&);
      TextFile& operator=(const TextFile&);
      ~TextFile();
      std::ostream& view(std::ostream& ostr)const;
      std::istream& getFile(std::istream& istr);
      operator bool()const;
      unsigned lines()const;
      const char* name()const;
      const char* operator[](unsigned index)const;
   };
```

Attributes (Member variables)

TextFile has four private member variables:

```
Line* m_textLines;
```

A pointer to hold the dynamic array of **Lines**. This attribute should be initialized to **nullptr**

```
char* m_filename;
```

A pointer to hold the dynamic Cstring holding the name of the file. This attribute should be initialized to **nullptr**

```
unsigned m_noOfLines;
```

An unsigned integer to be set to the number of lines in the file.

```
unsigned m_pageSize;
```

The page size is the number of lines that should be displayed on the screen before the display is paused. After these lines are displayed, the user must hit enter for the next page to appear.

Private Methods (Member functions)

setEmpty

```
void setEmpty();
```

deletes the **m_textLines** dynamic array and sets is to **nullptr** deletes the **m_filename** dynamic Cstring and sets is to **nullptr** sets **m_noOfLines** attribute to **zero**.

setFilename

```
void setFilename(const char* fname, bool isCopy = false);
```

If the isCopy argument is false, dynamically allocates a Cstring in **m_filename** and copies the content of the **fname** argument into it. If the isCopy argument is true, dynamically allocates a Cstring in **m_filename** and copies the content of the **fname** argument with a prefix of **"C_"** attached to it.

```
      Example:
setFilename("abc.txt"); // sets the m_filename to "abc.txt"
setFilename("abc.txt", true); // sets the m_filename to "C_abc.txt"
```

setNoOfLines

```
void setNoOfLines();
```

Counts the number of lines in the file:

Creates a local **ifstream** object to open the file with the name held in **m_filename**. Then it will read the file, character by character, and accumulates the number of newlines in the **m_noOfLines** attribute.

In the end, it will increase **m_noOfLines** by one, just in case, the last line does not have a new line at the end.

If the number of lines is zero, it will delete the m_filename and set it to nullptr. (Setting the TextFile to a safe empty state)

loadText

```
void loadText();
```

Loads the text file **m_filename** into the dynamic array of Lines pointed by **m_textLines** :
If the **m_filename** is null, this function does nothing.

If the **m_filename** is not null (**TextFile** is not in a safe empty state), **loadText()** will dynamically allocate an array of Lines pointed by **m_textLines** with the size kept in m_noOfLines.

*Make sure **m_textLine** is deleted before this to prevent memory leak.*

Create a local instance of **ifstream** using the file name **m_filename** to read the lines of the text file.

Since the length of each line is unknown, read the line using a local C++ string object and the **getlinehelper** function. (note: this is the **HELPERgetline** function and not a method of istream).

In a loop reads each line into the string object and then sets the **m_textLines** array elements to the values returned by the **c_str()** method of the string object until the reading fails (end of file reached).

After all the lines are read, make sure to update the value of m_noOfline to the actual number of lines read (This covers the possibility of one extra empty line at the end of the file)

saveAs

```
void saveAs(const char *fileName)const;
```

Saves the content of the TextFile under a new name.

Use a local ofstream object to open a new file using the name kept in the argument filename. Then loop through the elements of the m_textLines array and write them in the opened file adding a new line to the end of each line.

Constructors

```
TextFile(unsigned pageSize = 15);
```

Creates an empty TextFile and initializes the m_pageSize attribute using the pageSize argument.

```
TextFile(const char* filename, unsigned pageSize = 15);
```

Initializes the m_pageSize attribute using the pageSize argument and all the other attributes to nullptr and zero. Then if the filename is not null, it will set the filename, set the number of Lines and load the Text (using the corresponding private methods.)

Rule of three implementations for classes with resource

Implement The Copy Constructor, Copy assignment and destructor.

Copy Constructor

Initializes the m_pageSize attribute using the m_pageSize of the incoming TextFile object and all the other attributes to nullptr and zero. If the incoming Text object is in a valid State, performs the following tasks to copy the textfile and the content safely:

- Sets the file-name to the name of the incoming TextFile object (isCopy set to true) See setFilename()
- Saves the content of the incoming TextFile under the file name of the current TextFile
- set the number of lines
- loads the Text

Copy Assignment

If the current and the incoming TextFiles are valid it will first delete the current text and then overwrites the current file and data by the content of the incoming TextFile.

- deallocate the dynamic array of Text and sets the pointer to null
- Saves the content of the incoming TextFile under the current filename
- Sets the number of lines
- loads the Text

destructor

Deletes all the dynamic data.

public Methods

lines()

```
unsigned lines()const;
```

returns m_noOfLines;

view()

```
std::ostream& view(std::ostream& ostr)const;
```

Prints the filename and then the content of the file "m_pageSize" lines at a time.

- print the file name
- underline the file name with '=' character
- loops through the lines and print them one by line
- pauses after printing "m_pageSize" lines and prompts the user `Hit ENTER to continue...` and waits for the user to press enter and repeats until all lines are printed.

The function performs no action if the **TextFile** is in an empty state.

This function receives and returns an instance of istream and uses the instance for printouts.

getFile()

```
std::istream& getFile(std::istream& istr);
```

Receives a filename from istr to set the file name of the TextFile. Then sets the number of lines and loads the Text. When done it will return the istr;

index Operator Overload

```
const char* operator[](unsigned index)const;
```

Returns the element in the m_textLine array corresponding to the index argument. If the TextFile is in an empty state, it will return null. If the index exceeds the size of the array it should loop back to the beginning.

For example, if the number of lines is 10, the last index should be 9 and index 10 should return the first element and index 11 should return the second element.

Type conversion overloads:

boolean cast

```
operator bool()const;
```

Returns true if the TextFile is not in an empty state and returns false if it is.

constant character pointer cast

```
const char* name()const;
```

Returns the filename.

Insertion and extraction helper operator overloads

To print and read on and from istream and ostream overload operator<< and operator>>:

operator<<

```
ostream& operator<<(ostream& ostr, const TextFile& text);
```

uses the view() method to print the TextFile

operator>>

```
istream& operator>>(istream& istr, TextFile& text);
```

uses the getFile() method to get the file name from console and load the content from the file

Tester program

```
/***********************************************************************
// Intro. to Object Oriented Programming
// Workshop 6
// Version 1.0
// Description
// professor's tester program
//
// Revision History
// -----------------------------------------------------------
// Name            Date            Reason
//
/////////////////////////////////////////////////////////////////
***********************************************************************/
#include <iostream>
#include <fstream>
#include <string>
#include "TextFile.h"
using namespace sdds;
using namespace std;
void FirstTen(TextFile T);
void Copy(const string& dest, const string& source);
void Dump(const string& filename);
int main() {
   TextFile Empty;
   TextFile BadFilename("badFilename");
   Copy("echoes.txt", "echoesOriginal.txt");
   Copy("seamus.txt", "seamusOriginal.txt");
   TextFile E;
   TextFile S("seamus.txt");
   cout << "Enter the text file name: ";
   cin >> E;
   cout << E << endl;
   cout << S << endl;
   FirstTen(E);
   FirstTen(S);
   E = S;
   cout << E << endl;
   cout << "============================================================" << endl;
   Dump("echoes.txt");
   Dump("seamus.txt");
   Dump("C_echoes.txt");
   Dump("C_seamus.txt");
   cout << "*" << Empty << BadFilename << "*" << endl;
   return 0;
}
void FirstTen(TextFile T) {
   if (T) {
      cout << ">>> First ten lines of : " << T.name() << endl;
      for (unsigned i = 0; i < 10; i++) {
         cout << (i + 1) << ": " << T[i] << endl;
      }
   }
   else {
      cout << "Nothing to print!" << endl;
   }
   cout << endl << "-------------------------------------------------------------" << endl;
}
void Dump(const string& filename) {
   cout << "DUMP---------------------------------------------------------" << endl;
   cout << ">>>" << filename << "<<<" << endl ;
   ifstream fin(filename.c_str());
   char ch;
   while (fin.get(ch)) cout << ch;
   cout << endl << "-------------------------------------------------------------" << endl;
}
void Copy(const string& dest, const string& source) {
   ifstream fin(source.c_str());
   ofstream fout(dest.c_str());
   char ch;
   while (fin.get(ch)) fout << ch;
}
```

Execution sample

Output Sample

PART 1 Submission

Files to submit

```
      TextFile.h
TextFile.cpp
w6p1_tester.cpp
```

Submission Process:

Upload the files listed above to your `matrix` account. Compile and run your code using the `g++` compiler as shown in Compiling and Testing Your Program and make sure that everything works properly.

Then, run the following command from your matrix account

```
~profname.proflastname/submit 2??/wX/pY_sss  <ENTER>
```

- Replace **??** with your subject code (`00 or 44`)
- Replace **X** with Workshop number: [`1 to 10`]
- Replace **Y** with the part number: [`1 or 2`]
- Replace **sss** with the section: [`naa, nbb, nra, zaa, etc...`]

and follow the instructions.

Submitting Utils Module

To have your custom Utils module compiled with your workshop and submitted, add a **u** to the part number of your workshop (i.e **u**p1 for part one and **u**p2 for part two) and issue the following submission command instead of the above:

```
~profname.proflastname/submit 2??/wX/upY_sss  <ENTER>
```

See Custom Code Submission section for more detail

> :warning:**Important:** Please note that a successful submission does not guarantee full credit for this workshop. If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Re-submissions will attract a penalty

Part 2: Reflection

Study your final solutions for each deliverable of the workshop **and the most recent milestones of the project if applicable**, reread the related parts of the course notes, and make sure that you have understood the concepts covered by this workshop. **This should take no less than 30 minutes of your time and the result is suggested to be between 150 to 300 words in length.**

Create a file named `reflect.txt` that contains your detailed description of the topics that you have learned in completing this workshop and **the project milestones if applicable** and mention any issues that caused you difficulty.

Reflection Submission Process:

Upload `reflect.txt` to matrix containing the reflection

Then, run the following command from your matrix account

```
~profname.proflastname/submit 2??/wX/pY_sss  <ENTER>
```

- Replace **??** with your subject code (`00 or 44`)
- Replace **X** with Workshop number: [`1 to 10`]
- Replace **Y** with the part number: [`1 or 2`]
- Replace **sss** with the section: [`naa, nbb, nra, zaa, etc...`]

and follow the instructions.

> :warning:**Important:** Please note that a successful submission does not guarantee full credit for this workshop. If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Re-submissions will attract a penalty. act a penalty