Program structure and explanation
MD Harrington 09/05/2023

Hi all this is how I structured my solution for reading of text-files so that

Usage :: method of prompt for
media and explanations

The tree structure looks of my setup looks like this

Root folder (The Parent of the project) , contains the following

```
(Parent folder)
CMakeLists.txt
   dist  (Directory)
      check.txt
   include (Directory)
      ColorChooser.hpp
      PrintWriter.hpp
   src (Directory)
      ColorChooser.cpp
      main.cpp
      PrintWriter.cpp
```

The ColorChooser class explained

```
#ifndef COLORCHOOSER_HPP
#define COLORCHOOSER_HPP
#include <string>
#include <iostream>


using namespace std ;

class ColorChooser {
public:
// Constructor
ColorChooser();

// Destructor
~ColorChooser();

// Method to present user with a menu of colors to pick from using ansi escape sequences
void setColor();
string getColorSet() ;
```

Please continue

```
string getColorchars();
private:
string _reset;
string _colourchars ;
string _COLOR ;
};

#endif // COLORCHOOSER_HPP
```

The provided code defines a class named ColorChooser has a constructor, a destructor, and three member functions named setColor, getColorchars, and getColorSet.

```
ColorChooser::ColorChooser()
{
cout << "In constructor" << endl ;
}
```

The above code or constructor is called when an object of the ColorChooser class is created. In this case, it prints the message "In constructor" to the console.

```
ColorChooser::~ColorChooser()
{
cout << "In destructor of ColorChooser class , cleaning up " << endl <<
"Code written by MD Harrington date 09/05/2023" << endl <<
"FaceBook https://www.facebook.com/mark.harrington.14289/" << endl <<
"Instagram https://www.instagram.com/markukh2021/" << endl ;

}
```

The destructor in above code is called when an object of the ColorChooser class is destroyed. In this case, it prints a message to the console, including the name and date of the author, as well as links to their Facebook and Instagram profiles.

Please continue

```
void ColorChooser::setColor()

{

cout << endl ;
cout << "Choose a color:" << endl;
cout << "\033[30m 1. Black \033[0m" << endl;
cout << "\033[31m 2. Red \033[0m" << endl;
cout << "\033[33m 3. Yellow \033[0m" << endl;
cout << "\033[34m 4. Blue \033[0m" << endl;
cout << "\033[35m 5. Magenta \033[0m" << endl;
cout << "\033[36m 6. Cyan \033[0m" << endl;
cout << "\033[37m 7. White \033[0m" << endl;
cout << "\033[0m 8. Reset \033[0m" << endl;

int choice;
cout << "Enter choice :" ;
cin >> choice;

switch (choice)
{
case 1:
_COLOR = "\033[30mBlack\033[0m";
_colourchars = "\033[30m" ;
break;
case 2:
_COLOR = "\033[31mRed\033[0m";
_colourchars = "\033[31m" ;
break;
case 3:
_COLOR = "\033[33mYellow\033[0m";
_colourchars = "\033[33m" ;
break;
case 4:
_COLOR = "\033[34mBlue\033[0m";
_colourchars = "\033[34m" ;
break;
case 5:
_COLOR = "\033[35mMagenta\033[0m";
_colourchars = "\033[35m" ;
break;
case 6:
_COLOR = "\033[36mCyan\033[0m";
_colourchars = "\033[36m" ;
break;
case 7:
```

Please continue

```
_COLOR = "\033[37mWhite\033[0m";
_colourchars = "\033[37m" ;
break;

case 8:

_COLOR = "\033[0mReset\033[0m";
_colourchars = "\033[0m" ;
break ;

default:
cout << "Invalid choice. Please choose a number between 1 and 8." << endl;
setColor();
}
}
```

The setColor function prompts the user to choose a color from a list of options and stores the chosen color as a string in a private member variable named _COLOR. It also stores the ANSI escape code associated with the chosen color in another private member variable named _colourchars.

```
string ColorChooser::getColorchars()
{
return _colourchars ;

}
```

The getColorchars function returns the value of the _colourchars member variable.

```
string ColorChooser::getColorSet()
{
return _COLOR ;
}
```

The getColorSet function returns the value of the _COLOR member variable.

Please continue

The PrintWriter Class explained

```
#ifndef PRINTWRITER_HPP
#define PRINTWRITER_HPP


#include <fstream>
#include <string>
#include <vector>
#include <chrono>
#include <thread>

#include "ColorChooser.hpp"

using namespace std ;


class PrintWriter{

public:

// constructor

PrintWriter(const string f , string cc , string charcolor,int delay) ;

~PrintWriter() ;

void printParagraphs();

void printParagraph(const vector<std::string>& paragraph) ;

private:

std::string filename_;
std::ifstream file_;
int delay_;
std::string color_; // Private variable to store the color string
string charcc_ ;
protected:




} ;
```

Please continue

The PrintWriter class , has a constructor, a destructor, and two member functions.

```
PrintWriter::PrintWriter(const string fname, string cc,string charcolor, int delay)
: filename_{fname}, file_{fname}, delay_{delay}, color_{cc} ,charcc_{charcolor}
{
if (!file_)
{
throw std::runtime_error{"Failed to open file: " + filename_};
}
cout << "Color selected was : " << color_ << "\n"
<< "File chosen was : " << filename_ << "\n"
<< "Delay set to : " << delay_ << "\n"
<< endl;
}
```

The constructor takes four parameters: a string fname representing the filename, a string cc representing the color to be used for printing, a string charcolor representing the color of the characters, and an integer delay representing the delay between characters. The constructor opens the file with the given filename and throws a runtime error if the file cannot be opened. It also prints out the color, filename, and delay values.

```
PrintWriter::~PrintWriter()
{
cout << "In destructor PrintWriter class , cleaning up" << endl;
}
```

Please continue

The destructor is responsible for cleaning up any resources used by the PrintWriter object.

The member function printParagraphs reads the file line by line and groups the lines into paragraphs. It then calls the member function printParagraph to print each paragraph.

```
void PrintWriter::printParagraphs()
{
vector<std::string> paragraph;
string line;
while (std::getline(file_, line))
{
if (line.empty())
{
if (!paragraph.empty())
{
printParagraph(paragraph);
paragraph.clear();
}
}
else
{
paragraph.push_back(line);
}
}
if (!paragraph.empty())
{
printParagraph(paragraph);
}
}
```

Please continue

The member function printParagraph takes a vector of strings representing a paragraph and prints it to the console. It iterates over each line in the paragraph, and for each character in the line, it prints the character in the specified colour and sleeps for the specified delay. It also prompts the user to press enter for the next paragraph.

```
void PrintWriter::printParagraph(const vector<std::string> &paragraph)
{
if (paragraph.empty()) {
return;
}
cout << endl ;

for (const auto &line : paragraph)
{
for (const auto &c : line)
{
cout << charcc_ << c << "\033[0m" << flush;
this_thread::sleep_for(std::chrono::milliseconds(delay_));
}
cout << endl;
}
cout << "Enter for next paragraph" << std::flush;
cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
}
```

Overall, the PrintWriter class provides a way to print a file with specified colours and delays between characters.

```
cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
```

This line of code cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n'); is used to clear the input buffer of any remaining characters after reading input from the user via cin.

Here's a breakdown of what each part of the line does:

cin.ignore() - ignores input from the input stream

std::numeric_limits<std::streamsize>::max()

sets the maximum number of characters to be ignored. numeric_limits is a template class that provides a way to get various properties of fundamental data types in C++. In this case, std::streamsize is the type used to represent the size of a stream. max() is a member function that returns the maximum value that can be represented by that type.

Please continue

 '\n' - specifies the delimiter character to stop ignoring input. In this case, it's the newline character, which is the character that is typically added when the user presses the Enter key.

Together, the cin.ignore() function call with the two arguments ignores all remaining characters in the input buffer up to the next newline character, which is then discarded. This ensures that the input buffer is empty and ready for the next input operation.

Main.cpp The Main method entry point explained

```
#include "../include/ColorChooser.hpp"
#include "../include/PrintWriter.hpp"

int main()
{

string colortext;

// you can adjust delay here

int delay = 20;

//ColorChooser *m_colorchooser = new ColorChooser(); // create a pointer to the class
/* Use of auto with uniquepointers instead */

auto p = make_unique<ColorChooser>();

// set up colors for printing now create new instance of PrintWriter
// method shows user a color menu to select color of text printed
p->setColor();

// create a pointer that points to the Printwriter class

auto pw = make_unique<PrintWriter>("check.txt",p->getColorSet(),p->getColorchars(),delay);
/*
PrintWriter *m_printer = new PrintWriter("check.txt", p->getColorSet(),p->getColorchars() , delay);
*/

// call method to print paragraphs
pw->printParagraphs();

/*
clean up when using original pointers
delete m_colorchooser;
delete m_printer ;

*/

return 0;
}
```

The main function is the entry point of the program, where the execution of the program begins.

In this specific program, main does the following:

Declare a string variable colortext and an integer variable delay, which sets the delay between printing each character.

Creates a unique pointer p to an instance of ColorChooser class using the make_unique function.
Calls the setColor() method of the ColorChooser class to display a color menu to the user and allow them to choose the color of the text to be printed.

Creates a unique pointer pw to an instance of PrintWriter class using the make_unique function, passing the name of the output file, the color chosen by the user, the color characters, and the delay as arguments.

Calls the printParagraphs() method of the PrintWriter class to read the file, group the lines into paragraphs, and print each paragraph with the selected color and delay.
Returns 0 to indicate successful execution of the program.

Note: the code also includes commented-out lines that create instances of ColorChooser and PrintWriter using raw pointers instead of unique pointers. These raw pointers must be explicitly deleted using delete to avoid memory leaks

Hope this helps explain all

Thank you

MD Harrington 09/05/23 14:30:05