8.Design a class called cString to represent a string data type. Create   
a data member in the class to represent a string whose size is   
dynamically allocated.

#include <iostream>

#include <cstring>   
class cString {   
private:   
 char\* str;

public:   
 cString() : str(NULL) {}   
 cString(const char\* input) {   
 if (input) {   
 str = new char[strlen(input) + 1];   
 strcpy(str, input);   
 } else {   
 str = NULL;

}   
}   
cString(const cString& other) {   
 if (other.str) {   
 str = new char[strlen(other.str) + 1];   
 strcpy(str, other.str);   
 } else {   
 str = NULL;

}   
}   
cString& operator=(const cString& other) {   
 if (this != &other) {   
 delete[] str;   
 if (other.str) {   
 str = new char[strlen(other.str) + 1];   
 strcpy(str, other.str);   
 } else {

str = NULL;

}   
}   
return \*this;

}   
~cString() {   
 delete[] str;   
}   
cString concatenate(const cString& other) const {   
 cString result;   
 size\_t len1 = str ? strlen(str) : 0;   
 size\_t len2 = other.str ? strlen(other.str) : 0;   
 if (len1 + len2 > 0) {   
 result.str = new char[len1 + len2 + 1];   
 if (str) strcpy(result.str, str);   
 else result.str[0] = '\0';   
 if (other.str) strcat(result.str, other.str);   
 } else {   
 result.str = NULL;

}   
return result;

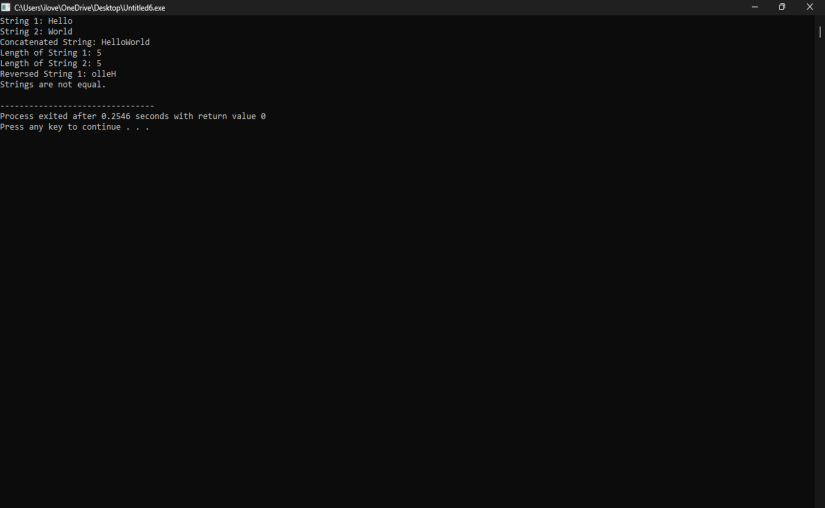
}   
int length() const {   
 return str ? strlen(str) : 0;   
}   
void reverse() {   
 if (str) {   
 int len = strlen(str);   
 for (int i = 0; i < len / 2; i++) {   
 char temp = str[i];   
 str[i] = str[len - i - 1];   
 str[len - i - 1] = temp;   
 }

}   
 }   
 bool compare(const cString& other) const {   
 if (str && other.str) {   
 return strcmp(str, other.str) == 0;   
 }   
 return (str == NULL && other.str == NULL);   
 }   
 void display() const {   
 if (str) std::cout << str;   
 else std::cout << "Empty String";   
 }   
};   
int main() {   
 cString str1("Hello");   
 cString str2("World");   
 std::cout << "String 1: ";   
 str1.display();   
 std::cout << "\nString 2: ";   
 str2.display();   
 cString concatenated = str1.concatenate(str2);   
 std::cout << "\nConcatenated String: ";   
 concatenated.display();   
 std::cout << "\nLength of String 1: " << str1.length();   
 std::cout << "\nLength of String 2: " << str2.length();   
 str1.reverse();   
 std::cout << "\nReversed String 1: ";   
 str1.display();   
 if (str1.compare(str2)) {   
 std::cout << "\nStrings are equal.";   
 } else {   
 std::cout << "\nStrings are not equal.";   
 }

std::cout << std::endl;

return 0;

}



Output: