Documentation: Critical Thinking 1

Alejandro Ricciardi

Colorado State University Global

CSC450: Programming III

Professor: Reginald Haseltine

October 13, 2024

Contents

Documentation: Critical Thinking 1

This documentation is part of the Critical Thinking 1 Assignment from CSC450: Programming III at Colorado State University Global. This Project Report is an overview of three programs' functionality and testing scenarios including console output screenshots. The programs re coded in C++ 23.

The Assignment Direction:

Console Application and Syntax Corrections
Demonstrate an understanding of basic C++ programming concepts by completing the following:

1. Create a simple C++ console application using Eclipse IDE that will accomplish the following:
    1. First Name,
    2. Last Name,
    3. Street Address
    4. City
    5. Zip code

    Prints the following information for a fictional person:
2. Given the provided code in file CSC450_CT1_mod1-1.cpp, correct all syntax errors so that the code will compile correctly.
3. Given the provided code in file CSC450_CT1_mod1-2.cpp, correct all syntax errors so that the code will compile correctly.

Compile and submit your pseudocode, source code, and screenshots of the application executing the application, the results and your GIT repository in a single document.

More Instruction:
To be eligible to receive full credit for your submission you must follow these submission requirements.

1) Put each of your 3 C++ source files into a separate .cpp file. Note that I execute all your programs to check them out.

2) In a Word or PDF "Documentation" file, labeled as such, put a copy of your C++ source code and execution output screen snapshots for each of the 3 programs.

3) Include a screenshot showing a listing of your C++ source code in a GitHub repository on github.com. Do not simply include a link to your GitHub repository.

4) Put all files into a single .zip file, and submit ONLY this .zip file for your solution. Do not submit any other separate Word, PDF, or .cpp files.

⚠️ **My notes:**

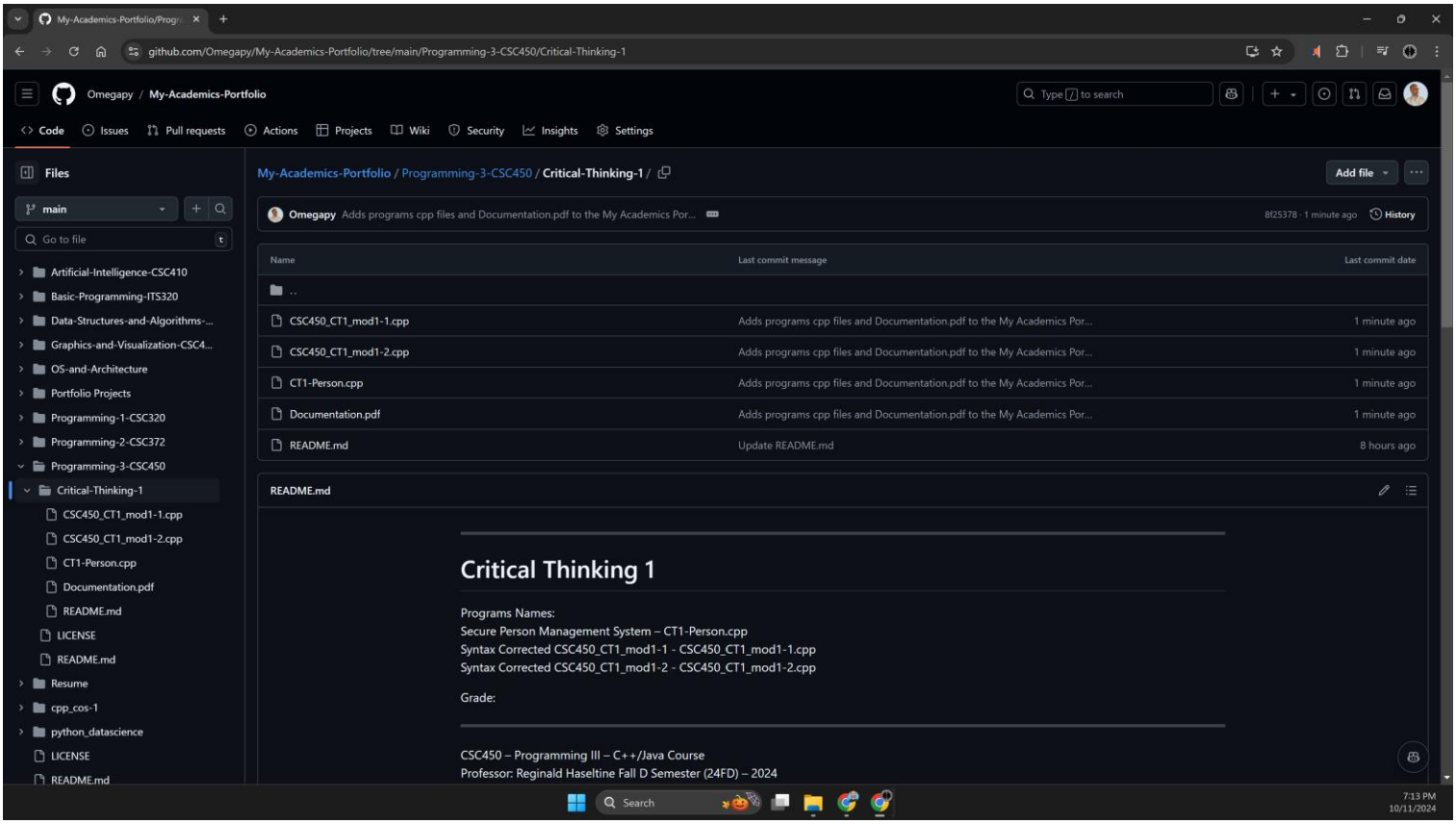
- The simple C++ console application is in file **CTA-1-Person.cpp**
- Corrected code syntax programs are found in the following files:
    - **CSC450_CT1_mod1-1.cpp**
    - **CSC450_CT1_mod1-2.cpp**
- I probably went beyond what was required for the simple C++ console application
- Note: Visual Studio 2022 IDE will give a warning when using `strncpy`, and you'll need to add the line `#define _CRT_SECURE_NO_WARNINGS` to compile the program. Another alternative is using `std::string` or `strncpy_s` for a safer implementation. However, to showcase the vulnerability of buffer overflow when using a char array, this program uses strncpy.

Git Repository

I use GitHub as my Distributed Version Control System (DVCS), the following is a link to my GitHub, Omegapy.
My GitHub repository that is used to store this assignment is named My-Academics-Portfolio.

The link to this specific assignment is: https://github.com/Omegapy/My-Academics-Portfolio/tree/main/Programming-3-CSC450/Critical-Thinking-1

Image of the GitHub repository for this assignment:



Continue next page

The Simple C++ Console Application Description:

Secure Person Management System
This program is a small procedural C++ application that manages an array of Person objects.
It tests and implements secure coding practices to mitigate vulnerabilities such as:

- Buffer overflows
- Integer overflows
- Incorrect type conversions
- Null pointer dereferencing

Note: Visual Studio 2022 IDE will give a warning when using '`strncpy`', and you'll need to add the line '`#define _CRT_SECURE_NO_WARNINGS`' to compile the program. Another alternative is using '`td::string`' or '`strncpy_s`' for a safer implementation. However, to showcase the vulnerability of buffer overflow when using a char array, this program uses '`strncpy`'.

Please see the full program Code in Figure 1

Buffer overflows:
An illustration of the buffer overblows vulnerability management is found in the functions that construct a person data '`createPerson()`' and '`createPersonFull()`', the following is code lines is an example of how the program addresses buffer overblows:

```
// lastNameInput to person.lastName
strncpy(person.lastName, lastNameInput.c_str(), MAX_STRING_LENGTH); // truncates if necessary
person.lastName[MAX_STRING_LENGTH] = '\0'; // Ensure null termination
if (lastNameInput.length() > MAX_STRING_LENGTH) {
    cerr << "Warning --- Input string for lastName exceeded maximum length of "
        << MAX_STRING_LENGTH << " characters and has been truncated." << endl;
}
```

This prevents an inputted string from surpassing the size of the character buffer allocated to data.

Integer overflows
An illustration of the integer overblows vulnerability management is found in the following function:

```
/** ---------------------------------------------------------------------------------------
    Increments numOfPersons and check for integer overflow

    Returns true if the incrementation is successful, false if UINT_MAX is reached
    UINT_MAX, Maximum size of an unsigned int
------------------------------------------------------------------------------------------ **/
bool static incrementNumOfPersons(unsigned& counter) {
    // Check if counter has reached the maximum value for unsigned int
    if (counter == UINT_MAX) {
        // Security measure: Prevent integer overflow by checking maximum value
        cerr << "\nError --- Maximum number of persons reached!" << endl;
        return false;
    }
    // Increment the counter safely
    ++counter;
    return true;
}
```

This prevents an inputted integer from surpassing the size allowed for an unsigned integer data type. Also, note that an unsigned integer is a non-negative.

Incorrect type conversions
An illustration of an incorrect type conversion management is found in the following code lines:

```
// Security measure: Check if personNum is valid
// Note that to comparator '<' comparates the value after the cast
if (static_cast<int>(negativeValue) < 0) {
     cerr << "--- Failed to assign new peraon number ---\n"
         << "\nIncorrect type conversion --- Negative value assigned to personNum.\n"
         << "The value: " << negativeValue << " will cast as a person number: "
         << static_cast<unsigned>(negativeValue)
         << endl;
}
else {
     persons[1].personNum = static_cast<unsigned>(negativeValue);
     cout << "persons[1] number number is: " << persons[1].personNum << "\n"
         << endl;
}
```

This prevents an inputted negative integer from being converted into an unsigned integer, which can only represent non-negative whole numbers.

Null pointer dereferencing
An illustration of null pointer dereferencing virality management is found in the following code lines:

```
// Attempt to access member of null pointer
if (personPtr == nullptr) {
    cerr << "\nPerson pointer is null! Cannot use!." << endl;
}
else {
    cout << "\nPerson first name: " << personPtr->firstName << endl;
}
```

This prevents a null pointer object person value from being cast into a person struct member variable.

Continue next page

Note that in the following struct constructor unused person struct member variables are assigned the value “nan”:

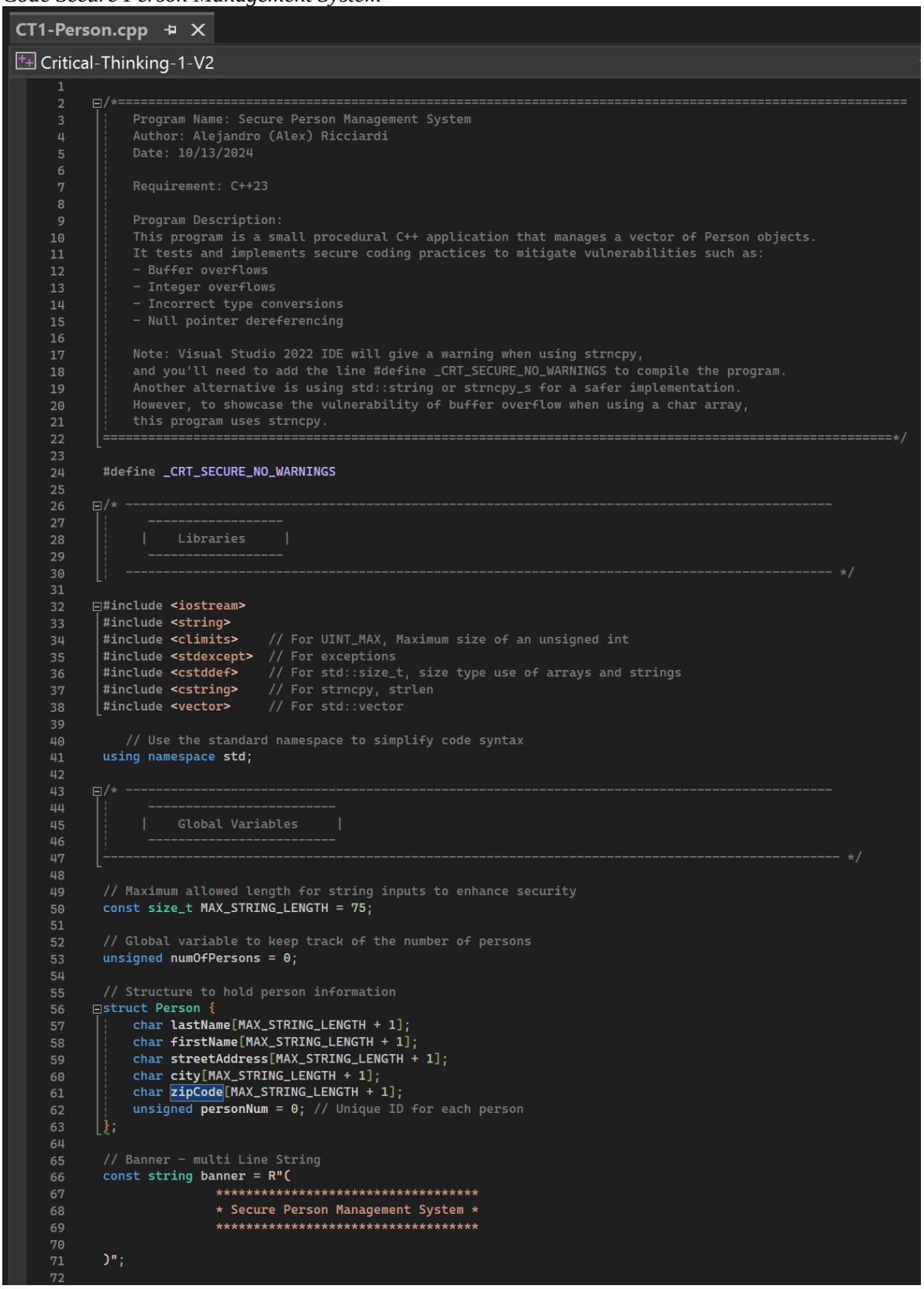
```
/** ------------------------------------------------------------------------------------------
  Creates a new Person with only lastName and firstName arguments inputted
  Uses safe string handling to prevent buffer overflows

  Returns true if person created successfully, false otherwise
------------------------------------------------------------------------------------------ **/
bool static createPerson(Person& person, const string& lastNameInput, const string& firstNameInput)
{

    // Increment the global numOfPersons counter
    if (!incrementNumOfPersons(numOfPersons)) {
        return false;
    }

    // lastNameInput to person.lastName
    strncpy(person.lastName, lastNameInput.c_str(), MAX_STRING_LENGTH); // truncates if necessary
    // C and C++ do not automatically know strings length. Instead,
    // at the end of a string is marked by the null character '\0'
    person.lastName[MAX_STRING_LENGTH] = '\0';
    if (lastNameInput.length() > MAX_STRING_LENGTH) {
        cerr << "Warning --- Input string for lastName exceeded maximum length of "
            << MAX_STRING_LENGTH << " characters and has been truncated." << endl;
    }

    // firstNameInput to person.firstName
    strncpy(person.firstName, firstNameInput.c_str(), MAX_STRING_LENGTH); // truncates if necessary
    person.firstName[MAX_STRING_LENGTH] = '\0'; // Ensure null termination
    if (firstNameInput.length() > MAX_STRING_LENGTH) {
        cerr << "Warning --- Input string for firstName exceeded maximum length of "
            << MAX_STRING_LENGTH << " characters and has been truncated." << endl;
    }

    // Initialize data with "nan" (Not Available)
    strncpy(person.streetAddress, "nan", MAX_STRING_LENGTH);
    strncpy(person.city, "nan", MAX_STRING_LENGTH);
    strncpy(person.zipCode, "nan", MAX_STRING_LENGTH);


    // Assign a unique person number
    person.personNum = numOfPersons;
    cout << "\nA person with number id: " << person.personNum << " was created successfully!"
    << endl;

    return true;
}
```

This prevents a null pointer struct person member value from being passed by accident.

Continue next page

Figure 1

Code Secure Person Management System



CT1-Person.cpp

Critical-Thinking-1-V2

```
/*=============================================================================================================
    Program Name: Secure Person Management System
    Author: Alejandro (Alex) Ricciardi
    Date: 10/13/2024

    Requirement: C++23

    Program Description:
    This program is a small procedural C++ application that manages a vector of Person objects.
    It tests and implements secure coding practices to mitigate vulnerabilities such as:
    - Buffer overflows
    - Integer overflows
    - Incorrect type conversions
    - Null pointer dereferencing

    Note: Visual Studio 2022 IDE will give a warning when using strncpy,
    and you'll need to add the line #define _CRT_SECURE_NO_WARNINGS to compile the program.
    Another alternative is using std::string or strncpy_s for a safer implementation.
    However, to showcase the vulnerability of buffer overflow when using a char array,
    this program uses strncpy.
=============================================================================================================*/

#define _CRT_SECURE_NO_WARNINGS

/* ------------------------------------------------------------------------------------------------
      ------------------
     |    Libraries     |
      ------------------
   ------------------------------------------------------------------------------------------------ */

#include <iostream>
#include <string>
#include <climits>    // For UINT_MAX, Maximum size of an unsigned int
#include <stdexcept>  // For exceptions
#include <cstddef>    // For std::size_t, size type use of arrays and strings
#include <cstring>    // For strncpy, strlen
#include <vector>     // For std::vector

    // Use the standard namespace to simplify code syntax
using namespace std;

/* ------------------------------------------------------------------------------------------------
      -------------------------
     |    Global Variables     |
      -------------------------
   ------------------------------------------------------------------------------------------------ */

// Maximum allowed length for string inputs to enhance security
const size_t MAX_STRING_LENGTH = 75;

// Global variable to keep track of the number of persons
unsigned numOfPersons = 0;

// Structure to hold person information
struct Person {
    char lastName[MAX_STRING_LENGTH + 1];
    char firstName[MAX_STRING_LENGTH + 1];
    char streetAddress[MAX_STRING_LENGTH + 1];
    char city[MAX_STRING_LENGTH + 1];
    char zipCode[MAX_STRING_LENGTH + 1];
    unsigned personNum = 0; // Unique ID for each person
};

// Banner - multi Line String
const string banner = R"(
                ***********************************
                * Secure Person Management System *
                ***********************************

)";
```

```
73    // ============================================================================================================
74    /* ------------------------------------------------------------------------------------------------------------
75          ----------------------------
76        |    Function Declaration    |
77          ----------------------------
78    ------------------------------------------------------------------------------------------------------------ */
79    // ============================================================================================================
80
81    /** ------------------------------------------------------------------------------------------------------------
82     * Increments numOfPersons and check for integer overflow
83     *
84     * Returns true if the incrementation is successful, false if UINT_MAX is reached
85     * UINT_MAX, Maximum size of an unsigned int
86     * ------------------------------------------------------------------------------------------------------------ **/
87    static bool incrementNumOfPersons(unsigned& counter);
88
89    /** ------------------------------------------------------------------------------------------------------------
90     * Creates a new Person with only lastName and firstName arguments inputted
91     * Uses safe string handling to prevent buffer overflows
92     *
93     * Returns true if person created successfully, false otherwise
94     * ------------------------------------------------------------------------------------------------------------ **/
95    static bool createPerson(Person& person, const string& lastName, const string& firstName);
96
97    /** ------------------------------------------------------------------------------------------------------------
98     * Creates a new Person with all arguments inputted
99     * Uses safe string handling to prevent buffer overflows
100    *
101    * Returns true if person created successfully, false otherwise
102    * ------------------------------------------------------------------------------------------------------------ **/
103   static bool createPersonFull(Person& person, const string& lastName, const string& firstName,
104       const string& streetAddress, const string& city,
105       const string& zipCode);
106
107   /** ------------------------------------------------------------------------------------------------------------
108    * Displays the contents of the persons vector
109    * ------------------------------------------------------------------------------------------------------------ **/
110   static void displayPersons(const vector<Person>& persons);
111
112   /** ------------------------------------------------------------------------------------------------------------
113    * Displays a person's data at the given index in the persons vector
114    * ------------------------------------------------------------------------------------------------------------ **/
115   static void displayAPerson(const vector<Person>& persons, size_t index);
116
```

Continue next page

```
// ================================================================================================================
/* ----------------------------------------------------------------------------------------------------------------
     ---------------------
    |    Main Function    |
     ---------------------

    tests secure coding practices such as:
    - Buffer overflows
    - Integer overflows
    - Incorrect type conversions
    - Null pointer dereferencing
---------------------------------------------------------------------------------------------------------------- */
// ================================================================================================================
int main() {

    /* ---------------------------------
     ---------------
    |   Variables   |
     --------------- */

    // Variables used to test code vulnerability
    int negativeValue = -5;      // Used for unsigned int vulnerability
    string longString(100, 'A'); // Create a string with 100 'A's for string overflow
    Person* personPtr = nullptr; // Initialize pointer to null for null pointer vulnerability

    // Vector of persons to store person struct objects
    vector<Person> persons;

    /* ---------------------------------
     -------------
    |   Program   |
     ------------- */

    cout << banner << endl; // std::endl forces flushing of the buffer - good practice

    // ---------------------------------- Test 1: Buffer Overflow with Overly Long Strings

    cout << "-----------------------------------------------------------------\n"
        << "Test 1: Buffer Overflow with Overly Long Strings\n"
        << "Creates a person with first and last names that are 100 characters long, filled with the letter 'A'.\n"
        << endl;
    // Not expected to fail due to safe string handling
    Person person1;
    if (!createPerson(person1, longString, longString)) {
        cout << "\n Error --- Failed to create person ---" << endl;
    }
    persons.push_back(person1);
    displayPersons(persons);

    // ---------------------------------- Test 2: Integer Overflow when Creating Many Persons
    cout << "\n-----------------------------------------------------------------\n"
        << "Test 2: Integer Overflow when Creating Too Many Persons\n"
        << "Simulate numOfPersons = UINT_MAX, Maximum size of an unsigned int\n"
        << "\nTrying to create a new person"
        << endl;

    numOfPersons = UINT_MAX;
    Person personFail
        ;
    if (!createPerson(personFail, "Doe", "John")) {
        // Expected to fail due to integer overflow security
        cout << "Failed to create person due to integer overflow." << endl;
    }
    // Reset numOfPersons for further tests
    numOfPersons = static_cast<unsigned>(persons.size());

```

Continue next page

```cpp
    // ---------------------------------- Test 3: Incorrect Type Conversion
    cout << "\n-----------------------------------------------------------------\n"
        << "Test 3: Incorrect Type Conversion" << endl;

    // new person
    Person person2;
    if (!createPerson(person2, "Conversion", "Alexandria")) {
        cout << "\n Error --- Failed to create person ---" << endl;
    }
    persons.push_back(person2);
    displayAPerson(persons, 1); // Index 1

    cout << "\nTrying to assign persons[1].personNum = " << negativeValue
        <<", which is a negative value\n" << endl;

    // Security measure: Check if personNum is valid
    // Note that to comparator '<' comparates the value after the cast
    if (static_cast<int>(negativeValue) < 0) {
        cerr << "--- Failed to assign new peraon number ---\n"
            << "\nIncorrect type conversion --- Negative value assigned to personNum.\n"
            << "The value: " << negativeValue << " will cast as a person number: "
            << static_cast<unsigned>(negativeValue)
            << endl;
    }
    else {
        persons[1].personNum = static_cast<unsigned>(negativeValue);
        cout << "persons[1] number number is: " << persons[1].personNum << "\n"
            << endl;
    }

    displayAPerson(persons, 1);

    // ---------------------------------- Test 4: Null Pointer Dereferencing with Person Pointer
    cout << "\n-----------------------------------------------------------------\n"
        << "Test 4: Testing Null Pointer before use - null pointer dereferencing\n"
        << "Checking if personPtr is null, and it is."
        << "But if it was not, displaying person first name."
        << endl;

    // Attempt to access member of null pointer
    if (personPtr == nullptr) {
        cerr << "\nPerson pointer is null! Cannot use!." << endl;
    }
    else {
        cout << "\nPerson first name: " << personPtr->firstName << endl;
    }

    // ---------------------------------- Additional test: Properly adding and displaying persons
    cout << "\n-----------------------------------------------------------------\n"
        << "\nAdditional Test: Adding and Displaying Persons" << endl;

    // new persons
    Person person3;
    if (!createPerson(person3, "More", "Bob")) {
        cout << "\n Error --- Failed to create person ---" << endl;
    }
    persons.push_back(person3);

    Person person4;
    if (!createPersonFull(person4, "Marquez", "Anita", "456 Ai Street", "Robot Town", "77442")) {
        cout << "\n Error --- Failed to create person ---" << endl;
    }
    persons.push_back(person4);

    Person person5;
    if (!createPersonFull(person5, "Wan", "Lu", "777 LLM Street", "AI Town", "77772")) {
        cout << "\n Error --- Failed to create person ---" << endl;
    }
    persons.push_back(person5);

    displayPersons(persons);

    return 0;
}
```

```
257
258   // ================================================================================================================
259   /* ----------------------------------------------------------------------------------------------
260          ---------------------------
261        |    Function Definition     |
262          ---------------------------
263      ---------------------------------------------------------------------------------------------- */
264   // ================================================================================================================
265
266     /** ----------------------------------------------------------------------------------------------
267         Increments numOfPersons and check for integer overflow
268
269         Returns true if the incrementation is successful, false if UINT_MAX is reached
270         UINT_MAX, Maximum size of an unsigned int
271       ---------------------------------------------------------------------------------------------- **/
272   bool static incrementNumOfPersons(unsigned& counter) {
273       // Check if counter has reached the maximum value for unsigned int
274       if (counter == UINT_MAX) {
275           // Security measure: Prevent integer overflow by checking maximum value
276           cerr << "\nError --- Maximum number of persons reached!" << endl;
277           return false;
278       }
279       // Increment the counter safely
280       ++counter;
281       return true;
282   }
283
284     // ----------------------------------------------------------------------------------------------
285
286   /** ----------------------------------------------------------------------------------------------
287     Creates a new Person with only lastName and firstName arguments inputted
288     Uses safe string handling to prevent buffer overflows
289
290     Returns true if person created successfully, false otherwise
291   ---------------------------------------------------------------------------------------------- **/
292   bool static createPerson(Person& person, const string& lastNameInput, const string& firstNameInput) {
293
294       // Increment the global numOfPersons counter
295       if (!incrementNumOfPersons(numOfPersons)) {
296           return false;
297       }
298
299       // lastNameInput to person.lastName
300       strncpy(person.lastName, lastNameInput.c_str(), MAX_STRING_LENGTH); // truncates if necessary
301       // C and C++ do not automatically know strings length. Instead,
302       // at the end of a string is marked by the null character '\0'
303       person.lastName[MAX_STRING_LENGTH] = '\0'; // Ensure null termination
304       if (lastNameInput.length() > MAX_STRING_LENGTH) {
305           cerr << "Warning --- Input string for lastName exceeded maximum length of "
306                << MAX_STRING_LENGTH << " characters and has been truncated." << endl;
307       }
308
309       // firstNameInput to person.firstName
310       strncpy(person.firstName, firstNameInput.c_str(), MAX_STRING_LENGTH); // truncates if necessary
311       person.firstName[MAX_STRING_LENGTH] = '\0'; // Ensure null termination
312       if (firstNameInput.length() > MAX_STRING_LENGTH) {
313           cerr << "Warning --- Input string for firstName exceeded maximum length of "
314                << MAX_STRING_LENGTH << " characters and has been truncated." << endl;
315       }
316
317       // Initialize data with "nan" (Not Available)
318       strncpy(person.streetAddress, "nan", MAX_STRING_LENGTH);
319       person.streetAddress[MAX_STRING_LENGTH] = '\0'; // Ensure null termination
320       strncpy(person.city, "nan", MAX_STRING_LENGTH);
321       person.city[MAX_STRING_LENGTH] = '\0'; // Ensure null termination
322       strncpy(person.zipCode, "nan", MAX_STRING_LENGTH);
323       person.zipCode[MAX_STRING_LENGTH] = '\0'; // Ensure null termination
324
325       // Assign a unique person number
326       person.personNum = numOfPersons;
327       cout << "\nA person with number id: " << person.personNum << " was created successfully!" << endl;
328       return true;
329   }
330
331     // ----------------------------------------------------------------------------------------------
332
```

```cpp
/** ------------------------------------------------------------------------------------------
 Creates a new Person with all arguments inputted
 Uses safe string handling to prevent buffer overflows

 Returns true if person created successfully, false otherwise
 ------------------------------------------------------------------------------------------ **/
bool static createPersonFull(Person& person, const string& lastNameInput, const string& firstNameInput,
    const string& streetAddressInput, const string& cityInput,
    const string& zipCodeInput) {

    // Increment the global numOfPersons counter if numOfPersons is not reached
    if (!incrementNumOfPersons(numOfPersons)) {
        return false;
    }

    // lastNameInput to person.lastName
    strncpy(person.lastName, lastNameInput.c_str(), MAX_STRING_LENGTH); // truncates if necessary
    // C and C++ do not automatically know strings length. Instead,
    // at the end of a string is marked by the null character '\0'
    person.lastName[MAX_STRING_LENGTH] = '\0';
    if (lastNameInput.length() > MAX_STRING_LENGTH) {
        cerr << "Warning --- Input string for lastName exceeded maximum length of "
            << MAX_STRING_LENGTH << " characters and has been truncated." << endl;
    }

    // firstNameInput to person.firstName
    strncpy(person.firstName, firstNameInput.c_str(), MAX_STRING_LENGTH); // truncates if necessary
    person.firstName[MAX_STRING_LENGTH] = '\0'; // Ensure null termination
    if (firstNameInput.length() > MAX_STRING_LENGTH) {
        cerr << "Warning --- Input string for firstName exceeded maximum length of "
            << MAX_STRING_LENGTH << " characters and has been truncated." << endl;
    }

    // streetAddressInput to person.streetAddress
    strncpy(person.streetAddress, streetAddressInput.c_str(), MAX_STRING_LENGTH); // truncates if necessary
    person.streetAddress[MAX_STRING_LENGTH] = '\0'; // Ensure null termination
    if (streetAddressInput.length() > MAX_STRING_LENGTH) {
        cerr << "Warning --- Input string for streetAddress exceeded maximum length of "
            << MAX_STRING_LENGTH << " characters and has been truncated." << endl;
    }

    // cityInput to person.city
    strncpy(person.city, cityInput.c_str(), MAX_STRING_LENGTH); // truncates if necessary
    person.city[MAX_STRING_LENGTH] = '\0'; // Ensure null termination
    if (cityInput.length() > MAX_STRING_LENGTH) {
        cerr << "Warning --- Input string for city exceeded maximum length of "
            << MAX_STRING_LENGTH << " characters and has been truncated." << endl;
    }

    // zipCodeInput to person.zipCode
    strncpy(person.zipCode, zipCodeInput.c_str(), MAX_STRING_LENGTH); // truncates if necessary
    person.zipCode[MAX_STRING_LENGTH] = '\0'; // Ensure null termination
    if (zipCodeInput.length() > MAX_STRING_LENGTH) {
        cerr << "Warning --- Input string for zipCode exceeded maximum length of "
            << MAX_STRING_LENGTH << " characters and has been truncated." << endl;
    }

    // Assign a unique person number
    person.personNum = numOfPersons;
    cout << "\nA person with number id: " << person.personNum << " was created successfully!" << endl;
    return true;
}

// ------------------------------------------------------------------------------------------

```

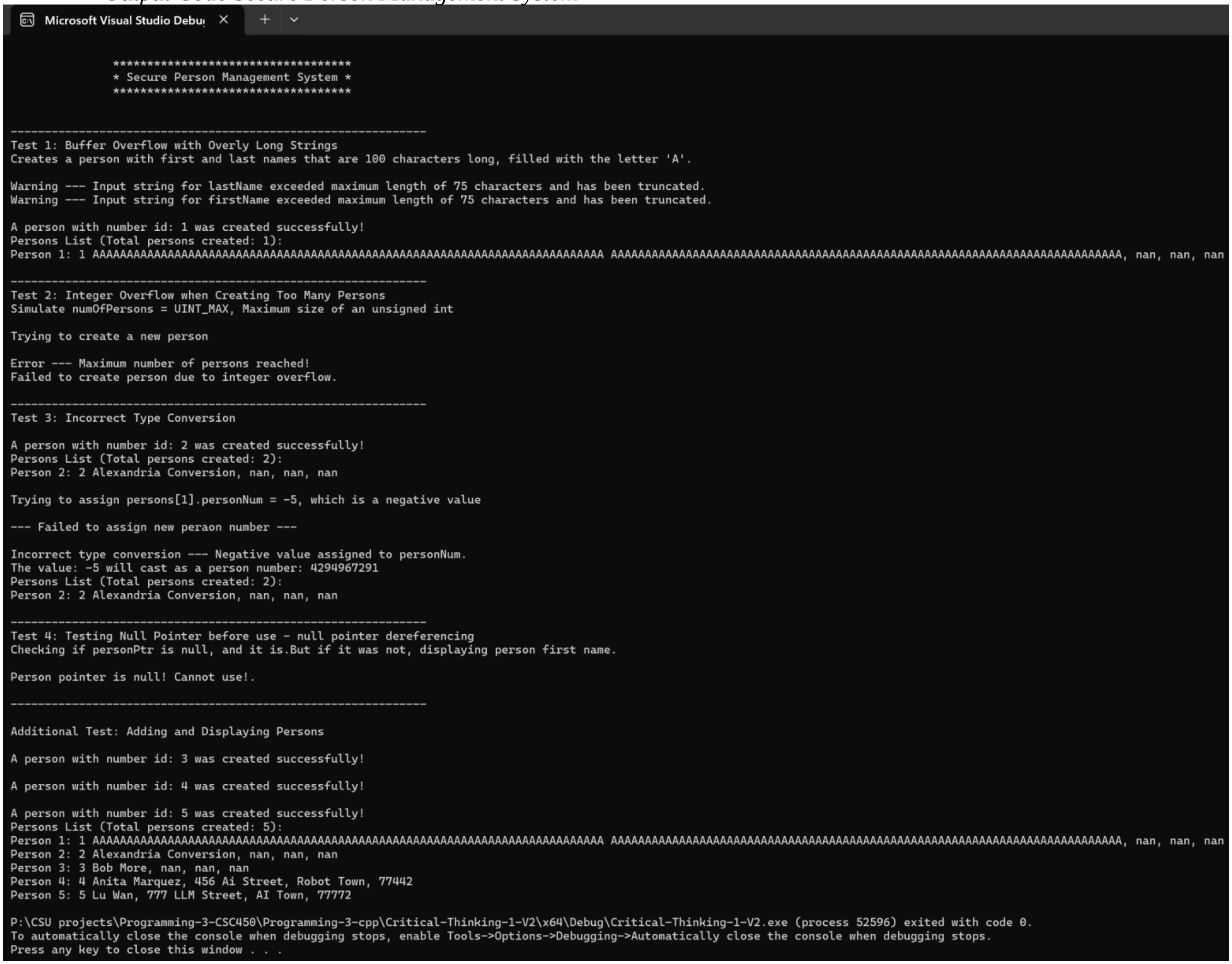
Continue next page

```
/** ------------------------------------------------------------------------------------------
 Displays the contents of the persons vector
 ------------------------------------------------------------------------------------------ **/
void static displayPersons(const vector<Person>& persons) {
    cout << "Persons List (Total persons created: " << numOfPersons << "):" << endl;
    for (size_t i = 0; i < persons.size(); ++i) {
        cout << "Person " << i + 1 << ": "
            << persons[i].personNum << " "
            << persons[i].firstName << " "
            << persons[i].lastName << ", "
            << persons[i].streetAddress << ", "
            << persons[i].city << ", "
            << persons[i].zipCode << endl;
    }
}

// ------------------------------------------------------------------------------------------

/** ------------------------------------------------------------------------------------------
 Displays a person's data at the given index in the persons vector
 ------------------------------------------------------------------------------------------ **/
void static displayAPerson(const vector<Person>& persons, size_t index) {
    if (index < persons.size()) {
        cout << "Persons List (Total persons created: " << numOfPersons << "):" << endl;
        cout << "Person " << index + 1 << ": "
            << persons[index].personNum << " "
            << persons[index].firstName << " "
            << persons[index].lastName << ", "
            << persons[index].streetAddress << ", "
            << persons[index].city << ", "
            << persons[index].zipCode << endl;
    }
    else {
        cout << "Index out of range" << endl;
    }
}

```

Continue next page

Figure 2

Output Code Secure Person Management System



```
Microsoft Visual Studio Debu

            ***********************************
            * Secure Person Management System *
            ***********************************

--------------------------------------------------------------
Test 1: Buffer Overflow with Overly Long Strings
Creates a person with first and last names that are 100 characters long, filled with the letter 'A'.

Warning --- Input string for lastName exceeded maximum length of 75 characters and has been truncated.
Warning --- Input string for firstName exceeded maximum length of 75 characters and has been truncated.

A person with number id: 1 was created successfully!
Persons List (Total persons created: 1):
Person 1: 1 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA, nan, nan, nan

--------------------------------------------------------------
Test 2: Integer Overflow when Creating Too Many Persons
Simulate numOfPersons = UINT_MAX, Maximum size of an unsigned int

Trying to create a new person

Error --- Maximum number of persons reached!
Failed to create person due to integer overflow.

--------------------------------------------------------------
Test 3: Incorrect Type Conversion

A person with number id: 2 was created successfully!
Persons List (Total persons created: 2):
Person 2: 2 Alexandria Conversion, nan, nan, nan

Trying to assign persons[1].personNum = -5, which is a negative value

--- Failed to assign new peraon number ---

Incorrect type conversion --- Negative value assigned to personNum.
The value: -5 will cast as a person number: 4294967291
Persons List (Total persons created: 2):
Person 2: 2 Alexandria Conversion, nan, nan, nan

--------------------------------------------------------------
Test 4: Testing Null Pointer before use - null pointer dereferencing
Checking if personPtr is null, and it is.But if it was not, displaying person first name.

Person pointer is null! Cannot use!.

--------------------------------------------------------------

Additional Test: Adding and Displaying Persons

A person with number id: 3 was created successfully!

A person with number id: 4 was created successfully!

A person with number id: 5 was created successfully!
Persons List (Total persons created: 5):
Person 1: 1 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA, nan, nan, nan
Person 2: 2 Alexandria Conversion, nan, nan, nan
Person 3: 3 Bob More, nan, nan, nan
Person 4: 4 Anita Marquez, 456 Ai Street, Robot Town, 77442
Person 5: 5 Lu Wan, 777 LLM Street, AI Town, 77772

P:\CSU projects\Programming-3-CSC450\Programming-3-cpp\Critical-Thinking-1-V2\x64\Debug\Critical-Thinking-1-V2.exe (process 52596) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

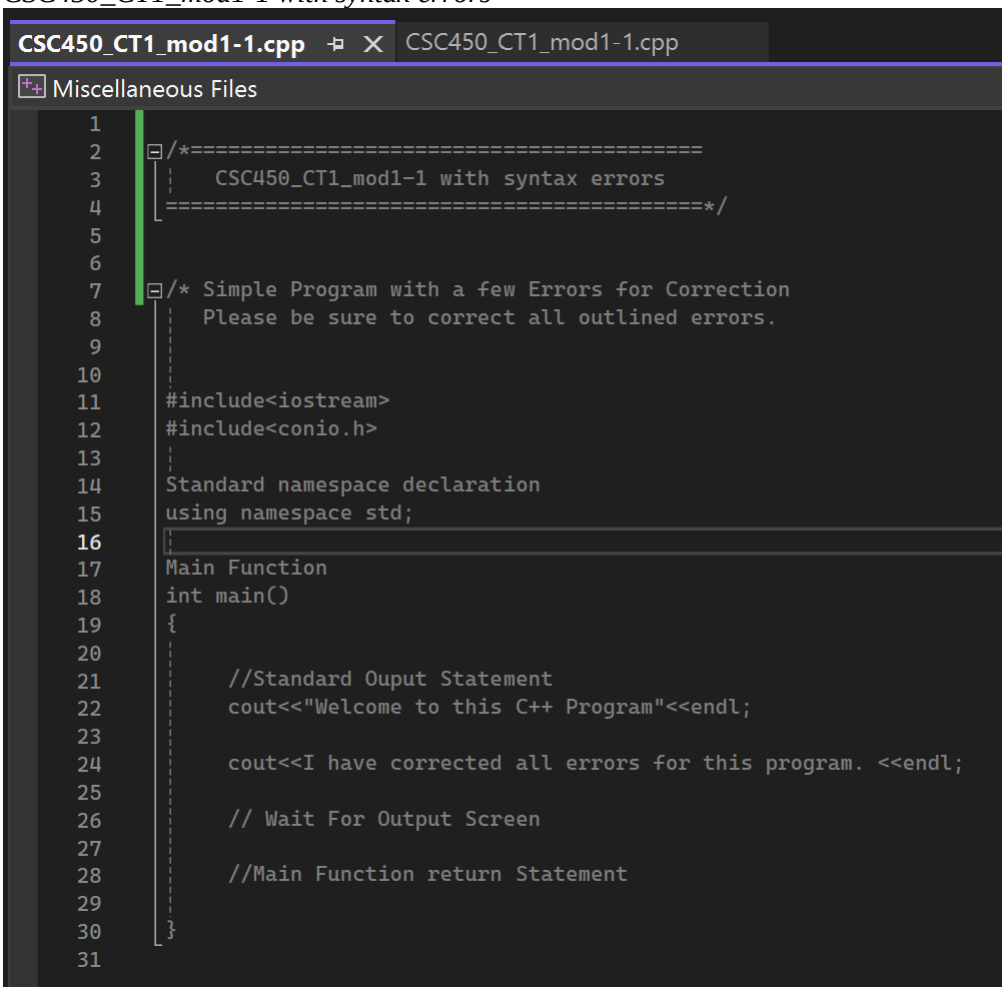
Note: The output of the program showcases a series of tests performed to test the program functionality against the Buffer overflows, Integer overflows, Incorrect type conversions, and Null pointer dereferencing vulgarities. It also displays person data.

Continue next page

Corrected Code Syntax CSC450_CT1_mod1-1:

In this part of the assignment showcases how I corrected code syntaxes from the given C++ program, CSC450_CT1_mod1-1.cpp.

Figure 3
CSC450_CT1_mod1-1 with syntax errors

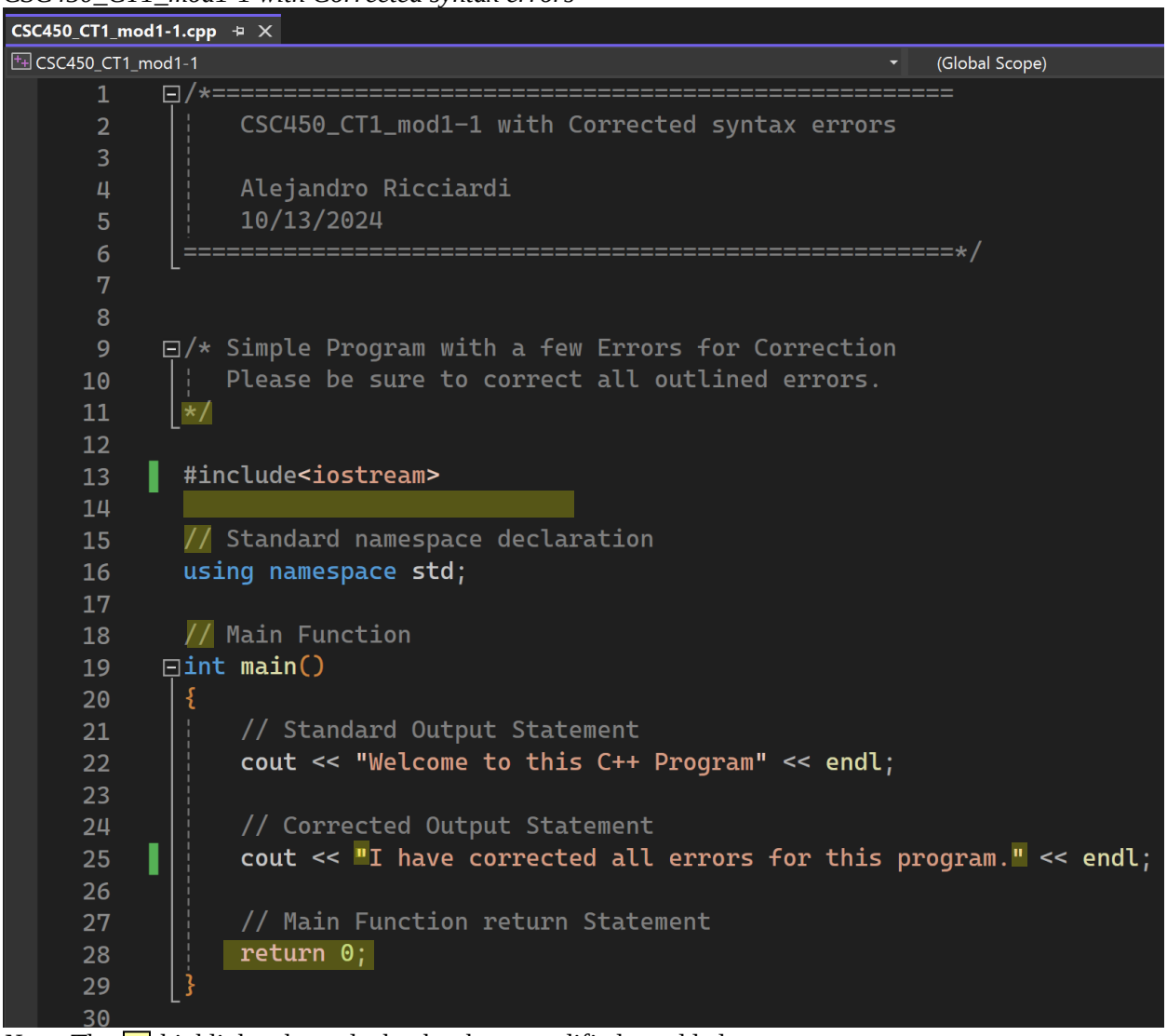


Note: All the code is commented out, this program runs without error. However, it does not output or compute anything as the compiler ignores all the code lines treating it as comments.

Continue next page

Figure 4

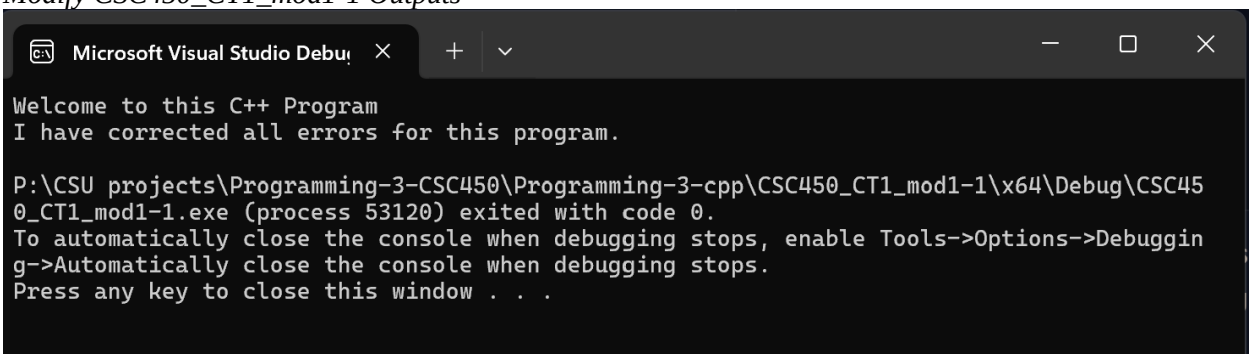
CSC450_CT1_mod1-1 with Corrected syntax errors



Note: The ☐ highlights the code that has been modified or added.

Figure 5

Modify CSC450_CT1_mod1-1 Outputs



I modified:

- Lines 1-6, added header comment.
- Line 14, removed the '#include<conio.h>', which is outdated and unnecessary.
- Line 15, added missing '//' in to comment line.
- Line 18, added space after '//' to comment line.
- Line 26, removed comment line '// Wait For Output Screen".
- Line 28. added 'return 0;'

Corrected Code Syntax CSC450_CT1_mod1-2:

In this part of the assignment showcases how I corrected code syntaxes from the given C++ program, CSC450_CT1_mod1-2.cpp.

Figure 6

CSC450_CT1_mod1-2 with syntax errors

CSC450_CT1_mod1-2.cpp

```
1  /*=====================================================
2     CSC450_CT1_mod1-2 with syntax errors
3  =====================================================*/
4
5
6  #include<iostream>
7  #include<conio.h>
8
9   Standard namespace declaration
10 using namespace std;
11
12  Main Function
13 int main()
14 {
15
16      double myMoney = 1000.50;  //this should be printed out
17
18
19      Standard Ouput Statement
20      cout"Please be sure to correct all syntax errors in this program"<<endl;
21
22      cout<<"I have corrected all errors for this program. <<endl;
23
24      cout<<" The total amount of money available is = "<<   <<endl;
25
26      // Wait For Output Screen
27
28      Main Function return Statement
29      return 0;
30 }
31
```

Note: some of the code lines are flagged by IDE as errors (red squeaky lines). This program generates compiling errors.

Figure 7

CSC450_CT1_mod1-2 with Corrected syntax errors

CSC450_CT1_mod1-2.cpp*

CSC450_CT1_mod1-2.cpp (Global Scope)

```cpp
/*=====================================================
    CSC450_CT1_mod1-2 with Corrected syntax errors

    Alejandro Ricciardi
    10/13/2024
=====================================================*/

#include<iostream>
#include<iomanip>  // setprecision

// Standard namespace declaration
using namespace std;

// Main Function
int main()
{

    double myMoney = 1000.50;  //this should be printed out

    // Standard Output Statement
    cout << "Please be sure to correct all syntax errors in this program" << endl;

    cout << "I have corrected all errors for this program." << endl;

    // Set precision to 2 decimal places
    cout << fixed << setprecision(2);

    cout << "The total amount of money available is = " << myMoney << endl;

    // Main Function return Statement
    return 0;
}
```

Note: The ☐ highlights the code that has been modified, removed, or added.

Figure 8

Modify CSC450_CT1_mod1-2 Outputs

Microsoft Visual Studio Debu

```
Please be sure to correct all syntax errors in this program
I have corrected all errors for this program.
The total amount of money available is = 1000.50

P:\CSU projects\Programming-3-CSC450\Programming-3-cpp\CSC450_CT1_mod1-2\x64\Debug\CSC45
0_CT1_mod1-2.cpp.exe (process 36448) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debuggin
g->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

I modified:

- Lines 1-6, added header comment.
- Line 9, replacd the '#include<conio.h>', which is outdated and unnecessary, with '#include<iomanip> // setprecision'.
- Line 11, added missing '//' in to comment line.
- Line 18, added space after '//' to comment line.
- Line 20, added missing '//' to comment line.
- Line 21, added missing '<<' to the character output line.
- Line 23. added missing ' " ' to the string literal.
- Lines 25-26, added '// Set precision to 2 decimal places' and 'cout << fixed << setprecision(2);'.
- Line 28, added 'myMoney' variable to the character output line.
- Line 29, removed comment line '// Wait For Output Screen".
- Line 30, added missing '//' to comment line.

As shown in Figures 1 through 8 the programs run without any issues displaying the correct outputs as expected.