

Documentation: Critical Thinking 1

Alejandro Ricciardi

Colorado State University Global

CSC450: Programming III

Professor: Reginald Haseltine

October 13, 2024

Contents

The Assignment Direction:.....	3
Git Repository	4
The Simple C++ Console Application Description:	5
Corrected Code Syntax CSC450_CT1_mod1-1:.....	16
Corrected Code Syntax CSC450_CT1_mod1-2:.....	18

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:
 - o **CSC450_CT1_mod1-1.cpp**
 - o **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:

Files

- main
- ...
- CSC450_CT1_mod1-1.cpp
- CSC450_CT1_mod1-2.cpp
- CT1-Person.cpp
- Documentation.pdf
- README.md

Commits

Name	Last commit message	Last commit date
...	Adds programs cpp files and Documentation.pdf to the My Academics Port...	8/25/2024 · 1 minute ago
CSC450_CT1_mod1-1.cpp	Adds programs cpp files and Documentation.pdf to the My Academics Port...	1 minute ago
CSC450_CT1_mod1-2.cpp	Adds programs cpp files and Documentation.pdf to the My Academics Port...	1 minute ago
CT1-Person.cpp	Adds programs cpp files and Documentation.pdf to the My Academics Port...	1 minute ago
Documentation.pdf	Adds programs cpp files and Documentation.pdf to the My Academics Port...	1 minute ago
README.md	Update README.md	8 hours ago

README.md

Critical Thinking 1

Programs Names:
Secure Person Management System – CT1-Person.cpp
Syntax Corrected CSC450_CT1_mod1-1 - CSC450_CT1_mod1-1.cpp
Syntax Corrected CSC450_CT1_mod1-2 - CSC450_CT1_mod1-2.cpp

Grade:

CSC450 – Programming III – C++/Java Course
Professor: Reginald Haseltine Fall D Semester (24FD) – 2024

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 overflows 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 overflows:

```
// 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 overflows 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);

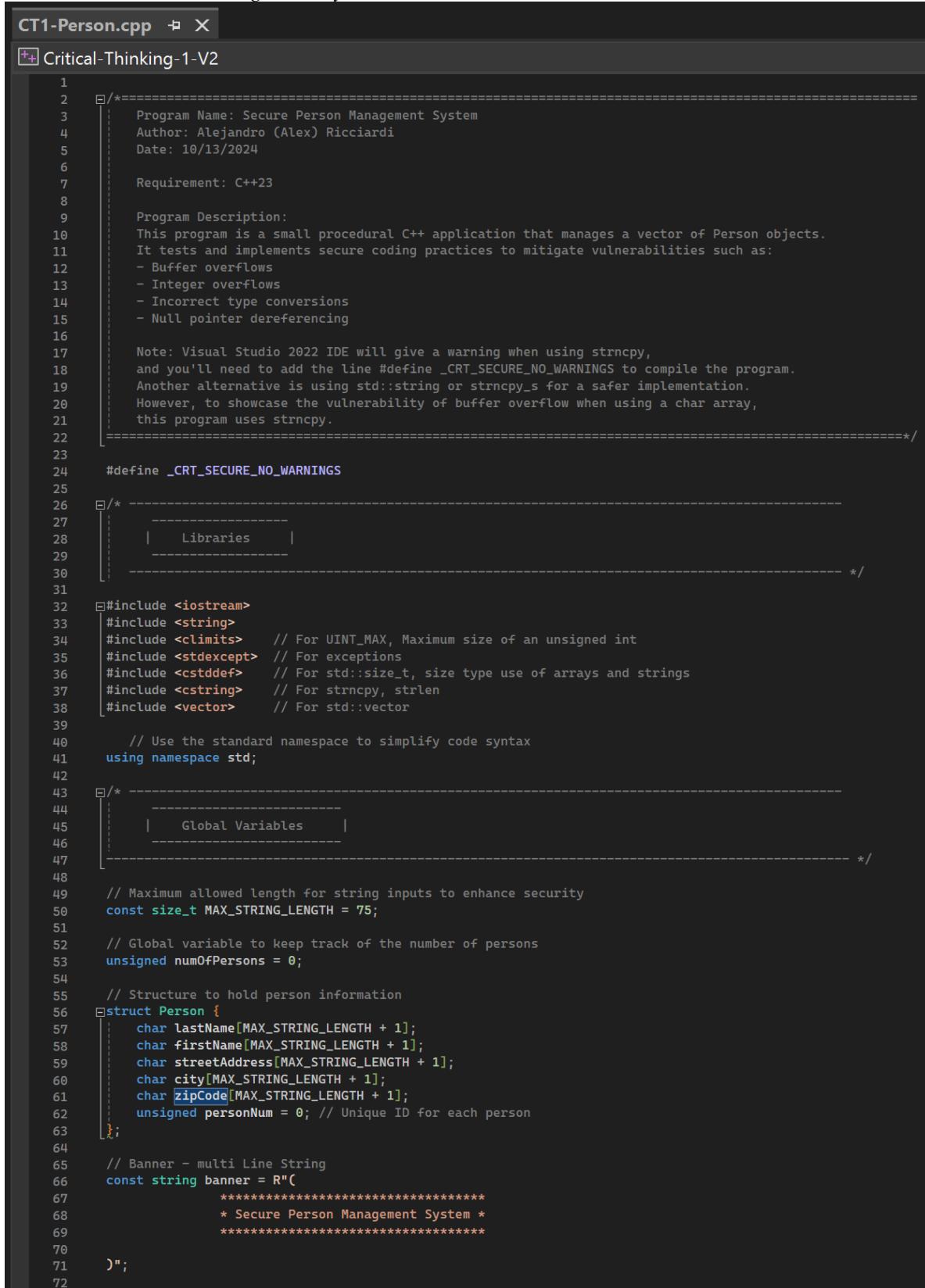
    // Assign a unique person number
    person.personNum = numOfPersons;
    cout << "\nA person with number id: " << person.personNum << " was created successfully!"
        << endl;

    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  ✘ X
Critical-Thinking-1-V2

1  /*
2   *=====
3   | Program Name: Secure Person Management System
4   | Author: Alejandro (Alex) Ricciardi
5   | Date: 10/13/2024
6   |
7   | Requirement: C++23
8   |
9   | Program Description:
10  | This program is a small procedural C++ application that manages a vector of Person objects.
11  | It tests and implements secure coding practices to mitigate vulnerabilities such as:
12  | - Buffer overflows
13  | - Integer overflows
14  | - Incorrect type conversions
15  | - Null pointer dereferencing
16  |
17  | Note: Visual Studio 2022 IDE will give a warning when using strncpy,
18  | and you'll need to add the line #define _CRT_SECURE_NO_WARNINGS to compile the program.
19  | Another alternative is using std::string or strncpy_s for a safer implementation.
20  | However, to showcase the vulnerability of buffer overflow when using a char array,
21  | this program uses strncpy.
22  */
23
24 #define _CRT_SECURE_NO_WARNINGS
25
26 /**
27 |-----|
28 | Libraries      |
29 |-----|
30 ----- */
31
32 #include <iostream>
33 #include <string>
34 #include <climits>    // For UINT_MAX, Maximum size of an unsigned int
35 #include <stdexcept> // For exceptions
36 #include <cstddef>   // For std::size_t, size type use of arrays and strings
37 #include <cstring>    // For strncpy, strlen
38 #include <vector>     // For std::vector
39
40 // Use the standard namespace to simplify code syntax
41 using namespace std;
42
43 /**
44 |-----|
45 | Global Variables      |
46 |-----|
47 ----- */
48
49 // Maximum allowed length for string inputs to enhance security
50 const size_t MAX_STRING_LENGTH = 75;
51
52 // Global variable to keep track of the number of persons
53 unsigned numOfPersons = 0;
54
55 // Structure to hold person information
56 struct Person {
57     char lastName[MAX_STRING_LENGTH + 1];
58     char firstName[MAX_STRING_LENGTH + 1];
59     char streetAddress[MAX_STRING_LENGTH + 1];
60     char city[MAX_STRING_LENGTH + 1];
61     char zipCode[MAX_STRING_LENGTH + 1];
62     unsigned personNum = 0; // Unique ID for each person
63 };
64
65 // Banner - multi Line String
66 const string banner = R"(
67     ****
68     * Secure Person Management System *
69     ****
70 )";
71
72

```

```
73 // =====
74 /* -----
75 |   Function Declaration   |
76 |----- */
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);
```

Continue next page

```

117 // =====
118 /* -----
119 |   Main Function   |
120 -----
121
122 tests secure coding practices such as:
123 - Buffer overflows
124 - Integer overflows
125 - Incorrect type conversions
126 - Null pointer dereferencing
127 -----
128 // -----
129 int main() {
130     /* -----
131     |   Variables   |
132     -----
133
134 // Variables used to test code vulnerability
135 int negativeValue = -5;           // Used for unsigned int vulnerability
136 string longString(100, 'A'); // Create a string with 100 'A's for string overflow
137 Person* personPtr = nullptr; // Initialize pointer to null for null pointer vulnerability
138
139 // Vector of persons to store person struct objects
140 vector<Person> persons;
141
142 /* -----
143     |   Program   |
144     -----
145
146 cout << banner << endl; // std::endl forces flushing of the buffer - good practice
147
148 // ----- Test 1: Buffer Overflow with Overly Long Strings
149
150 cout << "-----\n"
151     << "Test 1: Buffer Overflow with Overly Long Strings\n"
152     << "Creates a person with first and last names that are 100 characters long, filled with the letter 'A'.\n"
153     << endl;
154 // Not expected to fail due to safe string handling
155 Person person1;
156 if (!createPerson(person1, longString, longString)) {
157     cout << "\n Error --- Failed to create person ---" << endl;
158 }
159 persons.push_back(person1);
160 displayPersons(persons);
161
162 // ----- Test 2: Integer Overflow when Creating Many Persons
163 cout << "\n-----\n"
164     << "Test 2: Integer Overflow when Creating Too Many Persons\n"
165     << "Simulate numOfPersons = UINT_MAX, Maximum size of an unsigned int\n"
166     << "\nTrying to create a new person"
167     << endl;
168
169 numOfPersons = UINT_MAX;
170 Person personFail
171 ;
172 if (!createPerson(personFail, "Doe", "John")) {
173     // Expected to fail due to integer overflow security
174     cout << "Failed to create person due to integer overflow." << endl;
175 }
176
177 // Reset numOfPersons for further tests
178 numOfPersons = static_cast<unsigned>(persons.size());
179
180
181
182

```

Continue next page

```

183 // -----
184 cout << "\n-----\n"
185     << "Test 3: Incorrect Type Conversion" << endl;
186
187 // new person
188 Person person2;
189 if (!createPerson(person2, "Conversion", "Alexandria")) {
190     cout << "\n Error --- Failed to create person ---" << endl;
191 }
192 persons.push_back(person2);
193 displayAPerson(persons, 1); // Index 1
194
195 cout << "\nTrying to assign persons[1].personNum = " << negativeValue
196     << ", which is a negative value\n" << endl;
197
198 // Security measure: Check if personNum is valid
199 // Note that comparator '<' compares the value after the cast
200 if (static_cast<int>(negativeValue) < 0) {
201     cerr << "---- Failed to assign new person number ---\n"
202         << "\nIncorrect type conversion --- Negative value assigned to personNum.\n"
203         << "The value: " << negativeValue << " will cast as a person number: "
204         << static_cast<unsigned>(negativeValue)
205         << endl;
206 }
207 else {
208     persons[1].personNum = static_cast<unsigned>(negativeValue);
209     cout << "persons[1] number number is: " << persons[1].personNum << "\n"
210         << endl;
211 }
212
213 displayAPerson(persons, 1);
214
215 // -----
216 cout << "\n-----\n"
217     << "Test 4: Testing Null Pointer before use - null pointer dereferencing\n"
218     << "Checking if personPtr is null, and it is."
219     << "But if it was not, displaying person first name.."
220     << endl;
221
222 // Attempt to access member of null pointer
223 if (personPtr == nullptr) {
224     cerr << "\nPerson pointer is null! Cannot use!.." << endl;
225 }
226 else {
227     cout << "\nPerson first name: " << personPtr->firstName << endl;
228 }
229
230 // -----
231 cout << "\n-----\n"
232     << "\nAdditional Test: Adding and Displaying Persons" << endl;
233
234 // new persons
235 Person person3;
236 if (!createPerson(person3, "More", "Bob")) {
237     cout << "\n Error --- Failed to create person ---" << endl;
238 }
239 persons.push_back(person3);
240
241 Person person4;
242 if (!createPersonFull(person4, "Marquez", "Anita", "456 Ai Street", "Robot Town", "77442")) {
243     cout << "\n Error --- Failed to create person ---" << endl;
244 }
245 persons.push_back(person4);
246
247 Person person5;
248 if (!createPersonFull(person5, "Wan", "Lu", "777 LLM Street", "AI Town", "77772")) {
249     cout << "\n Error --- Failed to create person ---" << endl;
250 }
251 persons.push_back(person5);
252
253 displayPersons(persons);
254
255 return 0;
256 }
```

```

257 // -----
258 /* -----
259 |   Function Definition   |
260 ----- */
261 // -----
262
263 /**
264  * Increments numOfPersons and check for integer overflow
265
266  * Returns true if the incrementation is successful, false if UINT_MAX is reached
267  *     (UINT_MAX, Maximum size of an unsigned int)
268
269  * -----
270  * @param [in,out] counter - The counter to be incremented
271  * @return true if successful, false otherwise
272 */
273 bool static incrementNumOfPersons(unsigned& counter) {
274     // Check if counter has reached the maximum value for unsigned int
275     if (counter == UINT_MAX) {
276         // Security measure: Prevent integer overflow by checking maximum value
277         cerr << "\nError --- Maximum number of persons reached!" << endl;
278         return false;
279     }
280     // Increment the counter safely
281     ++counter;
282     return true;
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

```

```

333  /* -----
334      Creates a new Person with all arguments inputted
335      Uses safe string handling to prevent buffer overflows
336
337      Returns true if person created successfully, false otherwise
338  ----- */
339  bool static createPersonFull(Person& person, const string& lastNameInput, const string& firstNameInput,
340                             const string& streetAddressInput, const string& cityInput,
341                             const string& zipCodeInput) {
342
343  // Increment the global numOfPersons counter if numOfPersons is not reached
344  if (!incrementNumOfPersons(numOfPersons)) {
345      return false;
346
347
348  // lastNameInput to person.lastName
349  strncpy(person.lastName, lastNameInput.c_str(), MAX_STRING_LENGTH); // truncates if necessary
350  // C and C++ do not automatically know strings length. Instead,
351  // at the end of a string is marked by the null character '\0'
352  person.lastName[MAX_STRING_LENGTH] = '\0';
353
354  if (lastNameInput.length() > MAX_STRING_LENGTH) {
355      cerr << "Warning --- Input string for lastName exceeded maximum length of "
356          << MAX_STRING_LENGTH << " characters and has been truncated." << endl;
357
358
359  // firstNameInput to person.firstName
360  strncpy(person.firstName, firstNameInput.c_str(), MAX_STRING_LENGTH); // truncates if necessary
361  person.firstName[MAX_STRING_LENGTH] = '\0'; // Ensure null termination
362
363  if (firstNameInput.length() > MAX_STRING_LENGTH) {
364      cerr << "Warning --- Input string for firstName exceeded maximum length of "
365          << MAX_STRING_LENGTH << " characters and has been truncated." << endl;
366
367
368  // streetAddressInput to person.streetAddress
369  strncpy(person.streetAddress, streetAddressInput.c_str(), MAX_STRING_LENGTH); // truncates if necessary
370  person.streetAddress[MAX_STRING_LENGTH] = '\0'; // Ensure null termination
371
372  if (streetAddressInput.length() > MAX_STRING_LENGTH) {
373      cerr << "Warning --- Input string for streetAddress exceeded maximum length of "
374          << MAX_STRING_LENGTH << " characters and has been truncated." << endl;
375
376
377  // cityInput to person.city
378  strncpy(person.city, cityInput.c_str(), MAX_STRING_LENGTH); // truncates if necessary
379  person.city[MAX_STRING_LENGTH] = '\0'; // Ensure null termination
380
381  if (cityInput.length() > MAX_STRING_LENGTH) {
382      cerr << "Warning --- Input string for city exceeded maximum length of "
383          << MAX_STRING_LENGTH << " characters and has been truncated." << endl;
384
385
386  // zipCodeInput to person.zipCode
387  strncpy(person.zipCode, zipCodeInput.c_str(), MAX_STRING_LENGTH); // truncates if necessary
388  person.zipCode[MAX_STRING_LENGTH] = '\0'; // Ensure null termination
389
390  if (zipCodeInput.length() > MAX_STRING_LENGTH) {
391      cerr << "Warning --- Input string for zipCode exceeded maximum length of "
392          << MAX_STRING_LENGTH << " characters and has been truncated." << endl;
393
394
395  // Assign a unique person number
396  person.personNum = numOfPersons;
397  cout << "\nA person with number id: " << person.personNum << " was created successfully!" << endl;
398  return true;
399
400
401  -----
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
779

```

Continue next page

```
398  /* -----
399  | Displays the contents of the persons vector
400  ----- */
401  void static displayPersons(const vector<Person>& persons) {
402      cout << "Persons List (Total persons created: " << numOfPersons << ")" << endl;
403      for (size_t i = 0; i < persons.size(); ++i) {
404          cout << "Person " << i + 1 << ": "
405          << persons[i].personNum << " "
406          << persons[i].firstName << " "
407          << persons[i].lastName << ", "
408          << persons[i].streetAddress << ", "
409          << persons[i].city << ", "
410          << persons[i].zipCode << endl;
411      }
412  }
413
414  // -----
415
416  /* -----
417  | Displays a person's data at the given index in the persons vector
418  ----- */
419  void static displayAPerson(const vector<Person>& persons, size_t index) {
420      if (index < persons.size()) {
421          cout << "Persons List (Total persons created: " << numOfPersons << ")" << endl;
422          cout << "Person " << index + 1 << ": "
423          << persons[index].personNum << " "
424          << persons[index].firstName << " "
425          << persons[index].lastName << ", "
426          << persons[index].streetAddress << ", "
427          << persons[index].city << ", "
428          << persons[index].zipCode << endl;
429      }
430      else {
431          cout << "Index out of range" << endl;
432      }
433  }
434
```

Continue next page

Figure 2
Output Code Secure Person Management System

```

Microsoft Visual Studio Debug X + v

*****
* 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 AAAAAAAAAAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAA, 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 perao 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 AAAAAAAAAAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAA, 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.

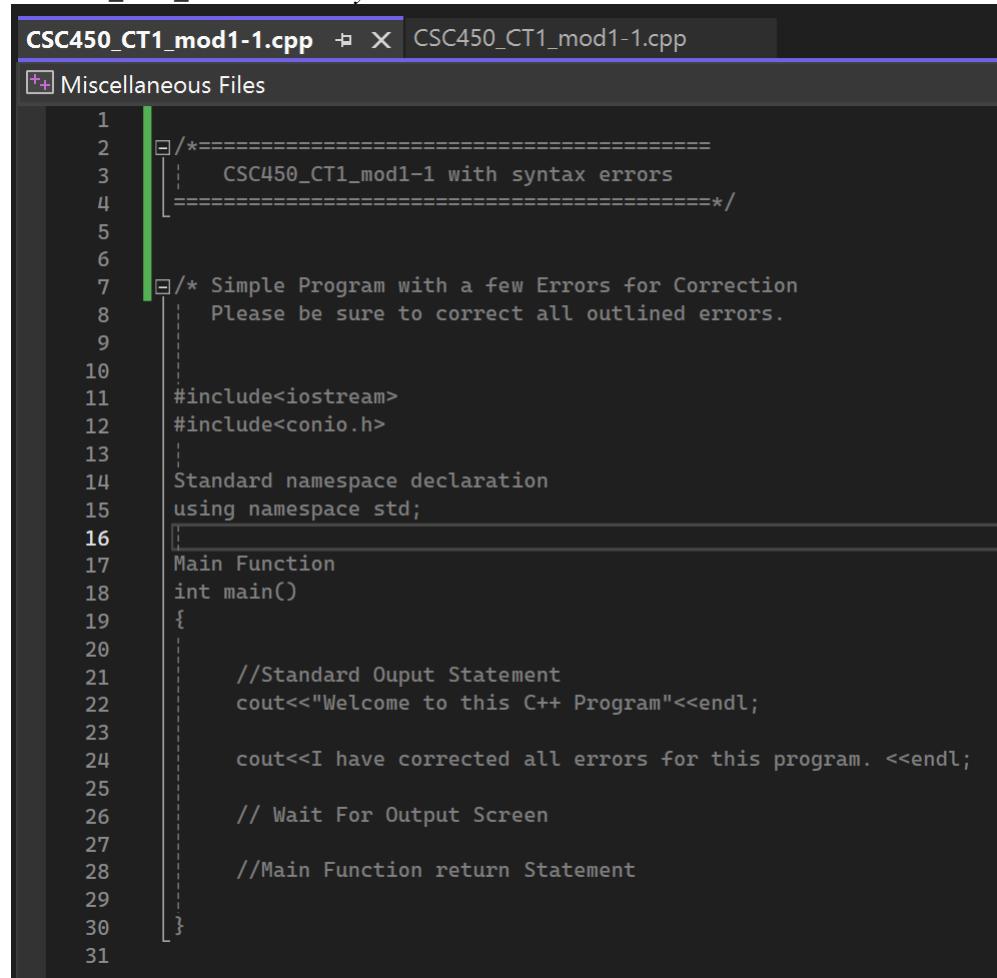
Continue next page

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



```

CSC450_CT1_mod1-1.cpp  X  CSC450_CT1_mod1-1.cpp

[+] Miscellaneous Files

1  /*
2  |=====
3  |     CSC450_CT1_mod1-1 with syntax errors
4  |=====
5
6
7  /* Simple Program with a few Errors for Correction
8  | Please be sure to correct all outlined errors.
9
10
11 #include<iostream>
12 #include<conio.h>
13
14 Standard namespace declaration
15 using namespace std;
16
17 Main Function
18 int main()
19 {
20
21     //Standard Ouput Statement
22     cout<<"Welcome to this C++ Program" << endl;
23
24     cout<<I have corrected all errors for this program. << endl;
25
26     // Wait For Output Screen
27
28     //Main Function return Statement
29
30 }
31

```

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*

```

CSC450_CT1_mod1-1.cpp  ✘ X
CSC450_CT1_mod1-1
=====
1  /*=====
2   | CSC450_CT1_mod1-1 with Corrected syntax errors
3   |
4   | Alejandro Ricciardi
5   | 10/13/2024
6   =====*/
7
8
9  /* Simple Program with a few Errors for Correction
10  | Please be sure to correct all outlined errors.
11  */
12
13 #include<iostream>
14
15 // Standard namespace declaration
16 using namespace std;
17
18 // Main Function
19 int main()
20 {
21     // Standard Output Statement
22     cout << "Welcome to this C++ Program" << endl;
23
24     // Corrected Output Statement
25     cout << "I have corrected all errors for this program." << endl;
26
27     // Main Function return Statement
28     return 0;
29 }
30

```

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

Figure 5*Modify CSC450_CT1_mod1-1 Outputs*

```

Microsoft Visual Studio Debug  +  -
Welcome to this C++ Program
I have corrected all errors for this program.

P:\CSU projects\Programming-3-CSC450\Programming-3-cpp\CSC450_CT1_mod1-1\x64\Debug\CSC450_CT1_mod1-1.exe (process 53120) 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 . . .

```

I modified:

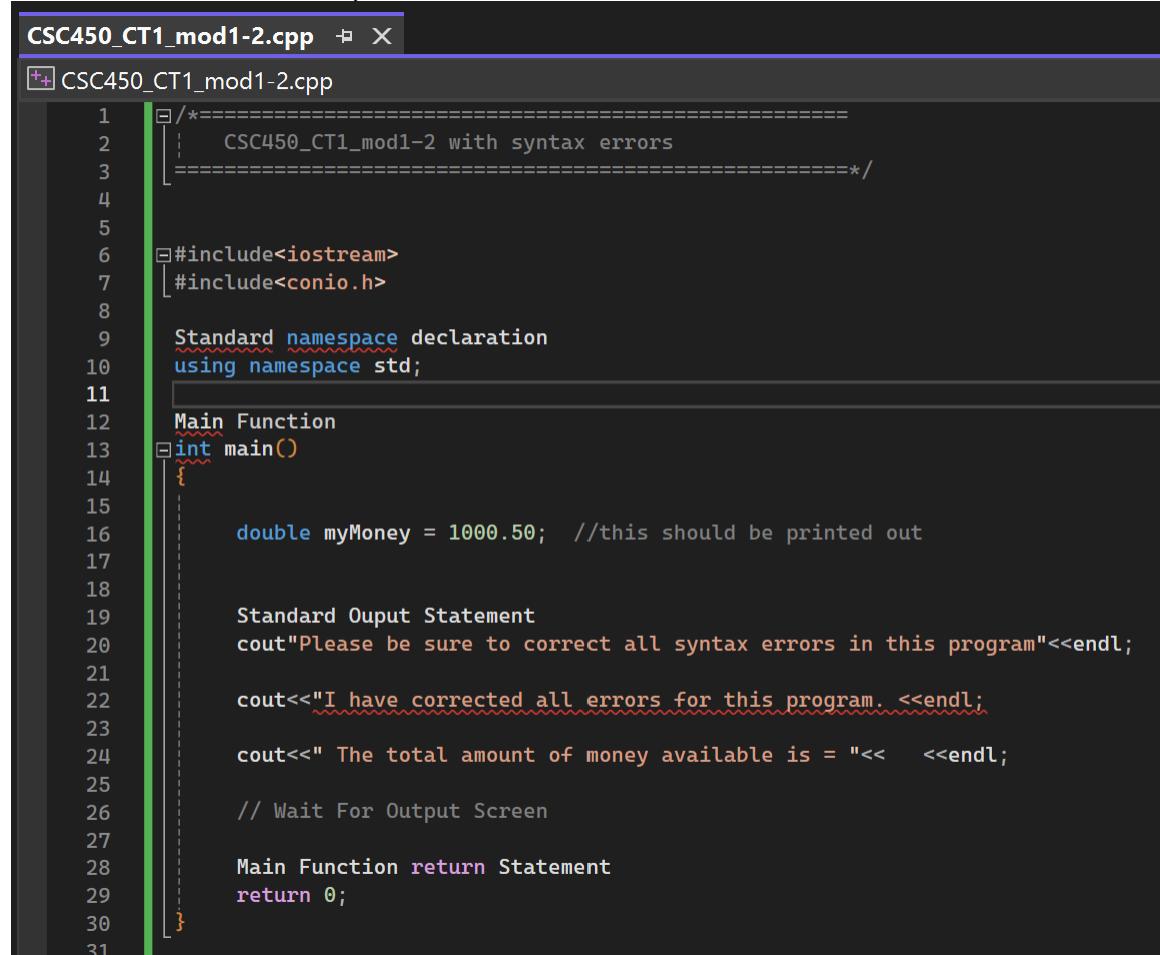
- Lines 1-6, added header comment.
- Line 14, removed the '#include<conio.h>', which is outdated and unnecessary.
- Line 15, added missing '//' in to comment line.
- Line 18, added space after '//' to comment line.
- Line 26, removed comment line '// Wait For Output Screen".
- Line 28. added 'return 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 ✎ X
+ 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 Output 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*  X  (Global Scope)
CSC450_CT1_mod1-2.cpp

1  /*=====
2   CSC450_CT1_mod1-2 with Corrected syntax errors
3
4   Alejandro Ricciardi
5   10/13/2024
6  =====*/
7
8  #include<iostream>
9  #include<iomanip> // setprecision
10
11 // Standard namespace declaration
12 using namespace std;
13
14 // Main Function
15 int main()
16 {
17
18     double myMoney = 1000.50; //this should be printed out
19
20     // Standard Output Statement
21     cout << "Please be sure to correct all syntax errors in this program" << endl;
22
23     cout << "I have corrected all errors for this program." << endl;
24
25     // Set precision to 2 decimal places
26     cout << fixed << setprecision(2);
27
28     cout << "The total amount of money available is = " << myMoney << endl;
29
30     // Main Function return Statement
31     return 0;
32 }
33

```

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

Figure 8*Modify CSC450_CT1_mod1-2 Outputs*

```

Microsoft Visual Studio Debug  X  +  V  -  □  ×
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\CSC450_CT1_mod1-2.cpp.exe (process 36448) 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 . . .

```

I modified:

- Lines 1-6, added header comment.
- Line 9, replaced 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.