

Documentation: Critical Thinking 3

Alejandro Ricciardi

Colorado State University Global

CSC450: Programming III

Professor: Reginald Haseltine

October 27, 2024

Table of Contents

The Assignment Direction:	3
Program Description:	4
Git Repository	4
Source Code in IDE	11
Output Screenshots	17

Documentation: Critical Thinking 3

This documentation is part of the Critical Thinking 3 Assignment from CSC450: Programming III at Colorado State University Global. The program's name is Integer Pointers. It provides an overview of the program's functionality and testing scenarios, including console output screenshots. The program is coded in C++ 23.

The Assignment Direction:

Integer Pointers Program

Demonstrate an understanding of basic C++ programming concepts by completing the following:

- Program: Create a C++ program that asks the user to enter three integer values as input. Store the values into three different variables. For each variable, create an integer pointer to dynamic memory. Display the contents of the variables and pointers. In your program, be sure to use the new operator and delete operators to manage memory.

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.

⚠ My notes:

- The simple C++ console application is in file **CTA-3-integerPointers.cpp**
- It is best practice to utilize smart pointers like '`std::unique_ptr`' or '`std::shared_ptr`' which automatically manage memory. However, to demonstrate the use of the new and delete operators to manage memory as required by the assignment, the program uses regular raw pointers.
- Instead of using '`std::cin`' and '`int`' to capture and store user inputs, the program uses '`std::getline`' and '`std::string`'. This allows the program to have more control and flexibility over input validation. For example, the program allows whitespaces to be entered before and/or after the integer value.
- Integer can be negative. The standard integer size, in C++, is typically 4 bytes and is platform-dependent.
- The program follows the following SEI CERT C/C++ Coding Standard:
 - EXP34-C. Do not dereference null pointers
 - EXP53-CPP. Do not read uninitialized memory
 - ERR50-CPP. Do not abruptly terminate the program
 - ERR51-CPP. Handle all exceptions
 - ERR56-CPP. Guarantee exception safety
 - ERR57-CPP. Do not leak resources when handling exceptions
 - MEM50-CPP. Do not access freed memory
 - MEM51-CPP. Properly deallocate dynamically allocated resources
 - MEM57-CPP. Avoid using default operator new for over-aligned types
 - INT50-CPP. Do not cast to an out-of-range enumeration value
 - STR50-CPP. Guarantee that storage for strings has sufficient space for character data and the null terminator

Program Description:

The program is a small procedural C++ program that prompts a user to enter three integer values, validates the input values as integers and stores the values using raw pointers.

Note:

- The standard integer is typically 4 bytes, it is platform-dependent.
- The Program accepts whitespaces to be entered before and/or after the integer value.
- The program follows the following SEI CERT C/C++ Coding Standard:
 - EXP34-C. Do not dereference null pointers
 - EXP53-CPP. Do not read uninitialized memory
 - ERR50-CPP. Do not abruptly terminate the program
 - ERR51-CPP. Handle all exceptions
 - ERR56-CPP. Guarantee exception safety
 - ERR57-CPP. Do not leak resources when handling exceptions
 - MEM50-CPP. Do not access freed memory
 - MEM51-CPP. Properly deallocate dynamically allocated resources
 - MEM57-CPP. Avoid using default operator new for over-aligned types
 - INT50-CPP. Do not cast to an out-of-range enumeration value
 - STR50-CPP. Guarantee that storage for strings has sufficient space for character data and the null terminator

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-3>



Image of the source code in the GitHub: see next page

Figure 1

Source Code in GitHub lines 1-54 (Program Header-Libraries-Global Variables)

The screenshot shows a GitHub repository interface. The left sidebar lists various projects and files, with 'CTA-3-integerPointers.cpp' selected. The main area displays the code content.

```

1  /*=====
2   * Program Name: Integer Pointers
3   * Author: Alejandro (Alex) Ricciardi
4   * Date: 10/27/2024
5
6   * Requirement: C++23
7
8   * Program Description:
9   * The program is a small procedural C++ program that prompts a user to enter three integer values,
10  * validates the input values as integers and stores the values using raw pointers.
11
12  * Note:
13  * - The standard integer is typically 4 bytes, it is platform dependent.
14  * - The Program accepts whitespaces to be entered before and/or after the integer value.
15  * - The program follows the following SEI CERT C/C++ Coding Standard:
16  *   - EXP34-C. Do not dereference null pointers
17  *   - EXP53-CPP. Do not read uninitialized memory
18  *   - ERR50-CPP. Do not abruptly terminate the program
19  *   - ERR51-CPP. Handle all exceptions
20  *   - ERR56-CPP. Guarantee exception safety
21  *   - ERR57-CPP. Do not leak resources when handling exceptions
22  *   - MEM50-CPP. Do not access freed memory
23  *   - MEM51-CPP. Properly deallocate dynamically allocated resources
24  *   - MEM57-CPP. Avoid using default operator new for over-aligned types
25  *   - INT50-CPP. Do not cast to an out-of-range enumeration value
26  *   - STR50-CPP. Guarantee that storage for strings has sufficient space
27  *     for character data and the null terminator
28
29 =====*/
30
31  /*
32  -----
33  |   Libraries   |
34  -----
35  -----
36  #include <iostream>
37  #include <limits>      // For INT_MAX and INT_MIN
38  #include <string>
39  #include <stdexcept>    // For exception handling
40  #include <new>         // For std::nothrow, prevents exceptions on memory allocation failure
41
42  using namespace std;
43
44  /*
45  -----
46  |   Global Variables   |
47  -----
48  -----
49  // Banner - multi Line String
50  const string banner = R"(
51  ****
52  * Integer Pointers *
53  ****
54 )";

```

Figure 2*Source Code in GitHub lines 55-93 (Function Declarations)*

The screenshot shows a GitHub repository interface. On the left, the 'Files' sidebar lists various projects and files. In the center, the 'Code' tab is selected for the file 'CTA-3-integerPointers.cpp'. The code itself is as follows:

```

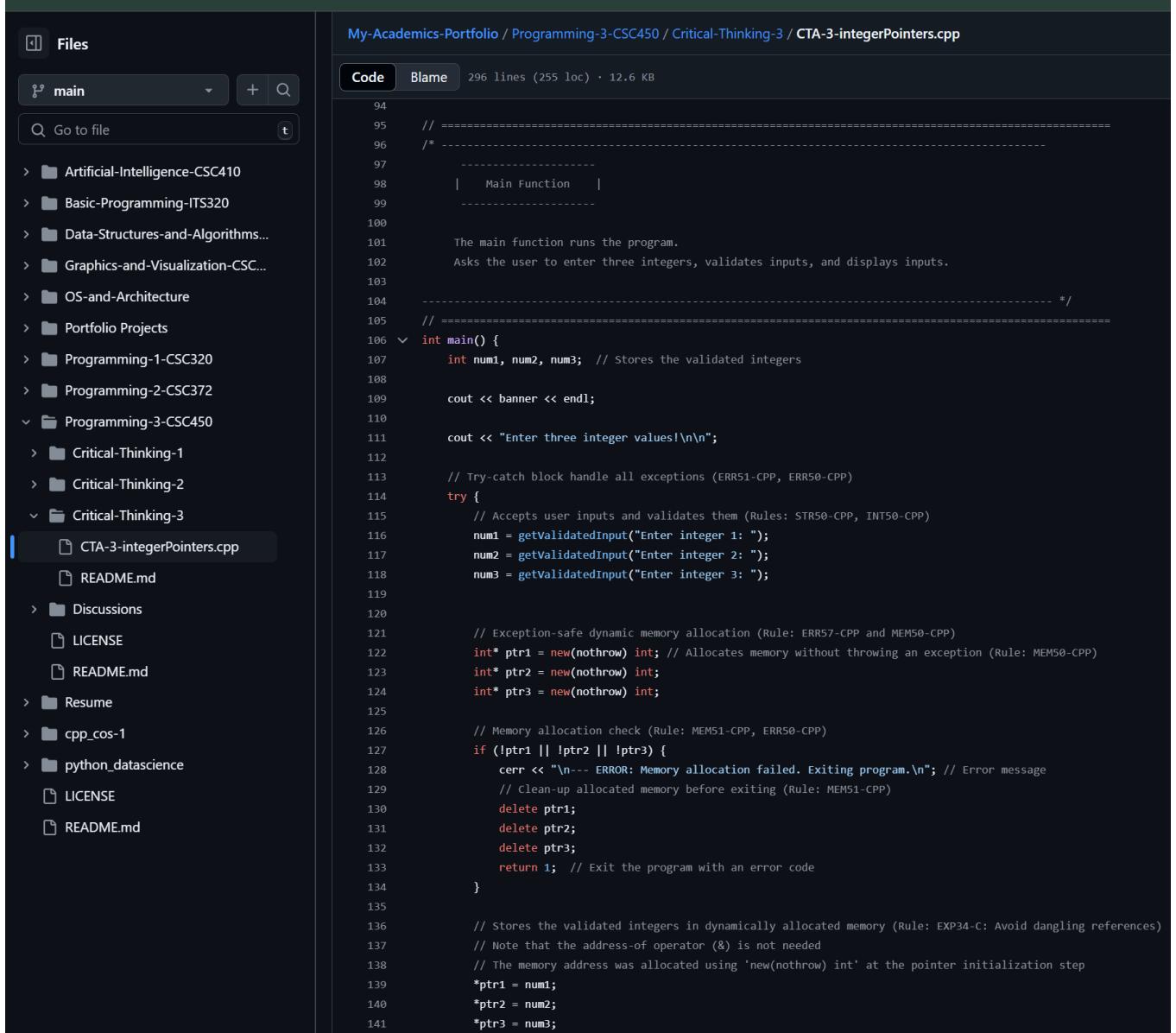
55 // -----
56 // -----
57 /* -----
58 -----| Function Declaration | -----
59 ----- */
60 // -----
61 // -----
62 // -----
63
64 /**
65 * Prompts the user to enter an integer and validates the input
66 * It will prompt the user until a valid integer is entered
67 *
68 * Note:
69 * The standard integer is typically 4 bytes, it is platform dependent
70 * The Program accepts spaces before and after the integer value
71 *
72 * Handles Rules:
73 * - ERR50-CPP. Do not abruptly terminate the program
74 * - ERR51-CPP. Handle all exceptions
75 * - INT50-CPP. Do not cast to an out-of-range enumeration value
76 * - STR50-CPP. Guarantee that storage for strings has sufficient space
77 *   for character data and the null terminator
78 *
79 * @param prompt The message to user
80 * @return validated integer
81 */
82 int getValidatedInput(const string& prompt) noexcept(false);
83
84 /**
85 * Displays the value pointed to by ptr.
86 * checks if the pointer is not null, if it is, it displays an error,
87 * otherwise it display the value pointed (Rule: EXP53-CPP)
88 *
89 * @param name The name of the pointer for display purposes.
90 * @param ptr The pointer to an integer
91 */
92 void displayPointer(const char* name, int* ptr);
93

```

Continue next page

Figure 3

Source Code in GitHub lines 94-141 (Main Function Part-1)



```

My-Academics-Portfolio / Programming-3-CSC450 / Critical-Thinking-3 / CTA-3-integerPointers.cpp

Code Blame 296 lines (255 loc) · 12.6 KB

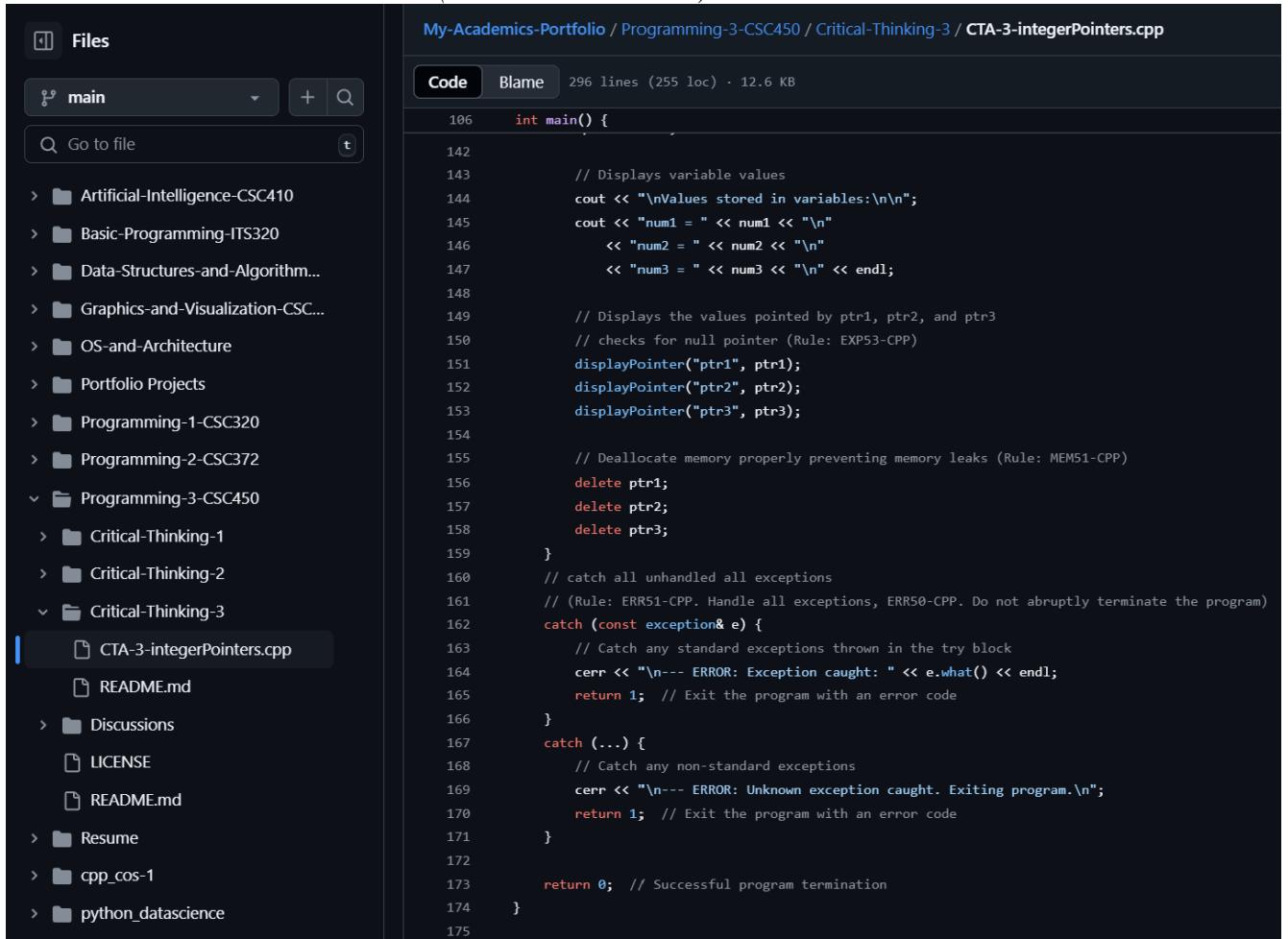
94 // =====
95 /*
96 * -----
97 * -----| Main Function |-----
98 * -----
99 *
100
101     The main function runs the program.
102     Asks the user to enter three integers, validates inputs, and displays inputs.
103
104 // -----
105 // -----
106 int main() {
107     int num1, num2, num3; // Stores the validated integers
108
109     cout << banner << endl;
110
111     cout << "Enter three integer values!\n\n";
112
113     // Try-catch block handle all exceptions (ERR51-CPP, ERR50-CPP)
114     try {
115         // Accepts user inputs and validates them (Rules: STR50-CPP, INT50-CPP)
116         num1 = getValidatedInput("Enter integer 1: ");
117         num2 = getValidatedInput("Enter integer 2: ");
118         num3 = getValidatedInput("Enter integer 3: ");
119
120
121         // Exception-safe dynamic memory allocation (Rule: ERR57-CPP and MEM50-CPP)
122         int* ptr1 = new(nothrow) int; // Allocates memory without throwing an exception (Rule: MEM50-CPP)
123         int* ptr2 = new(nothrow) int;
124         int* ptr3 = new(nothrow) int;
125
126         // Memory allocation check (Rule: MEM51-CPP, ERR50-CPP)
127         if (!ptr1 || !ptr2 || !ptr3) {
128             cerr << "\n--- ERROR: Memory allocation failed. Exiting program.\n"; // Error message
129             // clean-up allocated memory before exiting (Rule: MEM51-CPP)
130             delete ptr1;
131             delete ptr2;
132             delete ptr3;
133             return 1; // Exit the program with an error code
134         }
135
136         // Stores the validated integers in dynamically allocated memory (Rule: EXP34-C: Avoid dangling references)
137         // Note that the address-of operator (&) is not needed
138         // The memory address was allocated using 'new(nothrow) int' at the pointer initialization step
139         *ptr1 = num1;
140         *ptr2 = num2;
141         *ptr3 = num3;

```

Continue next page

Figure 4

Source Code in GitHub lines 142-175 (Main Function Part-2)



The screenshot shows a GitHub repository interface. On the left, there's a sidebar with a 'Files' tab and a search bar. Below it is a tree view of the project structure. A blue highlight box surrounds the file 'CTA-3-integerPointers.cpp'. The main area displays the code for this file, specifically the main function and its body.

```

My-Academics-Portfolio / Programming-3-CSC450 / Critical-Thinking-3 / CTA-3-integerPointers.cpp

Code Blame 296 lines (255 loc) · 12.6 KB

106 int main() {
142     // Displays variable values
143     cout << "\nValues stored in variables:\n\n";
144     cout << "num1 = " << num1 << "\n"
145         << "num2 = " << num2 << "\n"
146         << "num3 = " << num3 << "\n" << endl;
147
148
149     // Displays the values pointed by ptr1, ptr2, and ptr3
150     // checks for null pointer (Rule: EXP53-CPP)
151     displayPointer("ptr1", ptr1);
152     displayPointer("ptr2", ptr2);
153     displayPointer("ptr3", ptr3);
154
155     // Deallocate memory properly preventing memory leaks (Rule: MEM51-CPP)
156     delete ptr1;
157     delete ptr2;
158     delete ptr3;
159 }
160 // catch all unhandled exceptions
161 // (Rule: ERR51-CPP. Handle all exceptions, ERR50-CPP. Do not abruptly terminate the program)
162 catch (const exception& e) {
163     // Catch any standard exceptions thrown in the try block
164     cerr << "\n--- ERROR: Exception caught: " << e.what() << endl;
165     return 1; // Exit the program with an error code
166 }
167 catch (...) {
168     // Catch any non-standard exceptions
169     cerr << "\n--- ERROR: Unknown exception caught. Exiting program.\n";
170     return 1; // Exit the program with an error code
171 }
172
173 return 0; // Successful program termination
174 }
```

Continue next page

Figure 5Source Code in GitHub lines 176-231 (Function Definition Part-1 - `getValidatedInput()`)

```

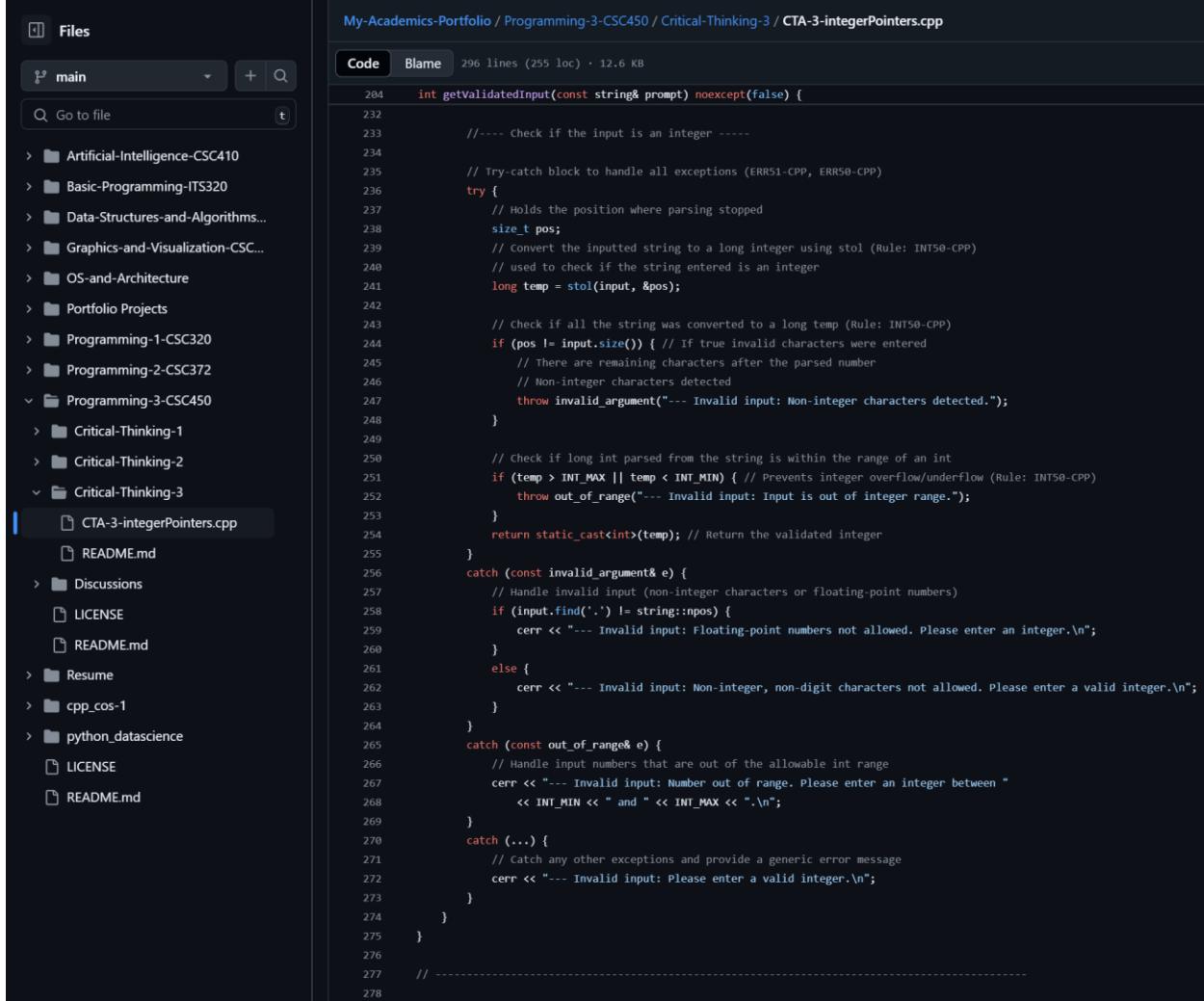
My-Academics-Portfolio / Programming-3-CSC450 / Critical-Thinking-3 / CTA-3-integerPointers.cpp

Code Blame 296 lines (255 loc) · 12.6 KB

176 // -----
177 // =====
178 /* -----
179 |-----|
180 | Function Definitions |
181 |-----|
182 ----- */
183 // -----
184 // -----
185 /**
186 * Prompts the user to enter an integer and validates the input
187 * It will prompt the user until a valid integer is entered
188 *
189 * Note:
190 * The standard integer is typically 4 bytes, it is platform dependent
191 * The Program accepts whitespaces before and after the integer value
192 *
193 * Handles Rules:
194 * - ERR50-CPP. Do not abruptly terminate the program
195 * - ERR51-CPP. Handle all exceptions
196 * - INT50-CPP. Do not cast to an out-of-range enumeration value
197 * - STR50-CPP. Guarantee that storage for strings has sufficient space
198 *   for character data and the null terminator
199 *
200 * @param prompt The message to user
201 * @return validated integer
202 */
203 int getValidatedInput(const string& prompt) noexcept(false) {
204     while (true) {
205         string input;
206
207         cout << prompt;
208
209         // Gets the entire line of input and stores it in a string object
210         // Using std::string is safer for string manipulation (STR50-CPP)
211         getline(cin, input);
212
213         //----- Whitespace Handling -----
214
215         // Remove leading and trailing whitespaces from the input
216         size_t start = input.find_first_not_of(" \t\n\r");
217         size_t end = input.find_last_not_of(" \t\n\r");
218
219         // Check if the input only contains whitespaces
220         // The following will be true if after removing all the whitespaces
221         // and if the input contains only whitespace
222         // 'string::npos' no position meaning that start has no position after parsing
223         if (start == string::npos) {
224             cerr << "---- Invalid input: Please enter an integer value.\n";
225             // skips the rest of the current while loop iteration
226             continue; // Prompt the user again
227         }
228
229         // Trim the input to remove leading and trailing whitespaces
230         input = input.substr(start, end - start + 1);
231

```

Continue next page

Figure 6Source Code in GitHub lines 232-278 (Function Definition Part-2 - `getValidatedInput()`)


```

My-Academics-Portfolio / Programming-3-CSC450 / Critical-Thinking-3 / CTA-3-integerPointers.cpp

Code Blame 296 lines (255 loc) · 12.6 KB
204 int getValidatedInput(const string& prompt) noexcept(false) {
232     //---- Check if the input is an integer ----
234
235     // Try-catch block to handle all exceptions (ERR51-CPP, ERR50-CPP)
236     try {
237         // Holds the position where parsing stopped
238         size_t pos;
239         // convert the inputted string to a long integer using stol (Rule: INT50-CPP)
240         // used to check if the string entered is an integer
241         long temp = stol(input, &pos);
242
243         // Check if all the string was converted to a long temp (Rule: INT50-CPP)
244         if (pos != input.size()) { // If true invalid characters were entered
245             // There are remaining characters after the parsed number
246             // Non-integer characters detected
247             throw invalid_argument("---- Invalid input: Non-integer characters detected.");
248         }
249
250         // check if long int parsed from the string is within the range of an int
251         if (temp > INT_MAX || temp < INT_MIN) { // Prevents integer overflow/underflow (Rule: INT50-CPP)
252             throw out_of_range("---- Invalid input: Input is out of integer range.");
253         }
254         return static_cast<int>(temp); // Return the validated integer
255     }
256     catch (const invalid_argument& e) {
257         // Handle invalid input (non-integer characters or floating-point numbers)
258         if (input.find('.') != string::npos) {
259             cerr << "---- Invalid input: Floating-point numbers not allowed. Please enter an integer.\n";
260         }
261         else {
262             cerr << "---- Invalid input: Non-integer, non-digit characters not allowed. Please enter a valid integer.\n";
263         }
264     }
265     catch (const out_of_range& e) {
266         // Handle input numbers that are out of the allowable int range
267         cerr << "---- Invalid input: Number out of range. Please enter an integer between "
268             << INT_MIN << " and " << INT_MAX << ".\n";
269     }
270     catch (...) {
271         // Catch any other exceptions and provide a generic error message
272         cerr << "---- Invalid input: Please enter a valid integer.\n";
273     }
274 }
275 }
276 // -----
277
278

```

Figure 7Source Code in GitHub lines 279-296 (Function Definition Part-3 - `displayPointer()`)


```

LICENSE
README.md

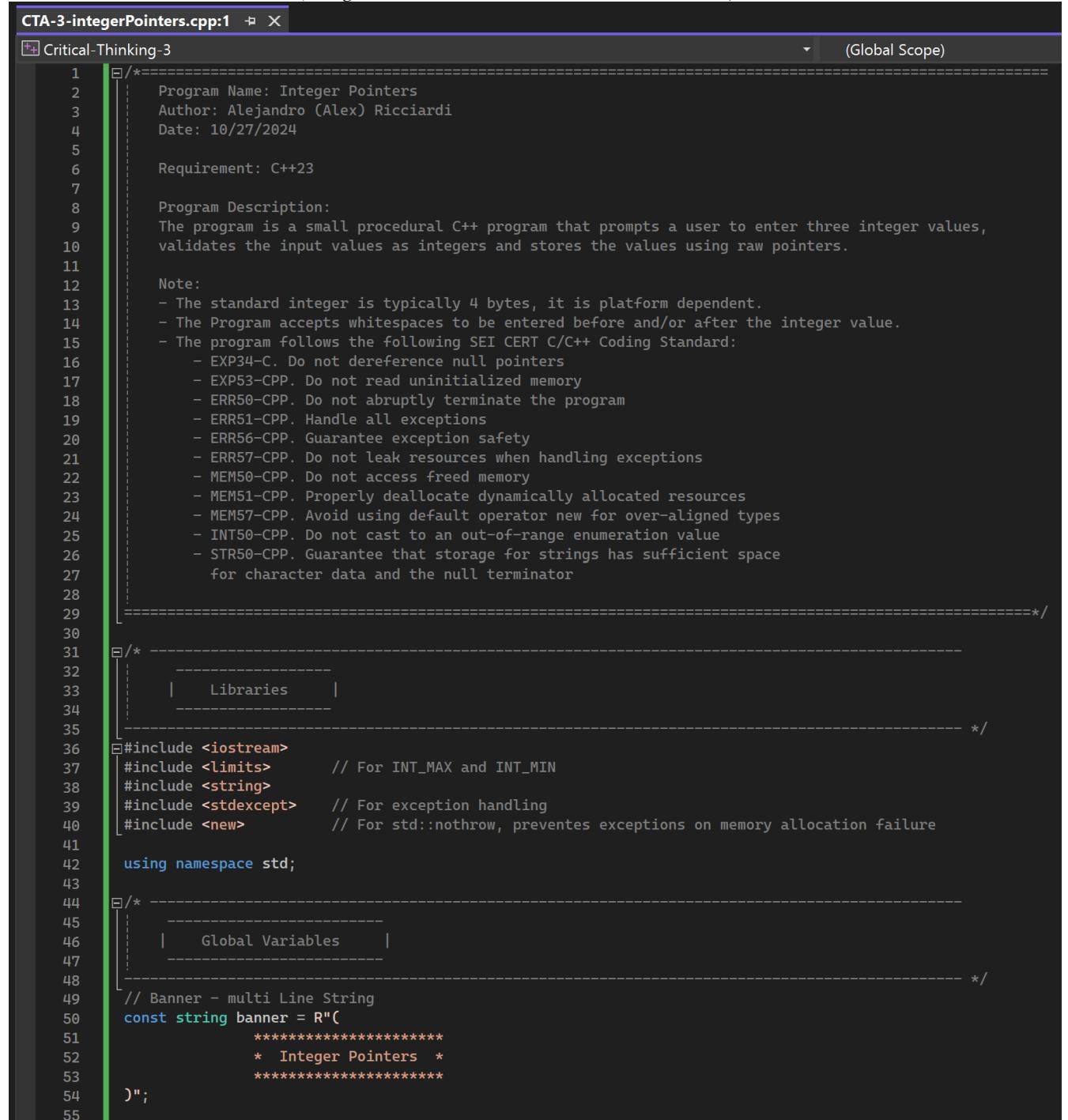
279     /*
280      * Displays the value pointed to by ptr.
281      * checks if the pointer is not null, if it is, it displays an error,
282      * otherwise it display the value pointed (Rule: EXP53-CPP)
283      *
284      * @param name The name of the pointer for display purposes.
285      * @param ptr The pointer to an integer
286      */
287     void displayPointer(const char* name, int* ptr) {
288         if (ptr == nullptr) { // Check if the pointer is null (Rule: EXP53-CPP)
289             cerr << name << "\n--- ERROR: null pointer!\n";
290         }
291         else { // Dereference the pointer and display its value
292             cout << "*" << name << " = " << *ptr << endl;
293         }
294     }
295
296     // -----

```

Source Code in IDE

Figure 8

Source Code in IDE lines 1-54 (Program Header-Libraries-Global Variables)



```

CTA-3-integerPointers.cpp:1 ✘ X
Critical-Thinking-3 (Global Scope)

1  /*=====
2   |   Program Name: Integer Pointers
3   |   Author: Alejandro (Alex) Ricciardi
4   |   Date: 10/27/2024
5   |
6   |   Requirement: C++23
7   |
8   |   Program Description:
9   |   The program is a small procedural C++ program that prompts a user to enter three integer values,
10  |   validates the input values as integers and stores the values using raw pointers.
11  |
12  |   Note:
13  |   - The standard integer is typically 4 bytes, it is platform dependent.
14  |   - The Program accepts whitespaces to be entered before and/or after the integer value.
15  |   - The program follows the following SEI CERT C/C++ Coding Standard:
16  |       - EXP34-C. Do not dereference null pointers
17  |       - EXP53-CPP. Do not read uninitialized memory
18  |       - ERR50-CPP. Do not abruptly terminate the program
19  |       - ERR51-CPP. Handle all exceptions
20  |       - ERR56-CPP. Guarantee exception safety
21  |       - ERR57-CPP. Do not leak resources when handling exceptions
22  |       - MEM50-CPP. Do not access freed memory
23  |       - MEM51-CPP. Properly deallocate dynamically allocated resources
24  |       - MEM57-CPP. Avoid using default operator new for over-aligned types
25  |       - INT50-CPP. Do not cast to an out-of-range enumeration value
26  |       - STR50-CPP. Guarantee that storage for strings has sufficient space
27  |           for character data and the null terminator
28  |
29  =====*/
30
31  /*
32  |   -----
33  |   |   Libraries   |
34  |   -----
35  */
36  #include <iostream>
37  #include <limits>      // For INT_MAX and INT_MIN
38  #include <string>
39  #include <stdexcept>   // For exception handling
40  #include <new>         // For std::nothrow, prevents exceptions on memory allocation failure
41
42  using namespace std;
43
44  /*
45  |   -----
46  |   |   Global Variables   |
47  |   -----
48  */
49  // Banner - multi Line String
50  const string banner = R"(
51  ****
52  * Integer Pointers *
53  ****
54 )";
55

```

Figure 9*Source Code in IDE lines 55-93 (Function Declarations)*

CTA-3-integerPointers.cpp:1 (Global Scope)

```

55 // -----
56 /* -----
57 |-----|
58 |       Function Declaration      |
59 |-----|
60 ----- */
61 // -----
62
63
64 /**
65 * Prompts the user to enter an integer and validates the input
66 * It will prompt the user until a valid integer is entered
67 *
68 * Note:
69 * The standard integer is typically 4 bytes, it is platform dependent
70 * The Program accepts spaces before and after the integer value
71 *
72 * Handles Rules:
73 * - ERR50-CPP. Do not abruptly terminate the program
74 * - ERR51-CPP. Handle all exceptions
75 * - INT50-CPP. Do not cast to an out-of-range enumeration value
76 * - STR50-CPP. Guarantee that storage for strings has sufficient space
77 *   for character data and the null terminator
78 *
79 * @param prompt The message to user
80 * @return validated integer
81 */
82 int getValidatedInput(const string& prompt) noexcept(false);
83
84 /**
85 * Displays the value pointed to by ptr.
86 * checks if the pointer is not null, if it is, it displays an error,
87 * otherwise it display the value pointed (Rule: EXP53-CPP)
88 *
89 * @param name The name of the pointer for display purposes.
90 * @param ptr The pointer to an integer
91 */
92 void displayPointer(const char* name, int* ptr);
93

```

Continue next page

Figure 10*Source Code in IDE lines 94-141 (Main Function Part-1)*

CTA-3-integerPointers.cpp:1

(Global Scope)

```

94 // -----
95 /* -----
96 |   Main Function   |
97 -----
98
99
100
101    The main function runs the program.
102    Asks the user to enter three integers, validates inputs, and displays inputs.
103
104 */
105
106 int main() {
107     int num1, num2, num3; // Stores the validated integers
108
109     cout << banner << endl;
110
111     cout << "Enter three integer values!\n\n";
112
113     // Try-catch block handle all exceptions (ERR51-CPP, ERR50-CPP)
114     try {
115         // Accepts user inputs and validates them (Rules: STR50-CPP, INT50-CPP)
116         num1 = getValidatedInput("Enter integer 1: ");
117         num2 = getValidatedInput("Enter integer 2: ");
118         num3 = getValidatedInput("Enter integer 3: ");
119
120
121         // Exception-safe dynamic memory allocation (Rule: ERR57-CPP and MEM50-CPP)
122         int* ptr1 = new(nothrow) int; // Allocates memory without throwing an exception (Rule: MEM50-CPP)
123         int* ptr2 = new(nothrow) int;
124         int* ptr3 = new(nothrow) int;
125
126         // Memory allocation check (Rule: MEM51-CPP, ERR50-CPP)
127         if (!ptr1 || !ptr2 || !ptr3) {
128             cerr << "\n--- ERROR: Memory allocation failed. Exiting program.\n"; // Error message
129             // Clean-up allocated memory before exiting (Rule: MEM51-CPP)
130             delete ptr1;
131             delete ptr2;
132             delete ptr3;
133             return 1; // Exit the program with an error code
134         }
135
136         // Stores the validated integers in dynamically allocated memory (Rule: EXP34-C: Avoid dangling references)
137         // Note that the address-of operator (&) is not needed
138         // The memory address was allocated using 'new(nothrow) int' at the pointer initialization step
139         *ptr1 = num1;
140         *ptr2 = num2;
141         *ptr3 = num3;

```

Continue next page

Figure 11*Source Code in IDE lines 142-175 (Main Function Part-2)*

The screenshot shows a code editor window with the title "CTA-3-integerPointers.cpp:1". The code is displayed in a dark-themed editor with syntax highlighting. The code itself is as follows:

```
142 // Displays variable values
143 cout << "\nValues stored in variables:\n\n";
144 cout << "num1 = " << num1 << "\n"
145     << "num2 = " << num2 << "\n"
146     << "num3 = " << num3 << "\n" << endl;
147
148 // Displays the values pointed by ptr1, ptr2, and ptr3
149 // checks for null pointer (Rule: EXP53-CPP)
150 displayPointer("ptr1", ptr1);
151 displayPointer("ptr2", ptr2);
152 displayPointer("ptr3", ptr3);
153
154 // Deallocate memory properly preventing memory leaks (Rule: MEM51-CPP)
155 delete ptr1;
156 delete ptr2;
157 delete ptr3;
158 }
159
160 // catch all unhandled exceptions
161 // (Rule: ERR51-CPP. Handle all exceptions, ERR50-CPP. Do not abruptly terminate the program)
162 catch (const exception& e) {
163     // Catch any standard exceptions thrown in the try block
164     cerr << "\n--- ERROR: Exception caught: " << e.what() << endl;
165     return 1; // Exit the program with an error code
166 }
167 catch (...) {
168     // Catch any non-standard exceptions
169     cerr << "\n--- ERROR: Unknown exception caught. Exiting program.\n";
170     return 1; // Exit the program with an error code
171 }
172
173 return 0; // Successful program termination
174 }
```

The code implements a main function that displays variable values, checks for null pointers, deallocates memory, and handles exceptions. It includes comments explaining the purpose of each section and adherence to specific coding rules.

Continue next page

Figure 12Source Code in IDE lines 176-231 (Function Definition Part-1 - `getValidatedInput()`)

CTA-3-integerPointers.cpp:1 ➔ X

Critical-Thinking-3 (Global Scope)

```

176 // -----
177 // =====
178 /* -----
179 |   Function Definitions   |
180 -----
181 // -----
182 // -----
183 /**
184 * Prompts the user to enter an integer and validates the input
185 * It will prompt the user until a valid integer is entered
186 *
187 * Note:
188 * The standard integer is typically 4 bytes, it is platform dependent
189 * The Program accepts whitespaces before and after the integer value
190 *
191 * Handles Rules:
192 * - ERR50-CPP. Do not abruptly terminate the program
193 * - ERR51-CPP. Handle all exceptions
194 * - INT50-CPP. Do not cast to an out-of-range enumeration value
195 * - STR50-CPP. Guarantee that storage for strings has sufficient space
196 *   for character data and the null terminator
197 *
198 * @param prompt The message to user
199 * @return validated integer
200 */
201 int getValidatedInput(const string& prompt) noexcept(false) {
202     while (true) {
203         string input;
204
205         cout << prompt;
206
207         // Gets the entire line of input and stores it in a string object
208         // Using std::string is safer for string manipulation (STR50-CPP)
209         getline(cin, input);
210
211         //----- Whitespace Handling -----
212
213         // Remove leading and trailing whitespaces from the input
214         size_t start = input.find_first_not_of(" \t\n\r");
215         size_t end = input.find_last_not_of(" \t\n\r");
216
217         // Check if the input only contains whitespaces
218         // The following will be true if after removing all the whitespaces
219         // and if the input contain only whitespace
220         // 'string::npos' no position meaning that start has no position after parsing
221         if (start == string::npos) {
222             cerr << "--- Invalid input: Please enter an integer value.\n";
223             // skips the rest of the current while loop iteration
224             continue; // Prompt the user again
225         }
226
227         // Trim the input to remove leading and trailing whitespaces
228         input = input.substr(start, end - start + 1);
229
230     }
231 }
```

Figure 13

Source Code in IDE lines 176-231 (Function Definition Part-1 - getValidatedInput())

CTA-3-integerPointers.cpp:1 X

Critical-Thinking-3 (Global Scope)

```
232 //---- Check if the input is an integer ----
233
234 // Try-catch block to handle all exceptions (ERR51-CPP, ERR50-CPP)
235 try {
236     // Holds the position where parsing stopped
237     size_t pos;
238     // Convert the inputted string to a long integer using stoi (Rule: INT50-CPP)
239     // used to check if the string entered is an integer
240     long temp = stoi(input, &pos);
241
242     // Check if all the string was converted to a long temp (Rule: INT50-CPP)
243     if (pos != input.size()) { // If true invalid characters were entered
244         // There are remaining characters after the parsed number
245         // Non-integer characters detected
246         throw invalid_argument("---- Invalid input: Non-integer characters detected.");
247     }
248
249     // Check if long int parsed from the string is within the range of an int
250     if (temp > INT_MAX || temp < INT_MIN) { // Prevents integer overflow/underflow (Rule: INT50-CPP)
251         throw out_of_range("---- Invalid input: Input is out of integer range.");
252     }
253
254     return static_cast<int>(temp); // Return the validated integer
255 }
256
257 catch (const invalid_argument& e) {
258     // Handle invalid input (non-integer characters or floating-point numbers)
259     if (input.find('.') != string::npos) {
260         cerr << "---- Invalid input: Floating-point numbers not allowed. Please enter an integer.\n";
261     }
262     else {
263         cerr << "---- Invalid input: Non-integer, non-digit characters not allowed. Please enter a valid integer.\n";
264     }
265 }
266 catch (const out_of_range& e) {
267     // Handle input numbers that are out of the allowable int range
268     cerr << "---- Invalid input: Number out of range. Please enter an integer between "
269     << INT_MIN << " and " << INT_MAX << ".\n";
270 }
271 catch (...) {
272     // Catch any other exceptions and provide a generic error message
273     cerr << "---- Invalid input: Please enter a valid integer.\n";
274 }
275
276 // -----
277
278 }
```

Figure 14

Source Code in IDE lines 279-296 (Function Definition Part-3 - displayPointer())

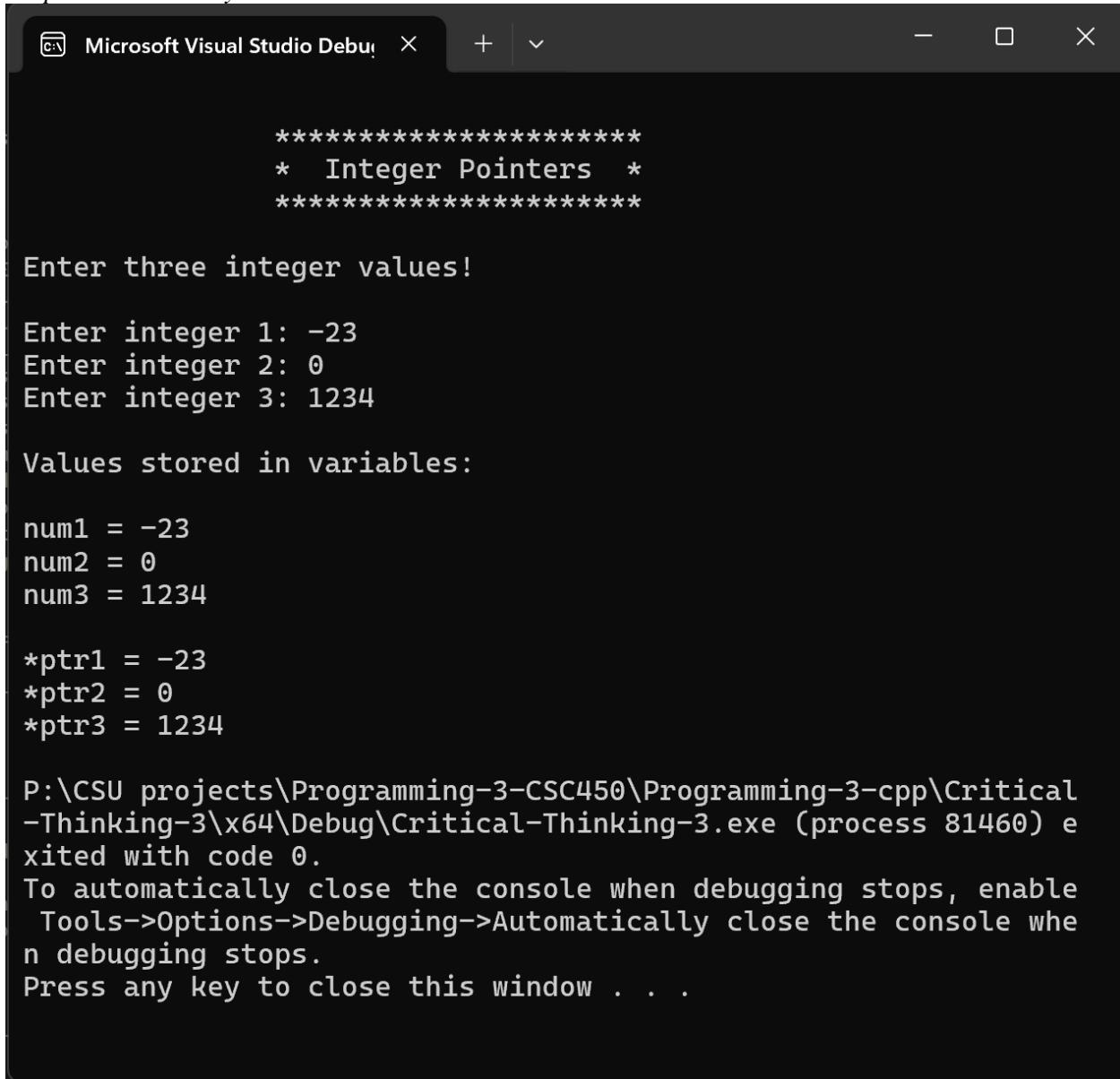
CTA-3-integerPointers.cpp:1 X

Critical-Thinking-3 (Global Scope)

```
279 // /**
280 * Displays the value pointed to by ptr.
281 * checks if the pointer is not null, if it is, it displays an error,
282 * otherwise it display the value pointed (Rule: EXP53-CPP)
283 *
284 * @param name The name of the pointer for display purposes.
285 * @param ptr The pointer to an integer
286 */
287 void displayPointer(const char* name, int* ptr) {
288     if (ptr == nullptr) { // Check if the pointer is null (Rule: EXP53-CPP)
289         cerr << name << "\n--- ERROR: null pointer!\n";
290     }
291     else { // Dereference the pointer and display its value
292         cout << "*" << name << " = " << *ptr << endl;
293     }
294 }
295
296 // ---
```

Output Screenshots

Figure 15
Outputs Functionality



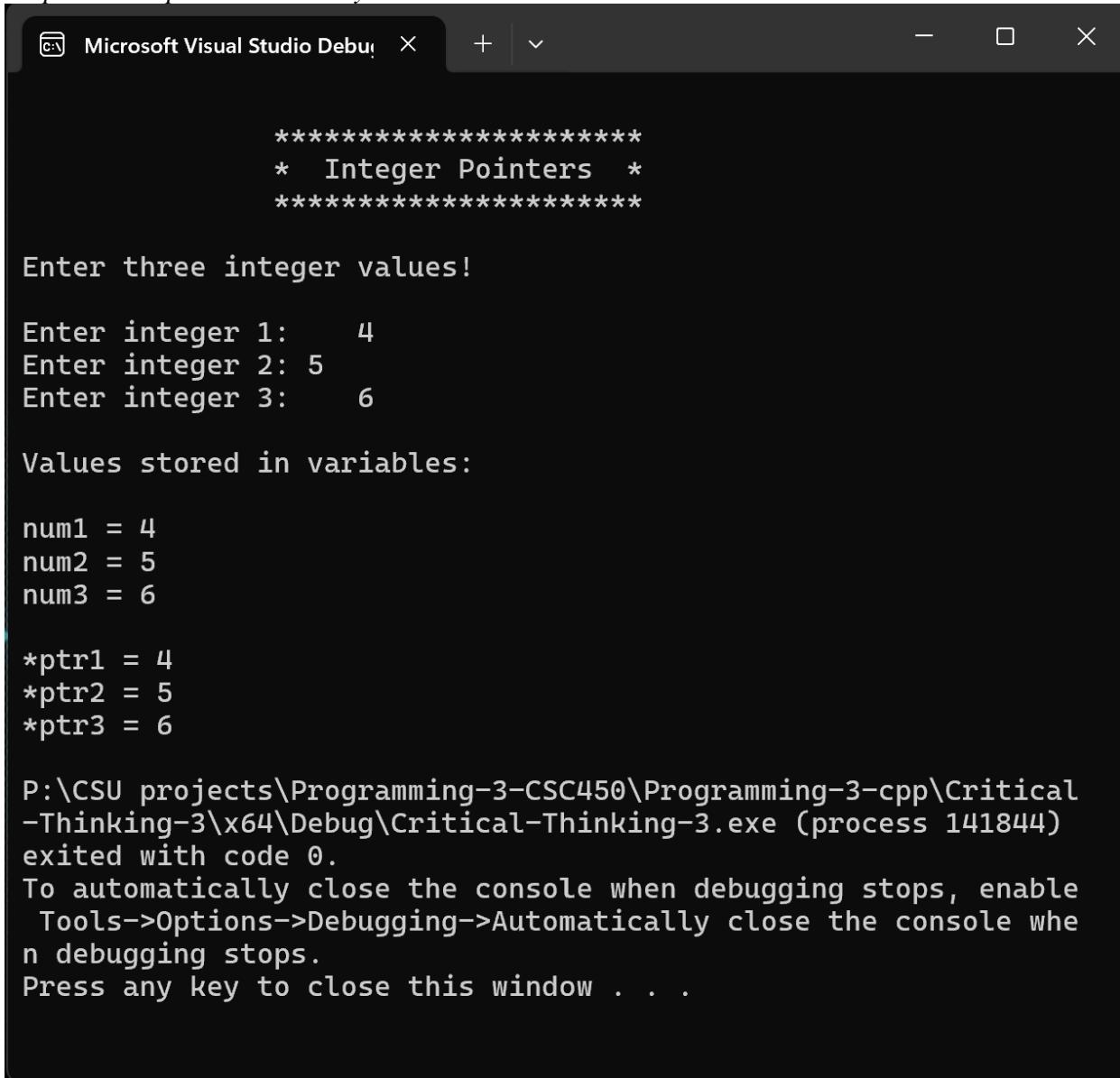
The screenshot shows a Microsoft Visual Studio Debug window. The title bar says "Microsoft Visual Studio Debug". The window displays the following text:

```
*****
* Integer Pointers *
*****  
Enter three integer values!  
Enter integer 1: -23  
Enter integer 2: 0  
Enter integer 3: 1234  
Values stored in variables:  
num1 = -23  
num2 = 0  
num3 = 1234  
*ptr1 = -23  
*ptr2 = 0  
*ptr3 = 1234  
P:\CSU projects\Programming-3-CSC450\Programming-3-cpp\Critical  
-Thinking-3\x64\Debug\Critical-Thinking-3.exe (process 81460) e  
xited with code 0.  
To automatically close the console when debugging stops, enable  
Tools->Options->Debugging->Automatically close the console whe  
n debugging stops.  
Press any key to close this window . . .
```

Note: This shows the functionality of the program. Integers are whole number that can be positive, negative, or zero.

Continue next page

Figure 16
Outputs Whitespace Functionality



The screenshot shows a Microsoft Visual Studio Debug window. The title bar reads "Microsoft Visual Studio Debug". The window displays the following text:

```
*****
* Integer Pointers *
*****  
Enter three integer values!  
Enter integer 1:    4  
Enter integer 2: 5  
Enter integer 3:   6  
Values stored in variables:  
num1 = 4  
num2 = 5  
num3 = 6  
*ptr1 = 4  
*ptr2 = 5  
*ptr3 = 6  
P:\CSU projects\Programming-3-CSC450\Programming-3-cpp\Critical  
-Thinking-3\x64\Debug\Critical-Thinking-3.exe (process 141844)  
exited with code 0.  
To automatically close the console when debugging stops, enable  
Tools->Options->Debugging->Automatically close the console whe  
n debugging stops.  
Press any key to close this window . . .
```

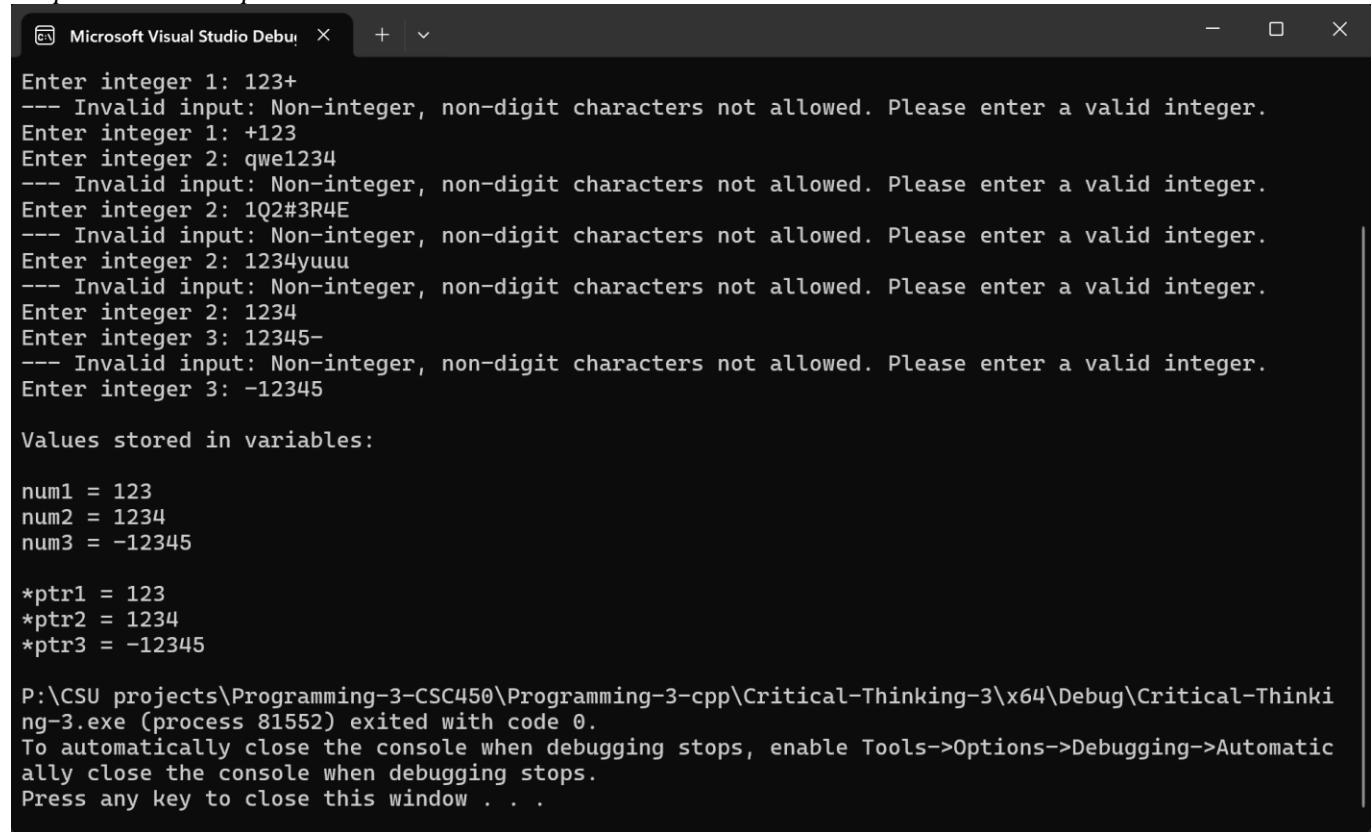
Note: The program accepts whitespaces to be entered before and/or after the integer value. It will crop out the spaces before storing the values as an integer. Integer 1 has three spaces before the character ‘4’. Integer 2 has three spaces after the character ‘5’. Integer 3 has three spaces before and after the character ‘6’.

Continue next page

Figure 17
Outputs Validation part-1

Note: The program validates integers only. Integer 1 example - Spaces between digits are not accepted. Integer 2 example – Decimal numbers are not accepted. Integer 3 example – only C++ regular ‘*int*’ data type are accepted.

Continue next page

Figure 18*Outputs Validation part-I*


The screenshot shows a Microsoft Visual Studio Debug window. The console output is as follows:

```

Microsoft Visual Studio Debug + - X

Enter integer 1: 123+
--- Invalid input: Non-integer, non-digit characters not allowed. Please enter a valid integer.
Enter integer 1: +123
Enter integer 2: qwe1234
--- Invalid input: Non-integer, non-digit characters not allowed. Please enter a valid integer.
Enter integer 2: 1Q2#3R4E
--- Invalid input: Non-integer, non-digit characters not allowed. Please enter a valid integer.
Enter integer 2: 1234yuuu
--- Invalid input: Non-integer, non-digit characters not allowed. Please enter a valid integer.
Enter integer 2: 1234
Enter integer 3: 12345-
--- Invalid input: Non-integer, non-digit characters not allowed. Please enter a valid integer.
Enter integer 3: -12345

Values stored in variables:

num1 = 123
num2 = 1234
num3 = -12345

*ptr1 = 123
*ptr2 = 1234
*ptr3 = -12345

P:\CSU projects\Programming-3-CSC450\Programming-3-cpp\Critical-Thinking-3\x64\Debug\Critical-Thinking-3.exe (process 81552) 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 characters ‘-’ and ‘+’ are only accepted at the beginning of the string to represent positive and negative integers, see example Integer 1 and Integer 3. In addition to the characters ‘-’ and ‘+’ only digits characters are accepted, see example Integer 2.

As shown in Figures 15 through 18 the program runs without any issues displaying the correct outputs as expected.