Documentation: Critical Thinking 3

Alejandro Ricciardi

Colorado State University Global

CSC450: Programming III

Professor: Reginald Haseltine

October 27, 2024

Table of Contents

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 management 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 tores 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)

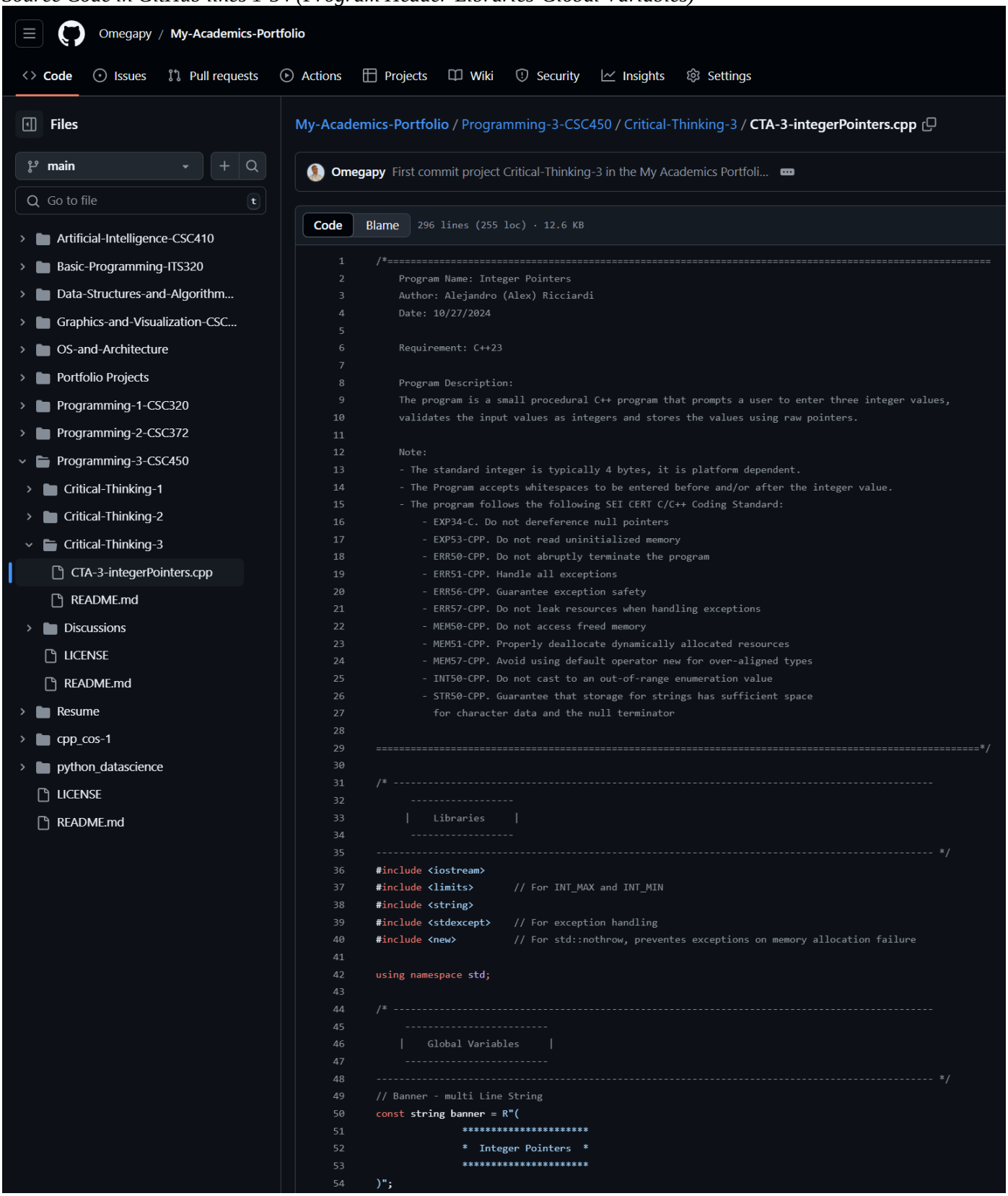
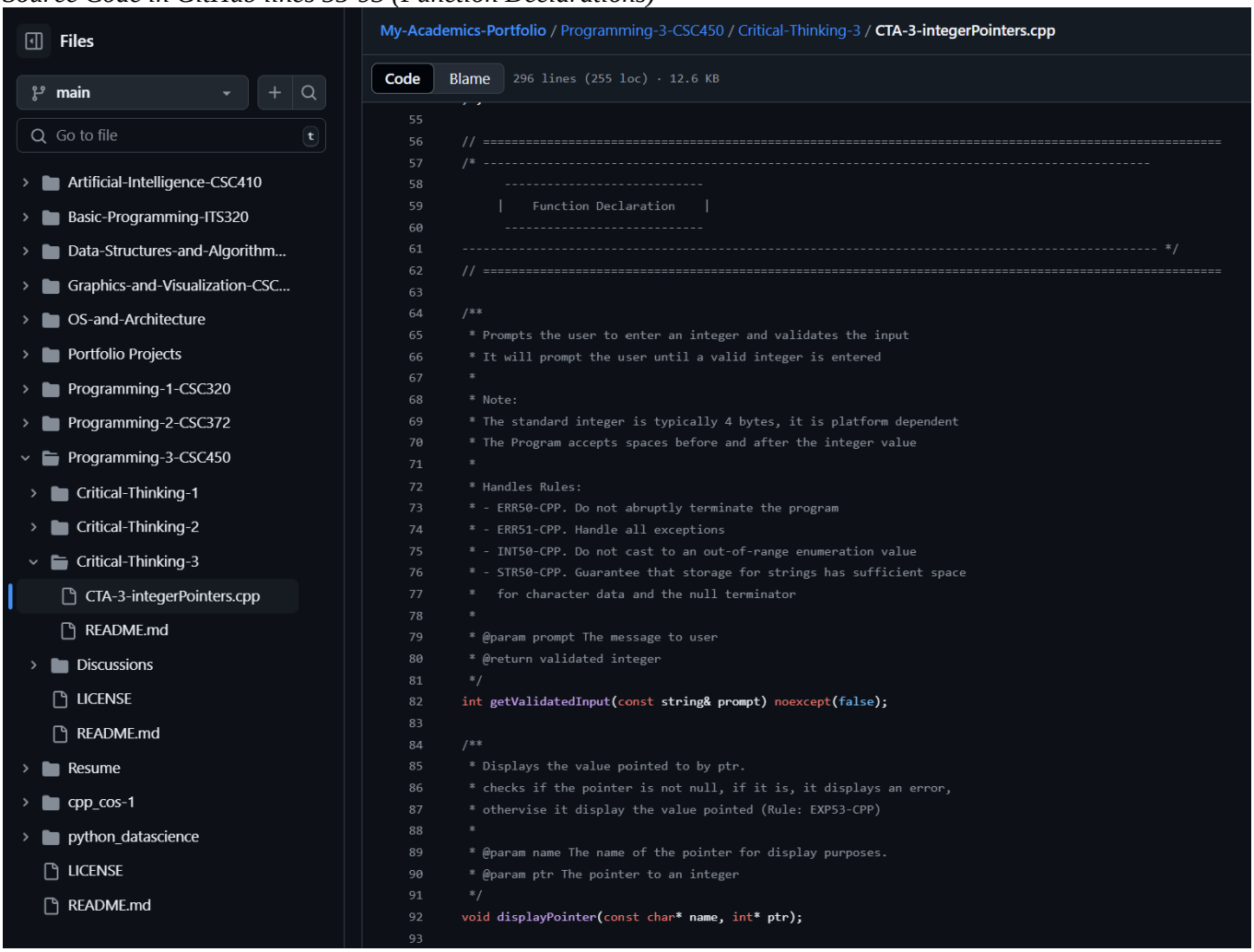


Figure 2

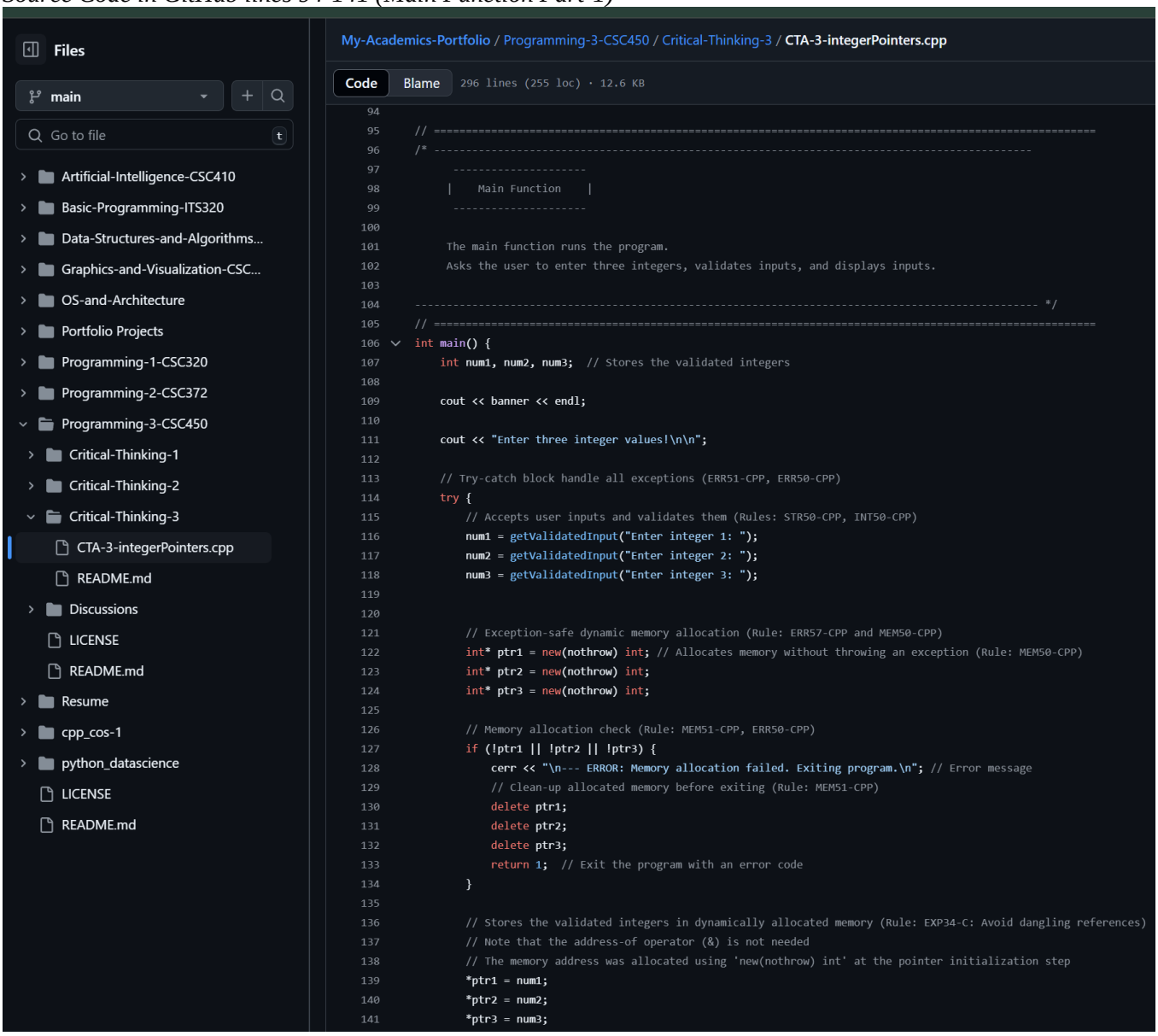
Source Code in GitHub lines 55-93 (Function Declarations)



Continue next page

Figure 3

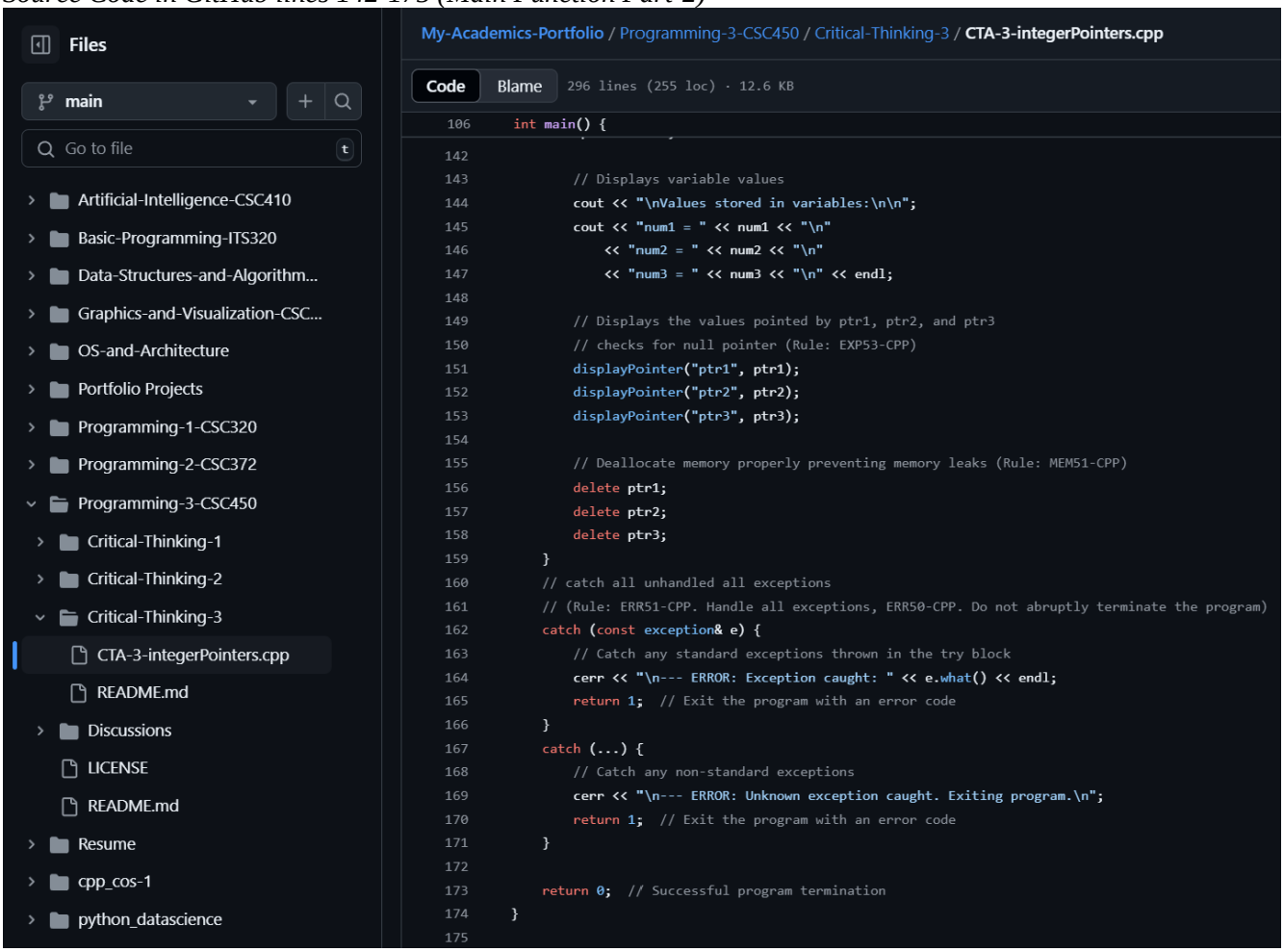
Source Code in GitHub lines 94-141 (Main Function Part-1)



Continue next page

Figure 4

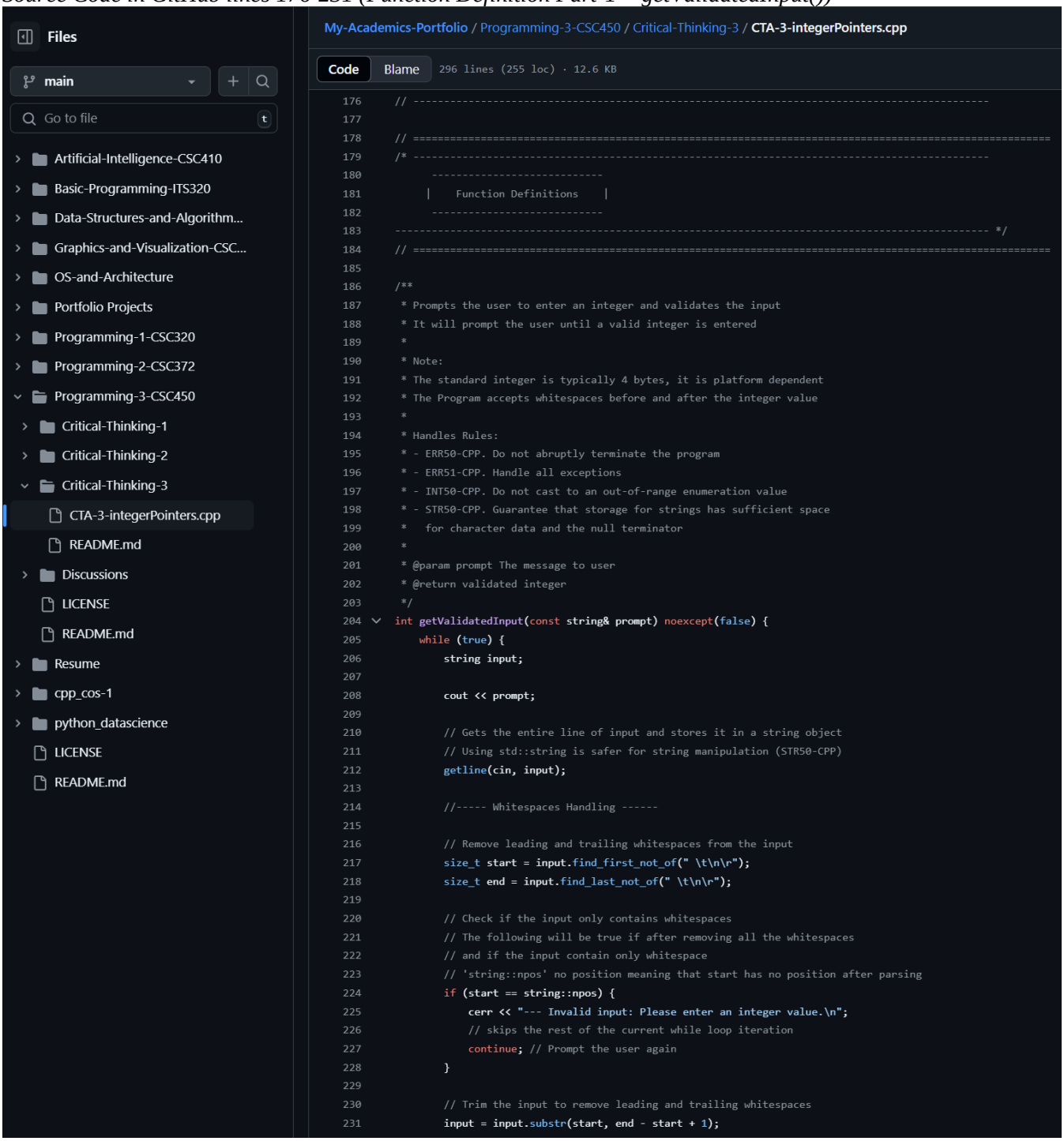
Source Code in GitHub lines 142-175 (Main Function Part-2)



Continue next page

Figure 5

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



Continue next page

Figure 6

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

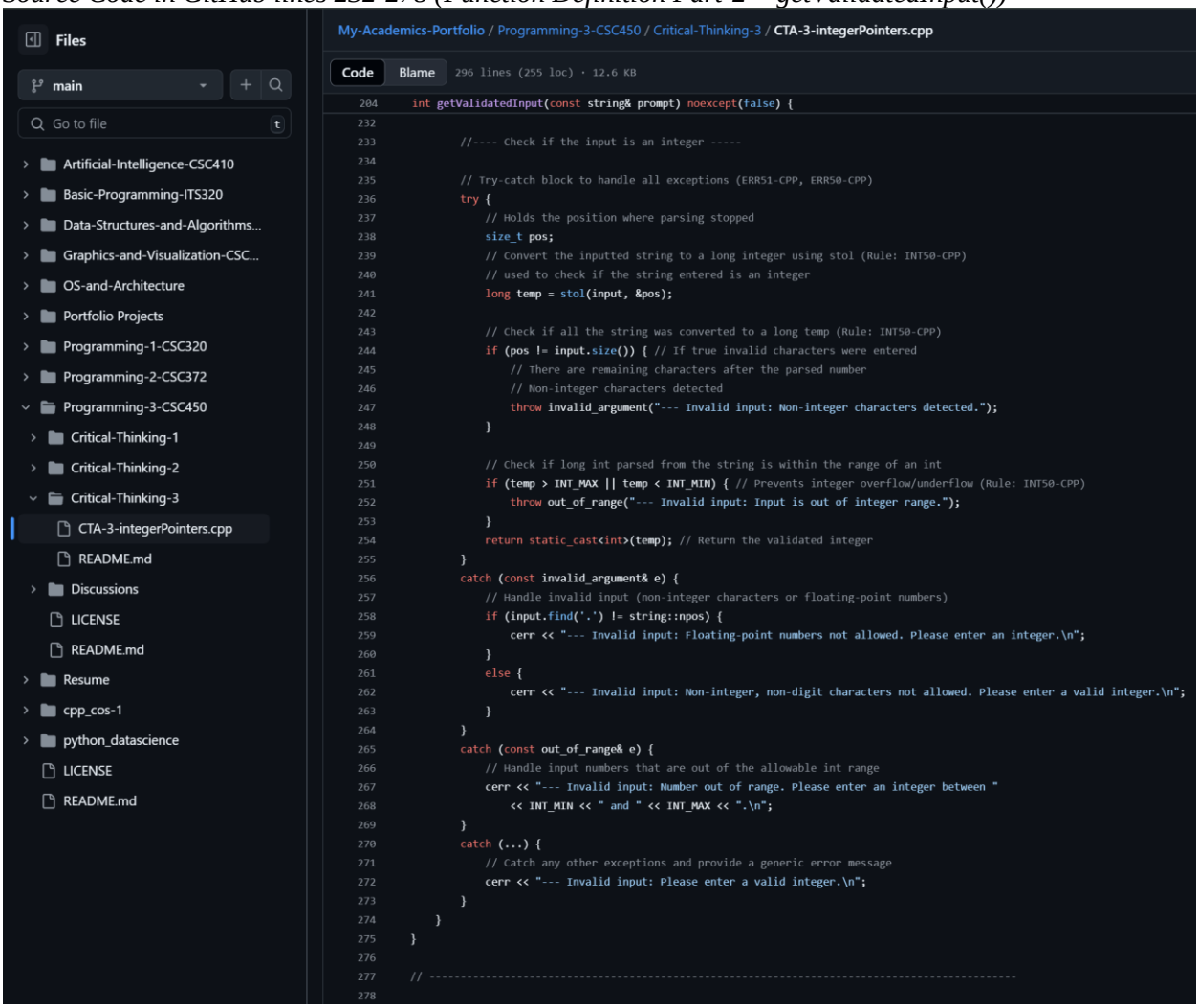
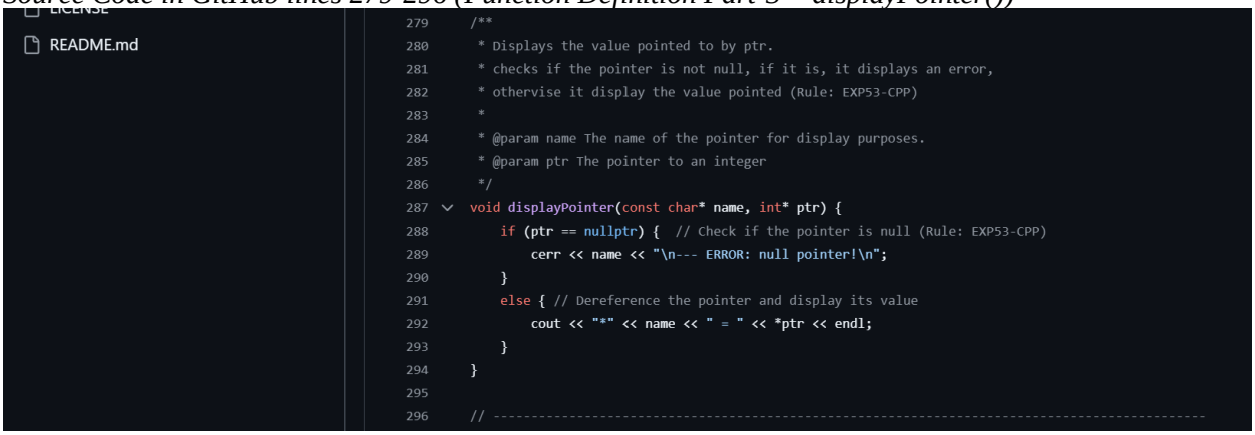


Figure 7

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



Source Code in IDE

Figure 8

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

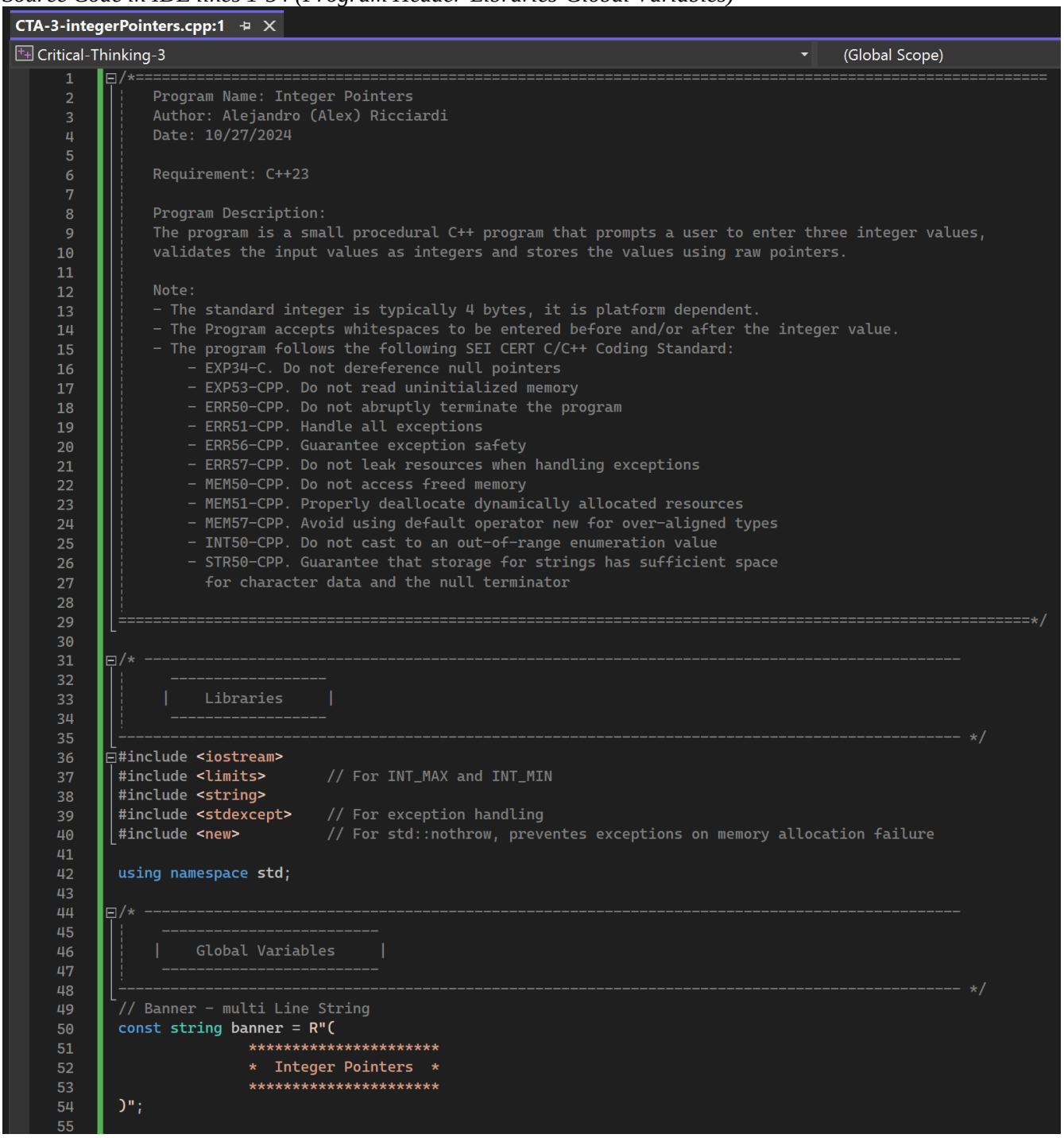
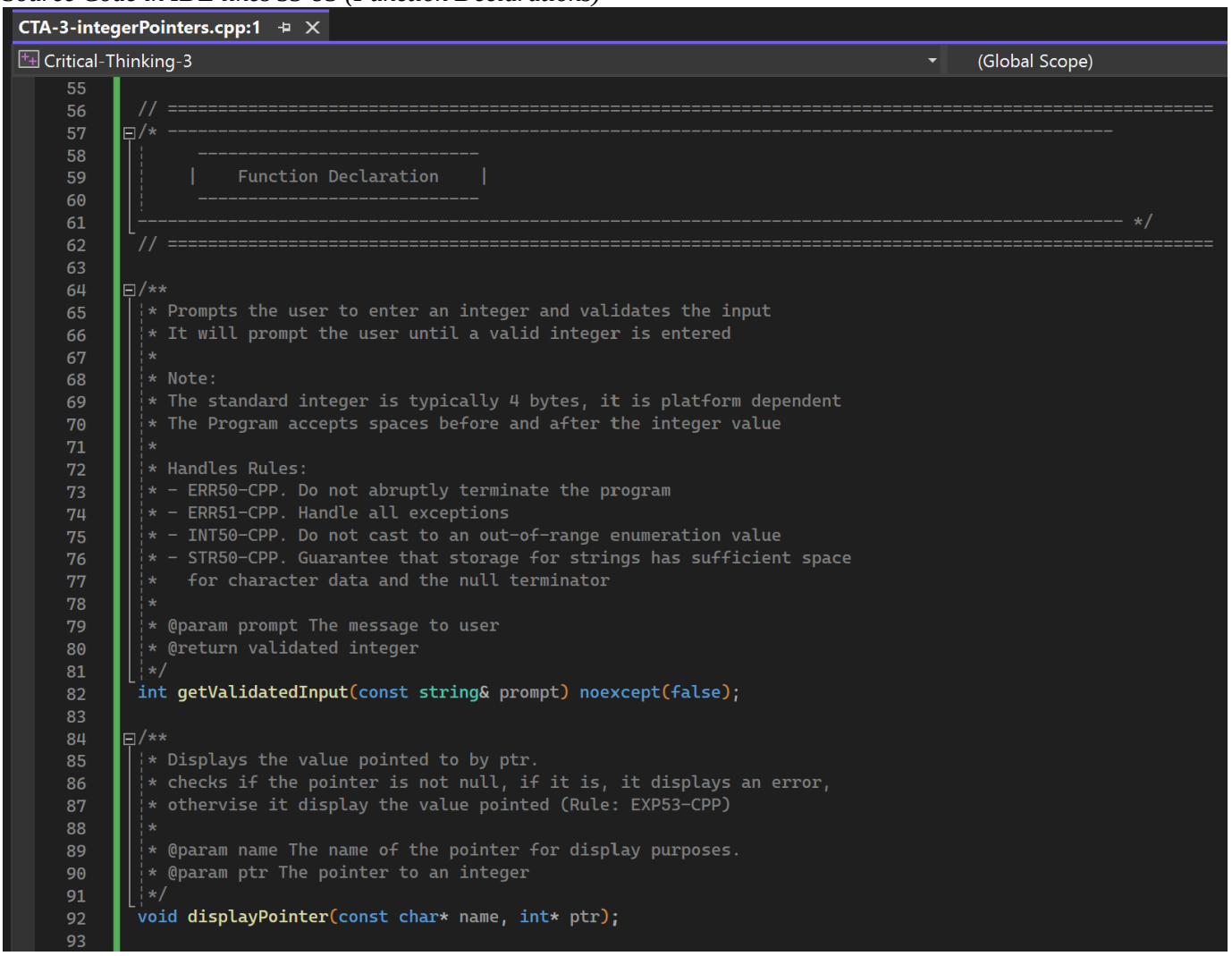


Figure 9

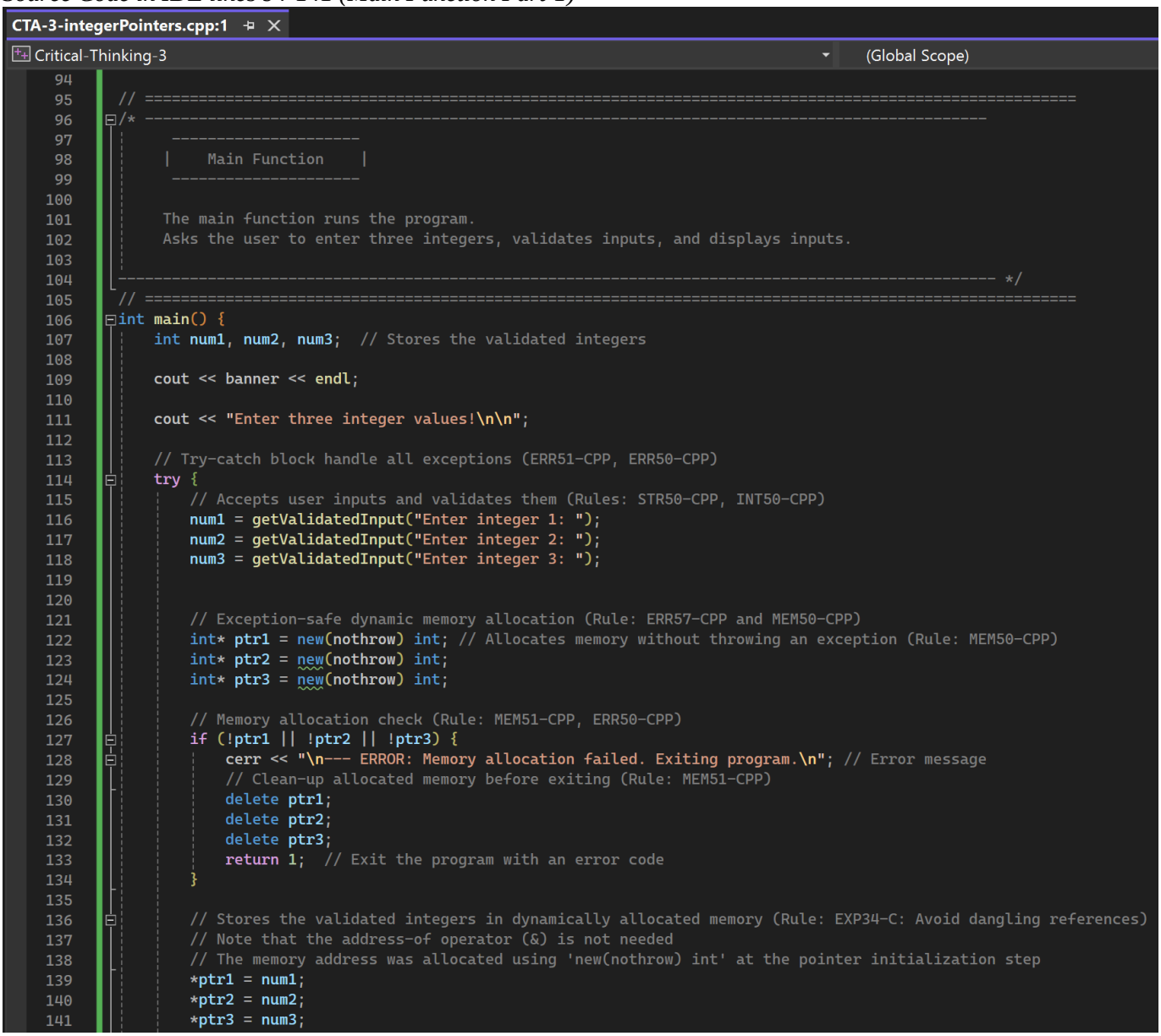
Source Code in IDE lines 55-93 (Function Declarations)



Continue next page

Figure 10

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

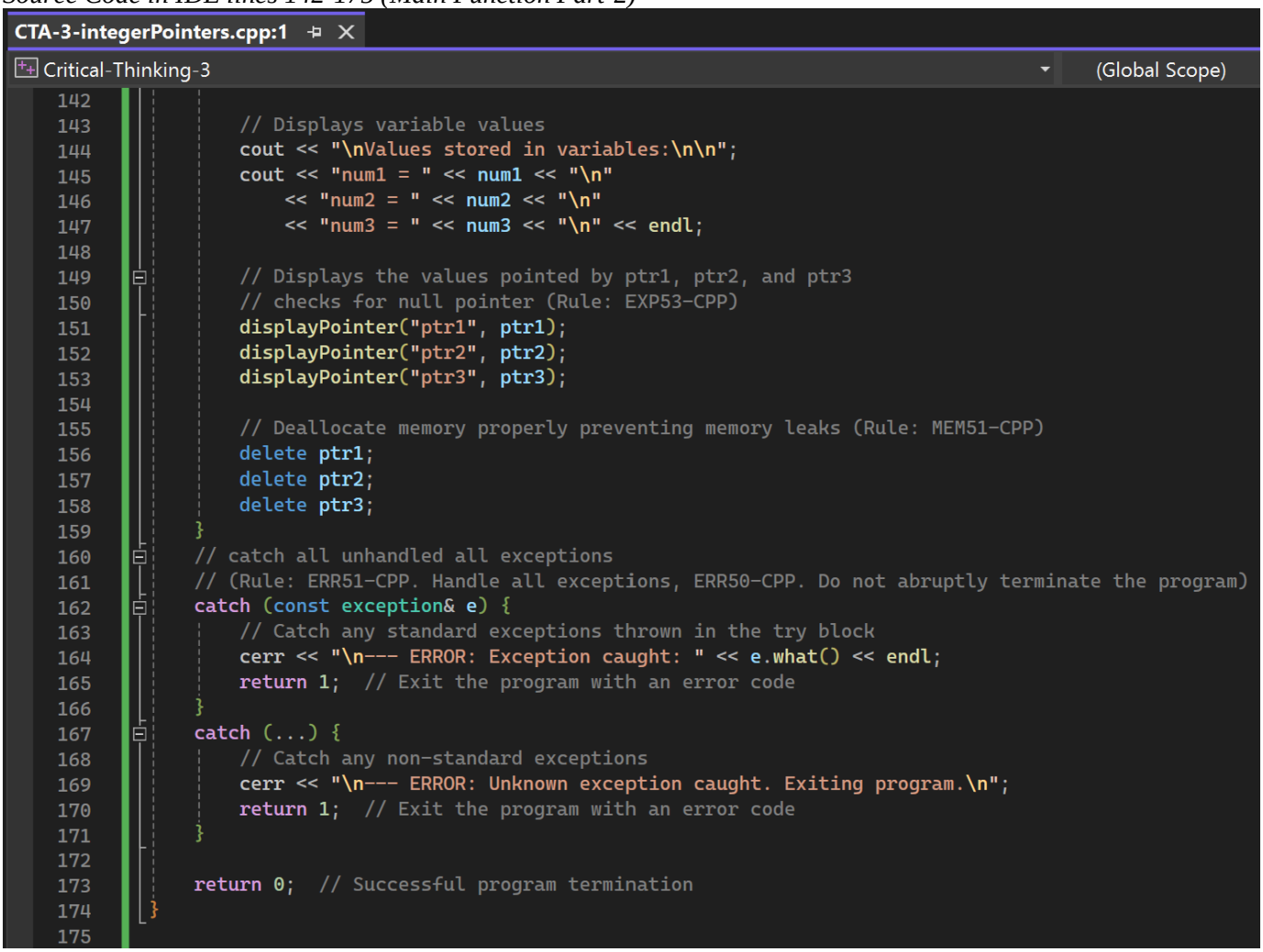



```
94
95  // ============================================================================================================
96  /* ------------------------------------------------------------------------------------------------------
97         --------------------
98      |    Main Function    |
99         --------------------
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

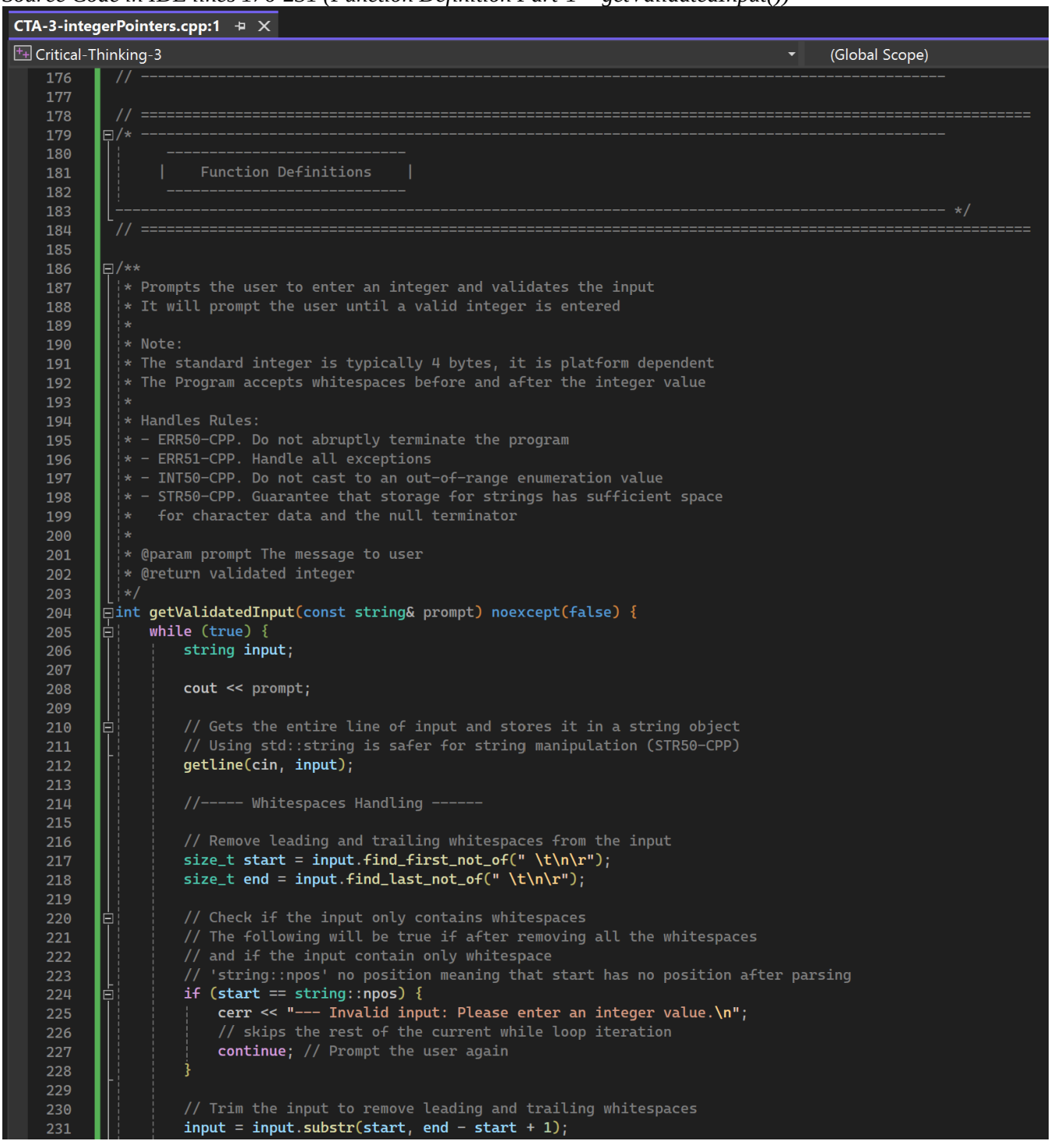
Source Code in IDE lines 142-175 (Main Function Part-2)



Continue next page

Figure 12

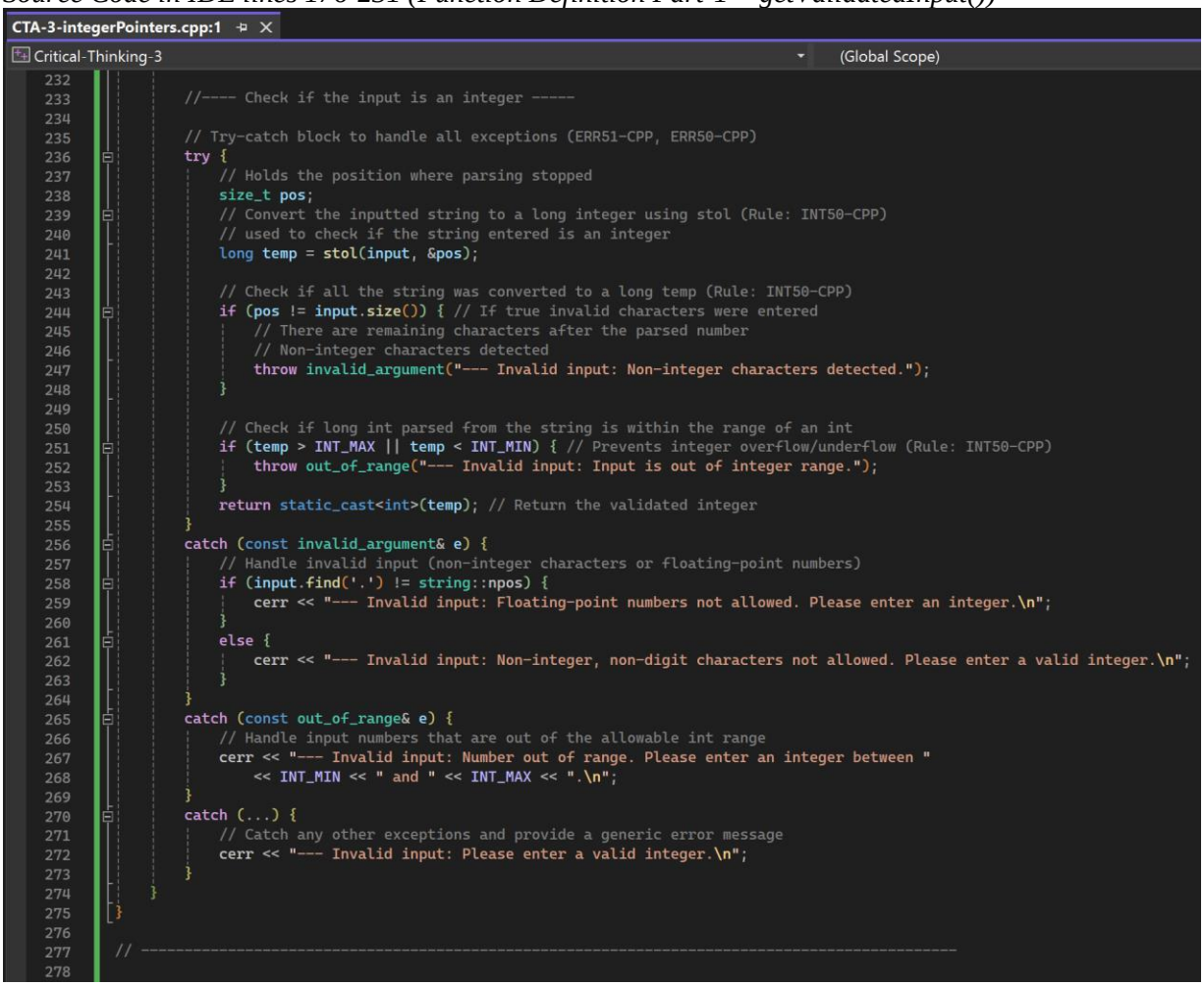
Source Code in IDE lines 176-231 (Function Definition Part-1 - getValidatedInput())



```
CTA-3-integerPointers.cpp:1
Critical-Thinking-3                                   (Global Scope)
176  // ------------------------------------------------------------------------------------------------
177
178  // ================================================================================================
179 /* ------------------------------------------------------------------------------------------------
180      ----------------------------
181     |    Function Definitions    |
182      ----------------------------
183 ------------------------------------------------------------------------------------------------ */
184  // ================================================================================================
185
186 /**
187  * Prompts the user to enter an integer and validates the input
188  * It will prompt the user until a valid integer is entered
189  *
190  * Note:
191  * The standard integer is typically 4 bytes, it is platform dependent
192  * The Program accepts whitespaces before and after the integer value
193  *
194  * Handles Rules:
195  * - ERR50-CPP. Do not abruptly terminate the program
196  * - ERR51-CPP. Handle all exceptions
197  * - INT50-CPP. Do not cast to an out-of-range enumeration value
198  * - STR50-CPP. Guarantee that storage for strings has sufficient space
199  *   for character data and the null terminator
200  *
201  * @param prompt The message to user
202  * @return validated integer
203  */
204 int getValidatedInput(const string& prompt) noexcept(false) {
205     while (true) {
206         string input;
207
208         cout << prompt;
209
210         // Gets the entire line of input and stores it in a string object
211         // Using std::string is safer for string manipulation (STR50-CPP)
212         getline(cin, input);
213
214         //----- Whitespaces Handling ------
215
216         // Remove leading and trailing whitespaces from the input
217         size_t start = input.find_first_not_of(" \t\n\r");
218         size_t end = input.find_last_not_of(" \t\n\r");
219
220         // Check if the input only contains whitespaces
221         // The following will be true if after removing all the whitespaces
222         // and if the input contain only whitespace
223         // 'string::npos' no position meaning that start has no position after parsing
224         if (start == string::npos) {
225             cerr << "--- Invalid input: Please enter an integer value.\n";
226             // skips the rest of the current while loop iteration
227             continue; // Prompt the user again
228         }
229
230         // Trim the input to remove leading and trailing whitespaces
231         input = input.substr(start, end - start + 1);
```

Figure 13

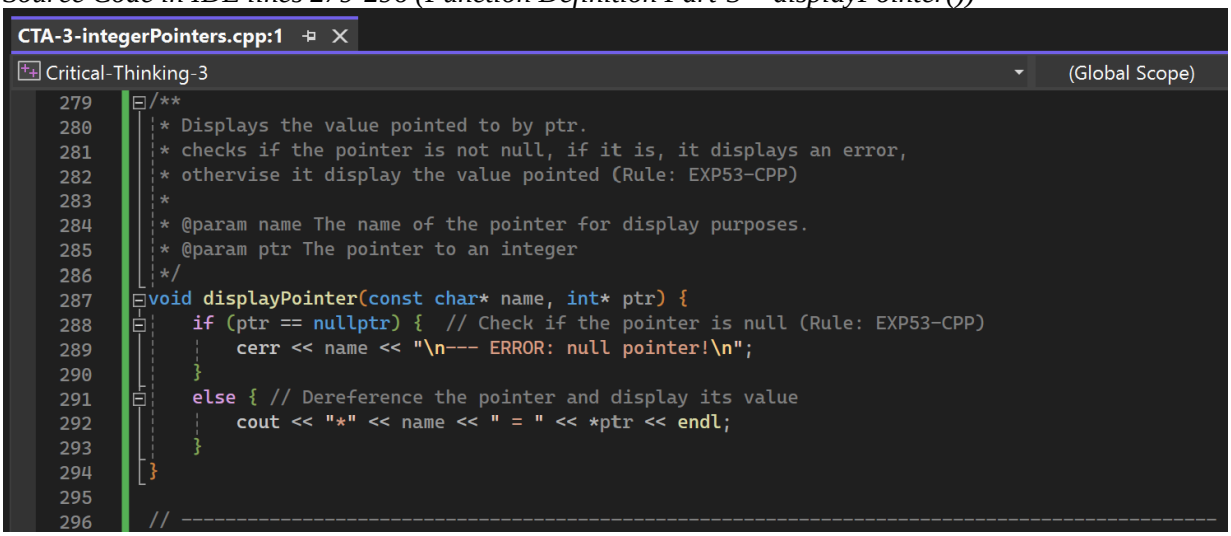
Source Code in IDE lines 176-231 (Function Definition Part-1 - getValidatedInput())



```
CTA-3-integerPointers.cpp:1
Critical-Thinking-3                                   (Global Scope)
232
233         //---- 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 14

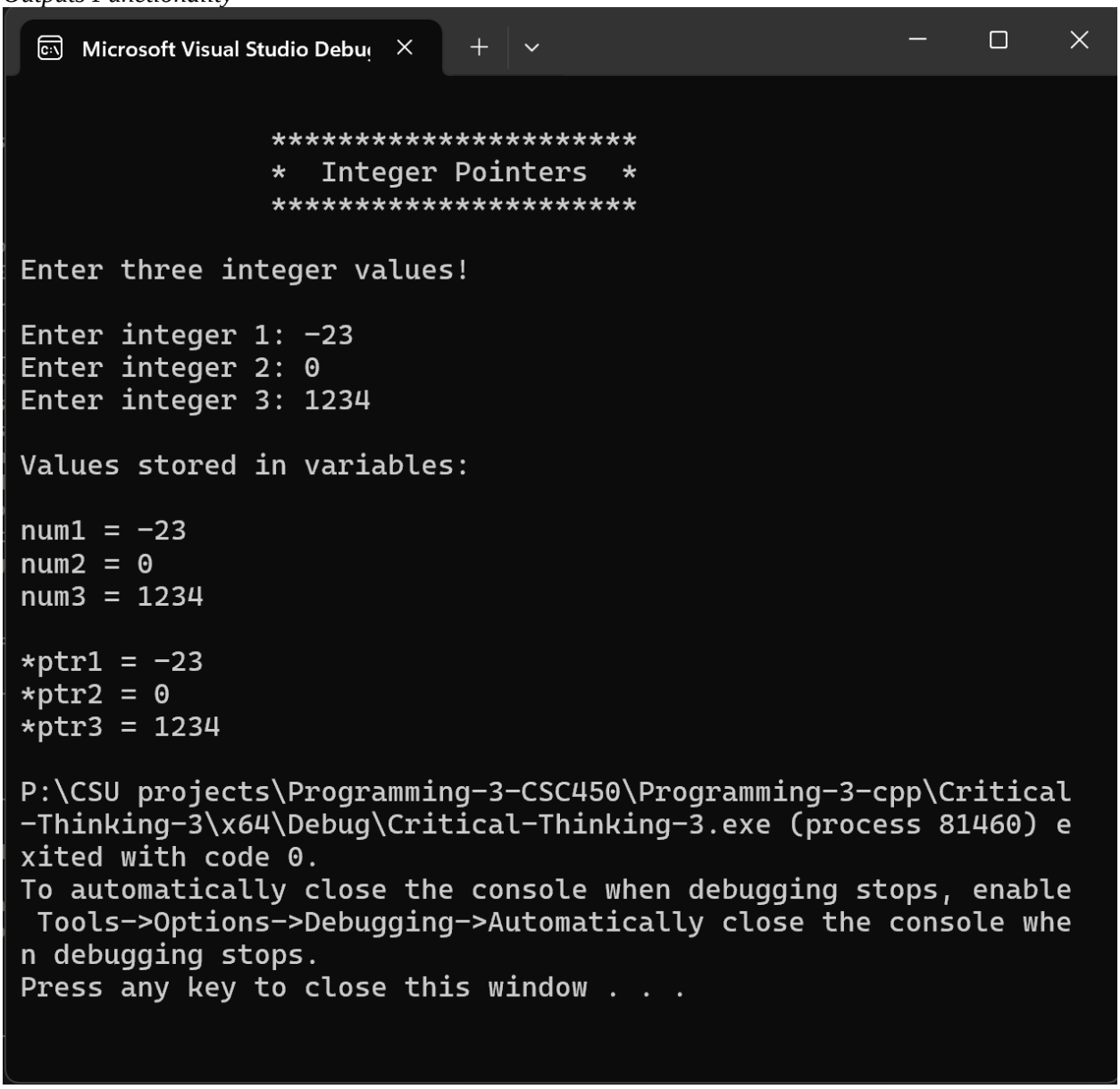
Source Code in IDE lines 279-296 (Function Definition Part-3 - displayPointer())



```
CTA-3-integerPointers.cpp:1
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  * othervise 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

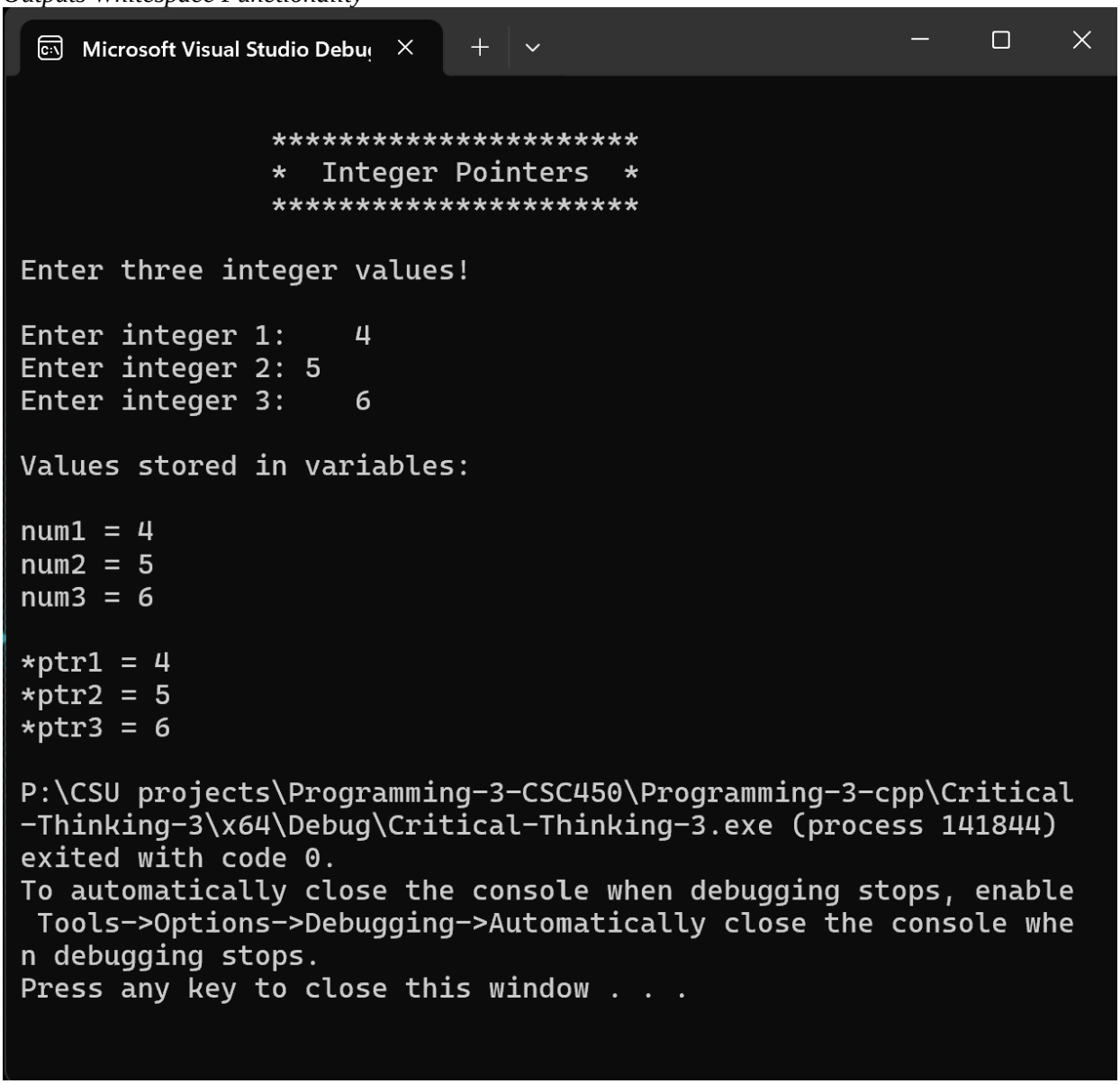


Note: This shows the functionality of the program. Integers are whole number that can be positive, negative, or zero.

Continue next page

Figure 16

Outputs Whitespace Functionality



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

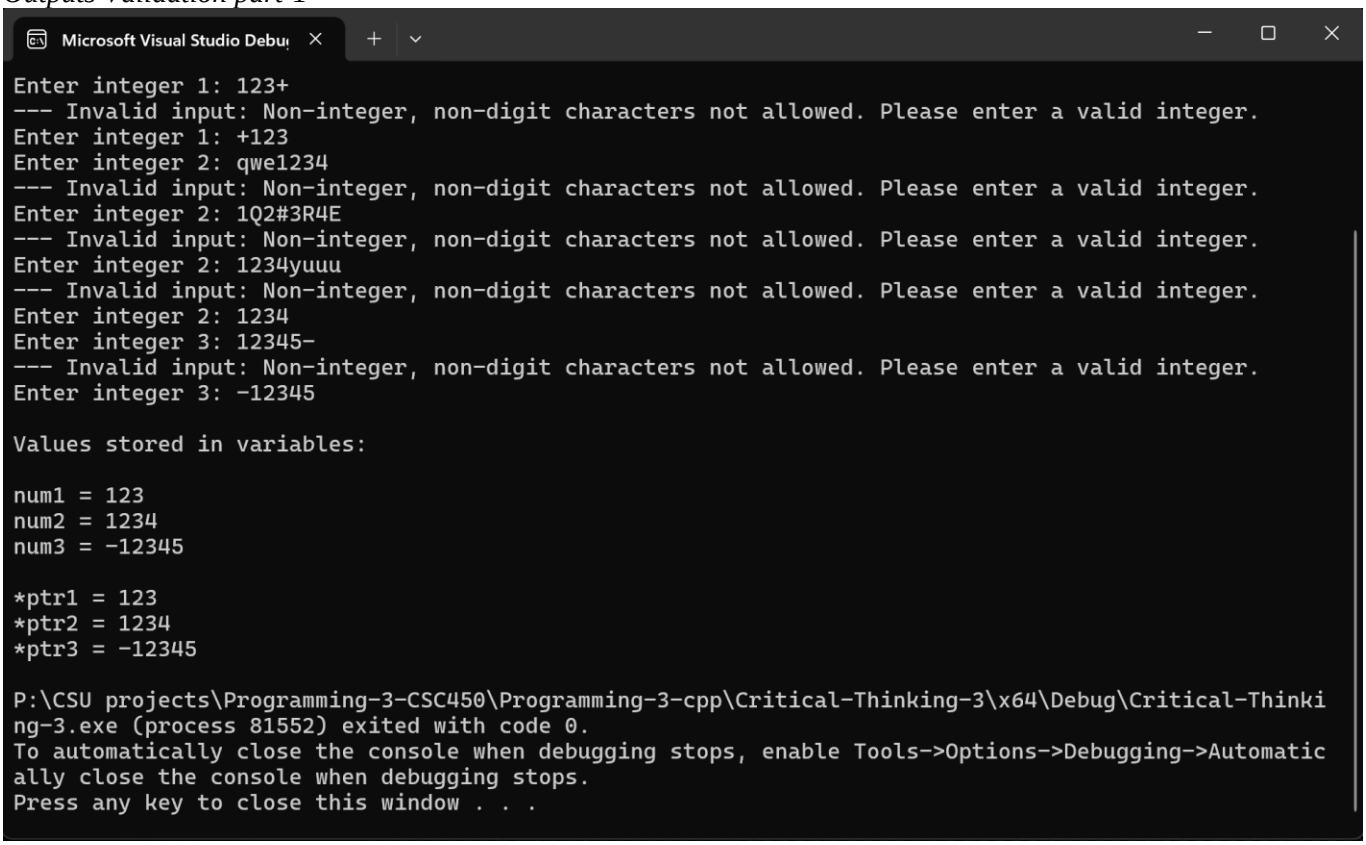
```
Microsoft Visual Studio Debu

Enter three integer values!

Enter integer 1: 3 4 5
--- Invalid input: Non-integer, non-digit characters not allowed. Please enter a valid integer.
Enter integer 1: 345
Enter integer 2: 12.34
--- Invalid input: Floating-point numbers not allowed. Please enter an integer.
Enter integer 2: 12
Enter integer 3: 11111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
--- Invalid input: Number out of range. Please enter an integer between -2147483648 and 2147483647.
Enter integer 3: -2147483647

Values stored in variables:

num1 = 345
num2 = 12
num3 = -2147483647

*ptr1 = 345
*ptr2 = 12
*ptr3 = -2147483647

P:\CSU projects\Programming-3-CSC450\Programming-3-cpp\Critical-Thinking-3\x64\Debug\Critical-Thinki
ng-3.exe (process 89424) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatic
ally close the console when debugging stops.
Press any key to close this window . . .
```

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-1



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.