# IT 230 Coding Activity Submission Template

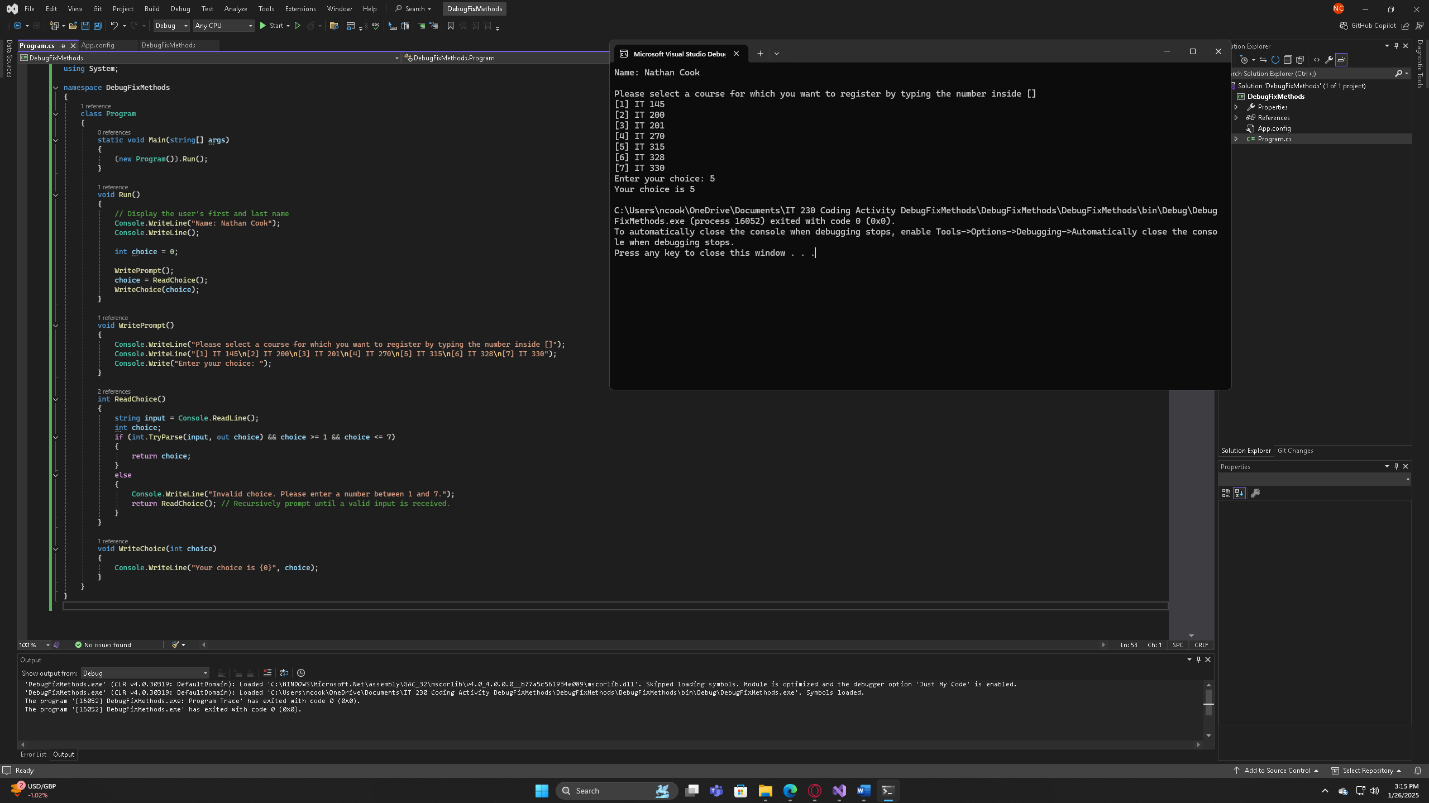
Submit your work on the coding activities for Modules One, Two, Three, Four, and Six in this document. In addition to this document, you should submit a ZIP file containing all your Visual Studio project files and source code that can be run in Visual Studio on a different computer.

For each coding activity, complete the following steps:

* Download and rename this document to meet the file naming conventions requested in the assignment instructions.
* Fill in the required information below by replacing the bracketed text with the relevant information.
* Submit this document and your ZIP file for grading and feedback. Your ZIP file should follow the same naming conventions.

Document your work in the coding activity by completing each of the following items:

1. Provide a screenshot of the output that resulted from running your program successfully in Visual Studio. See the coding assignment instructions for an example of what should be included in the screenshot. Your screenshot must include the following elements:
   1. Your last name as the first printed text on the screen
   2. Verification that the program is fully functioning and data results are accurate for the given problem



1. Copy and paste the source code text you wrote for this assignment from the \*.cs file into the space below. Only providing the \*.cs files or a screenshot does not meet the requirements for this part of the assignment. Code should be logically organized. It should also follow proper syntax and conventions noted in the Coding Activity Guidelines and Rubric.

using System;

namespace DebugFixMethods

{

class Program

{

static void Main(string[] args)

{

(new Program()).Run();

}

void Run()

{

// Display the user's first and last name

Console.WriteLine("Name: Nathan Cook");

Console.WriteLine();

int choice = 0;

WritePrompt();

choice = ReadChoice();

WriteChoice(choice);

}

void WritePrompt()

{

Console.WriteLine("Please select a course for which you want to register by typing the number inside []");

Console.WriteLine("[1] IT 145\n[2] IT 200\n[3] IT 201\n[4] IT 270\n[5] IT 315\n[6] IT 328\n[7] IT 330");

Console.Write("Enter your choice: ");

}

int ReadChoice()

{

string input = Console.ReadLine();

int choice;

if (int.TryParse(input, out choice) && choice >= 1 && choice <= 7)

{

return choice;

}

else

{

Console.WriteLine("Invalid choice. Please enter a number between 1 and 7.");

return ReadChoice(); // Recursively prompt until a valid input is received.

}

}

void WriteChoice(int choice)

{

Console.WriteLine("Your choice is {0}", choice);

}

}

}

1. Show that you understand the task by explaining the design of your program in the space below. Include the process and steps you took to write your code. Explain how you arrived at the solution to the problem and completed the activity.

The goal of this program is to allow a user to register for a course by selecting a course number from a predefined list of courses. The program guides the user through a prompt, validates their input, and confirms their selection. Below is an explanation of the design process and the steps taken to write the code:

**Steps to Solve the Problem**

1. Understand the Requirements:
   * The program must prompt the user to select a course from a list of options.
   * It should validate the user’s input to ensure it corresponds to a valid course number.
   * The program should display the user’s choice as confirmation.
2. Design the Structure:
   * Main Method: Acts as the entry point of the program.
   * Run Method: The core function that organizes the workflow by calling other methods sequentially.
   * WritePrompt Method: Displays the list of courses and instructions to the user.
   * ReadChoice Method: Captures and validates the user’s input, ensuring it is within the expected range.
   * WriteChoice Method: Outputs the user’s final choice to confirm the selection.
3. Implementation Process:
   * Step 1: Set up the Main method to call the Run method, ensuring modularity.
   * Step 2: Create a WritePrompt method to display the course options to the user.
   * Step 3: Implement the ReadChoice method to read user input as a string, validate it using int.TryParse, and ensure the input is within the range of valid options (1–7). If invalid, prompt the user again until valid input is provided.
   * Step 4: Develop the WriteChoice method to output the user’s choice in a readable format.
   * Step 5: Test the program for different inputs, including invalid ones (e.g., letters or numbers outside the range), to confirm proper functionality.
4. Key Design Considerations:
   * Input Validation: Prevent invalid input from causing errors by checking for proper integer conversion and range.
   * Modularity: Divide the program into smaller, reusable methods to enhance readability and maintainability.
   * User-Friendly Experience: Provide clear instructions and error messages to guide the user through the process.
5. Reflect on your learning experience and what you learned from completing the activity.

Completing this activity was a valuable learning experience for me. I gained a deeper understanding of how to design a program that is both functional and user-friendly. One of the key lessons I learned was the importance of input validation. Before this activity, I didn’t fully realize how crucial it is to check for invalid input. Using techniques like int.TryParse showed me how to handle errors in a way that ensures the program works smoothly and reliably.

Breaking the program into smaller, manageable parts also helped me stay organized and less overwhelmed. By creating separate methods for tasks like writing the prompt, reading input, and displaying the user’s choice, I could focus on one part of the program at a time, which made debugging much easier. Additionally, testing the program with various inputs, such as numbers, letters, and out-of-range values, taught me the value of anticipating how users might interact with the program and preparing for unexpected inputs.

This activity also emphasized the importance of the user experience. I learned that providing clear instructions and helpful error messages can make even a simple program more intuitive and less frustrating for users. Initially, I wasn’t confident about how to handle some of the challenges, but by breaking the problem into steps and testing solutions, I was able to figure out an effective approach. This process has given me more confidence in my problem-solving skills and has shown me how to think critically when writing code.

Overall, this activity helped me understand the importance of planning, testing, and focusing on user needs when developing programs. It was a great opportunity to practice writing clean, functional code and to learn skills that I can apply to future projects.