**Problem Statements**

1. **Write a C program that declares an integer pointer, initializes it to point to an integer variable, and prints the value of the variable using the pointer.**

A computer screen shot of a code

Description automatically generated

1. **Create a program where you declare a pointer to a float variable, assign a value to the variable, and then use the pointer to change the value of the float variable. Print both the original and modified values.**

A screen shot of a computer program

Description automatically generated

1. **Given an array of integers, write a function that takes a pointer to the array and its size as arguments. Use pointer arithmetic to calculate and return the sum of all elements in the array.**

A screen shot of a computer program

Description automatically generated

1. **Write a program that demonstrates the use of a null pointer. Declare a pointer, assign it a null value, and check if it is null before attempting to dereference it.**

A screen shot of a computer

Description automatically generated

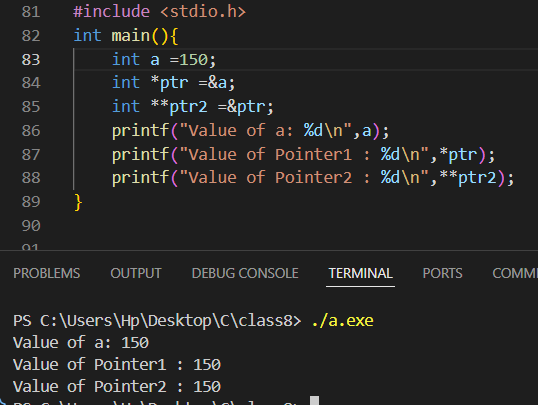
1. **Create an example that illustrates what happens when you attempt to dereference a wild pointer (a pointer that has not been initialized). Document the output and explain why this leads to undefined behavior.**

A screen shot of a computer

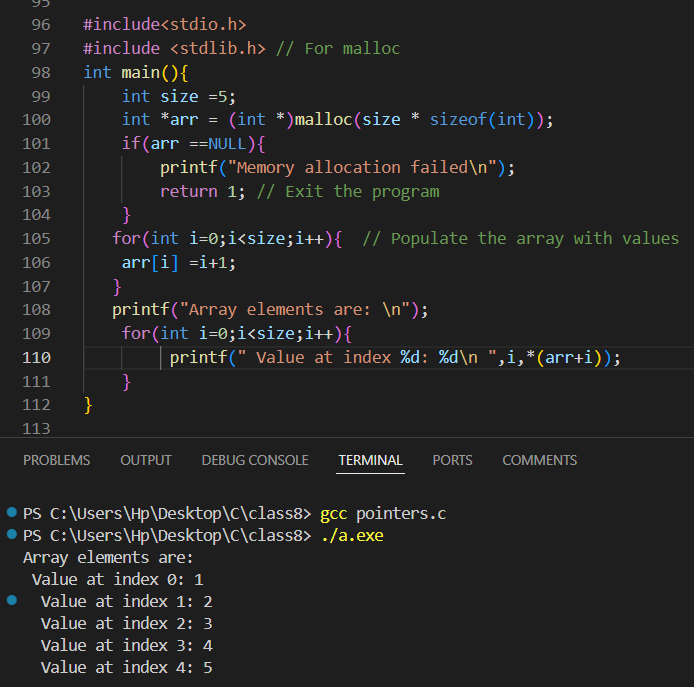
Description automatically generated

* The pointer ptr is uninitialized, making it a wild pointer containing a random memory address.
* Dereferencing it (\*ptr) attempts to access memory at that address, leading to undefined behavior.
* Output may vary (garbage value or crash) because the memory location may be invalid or contain random data.
* Undefined behavior is dangerous as it causes program instability, crashes, or security vulnerabilities.
* Always initialize pointers to NULL or a valid address before dereferencing.

1. **Implement a C program that uses a pointer to a pointer. Initialize an integer variable, create a pointer that points to it, and then create another pointer that points to the first pointer. Print the value using both levels of indirection.**



1. **Write a program that dynamically allocates memory for an array of integers using malloc. Populate the array with values, print them using pointers, and then free the allocated memory.**

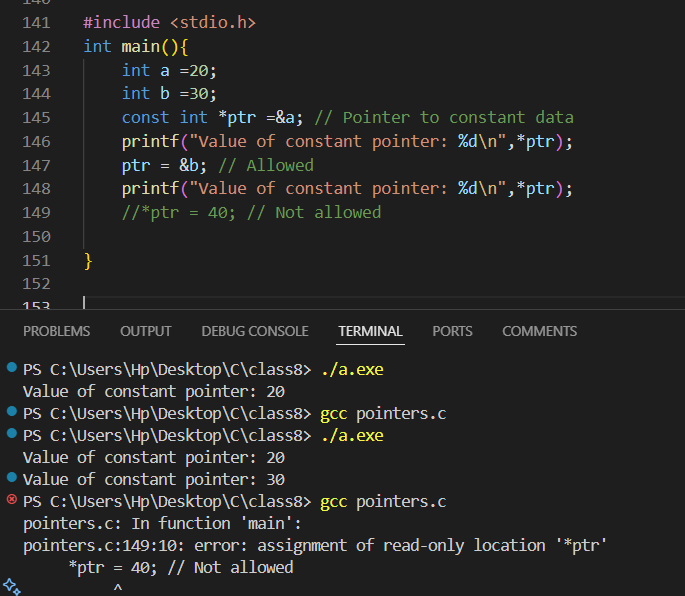


1. **Define a function that takes two integers as parameters and returns their sum. Then, create a function pointer that points to this function and use it to call the function with different integer values.**

A screen shot of a computer program

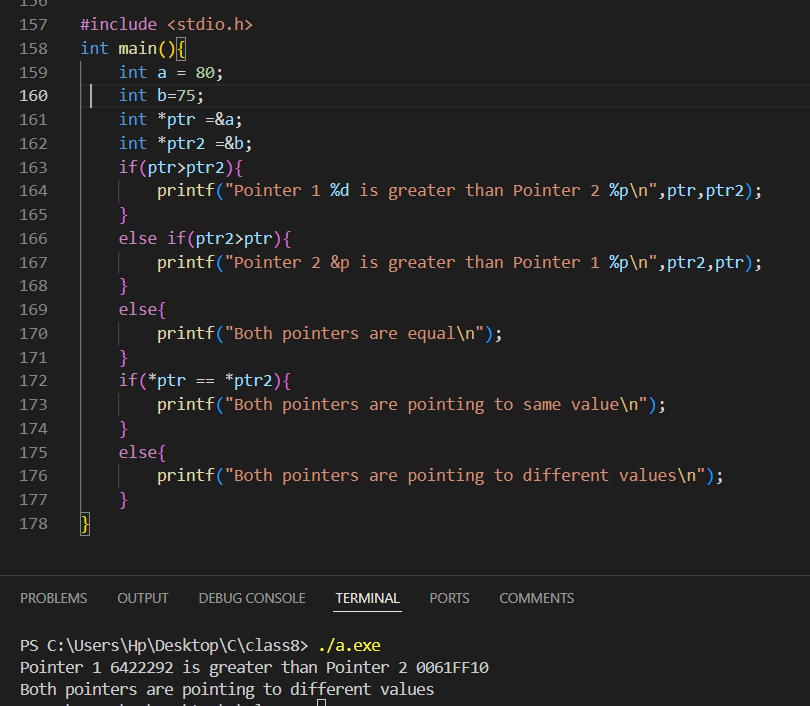
Description automatically generated

1. **Create two examples: one demonstrating a constant pointer (where you cannot change what it points to) and another demonstrating a pointer to constant data (where you cannot change the data being pointed to). Document your findings.**



The pointer itself is constant and cannot be reassigned to point to another address, but the data it points to can be modified.

1. **Write a program that compares two pointers pointing to different variables of the same type. Use relational operators to determine if one pointer points to an address greater than or less than another and print the results.has context menu**

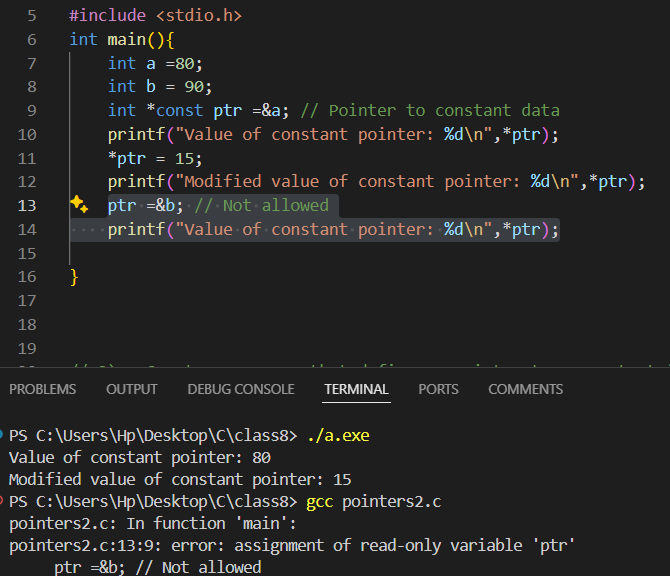


Pointer comparison does not compare values; it compares the addresses where the variables are stored in memory.

The result can vary depending on the compiler, memory allocation, and system architecture.

**Problem Statements**

1. **Write a program that declares a constant pointer to an integer. Initialize it with the address of an integer variable and demonstrate that you can change the value of the integer but cannot reassign the pointer to point to another variable.**



1. **Create a program that defines a pointer to a constant integer. Attempt to modify the value pointed to by this pointer and observe the compiler's response.**

A screen shot of a computer program

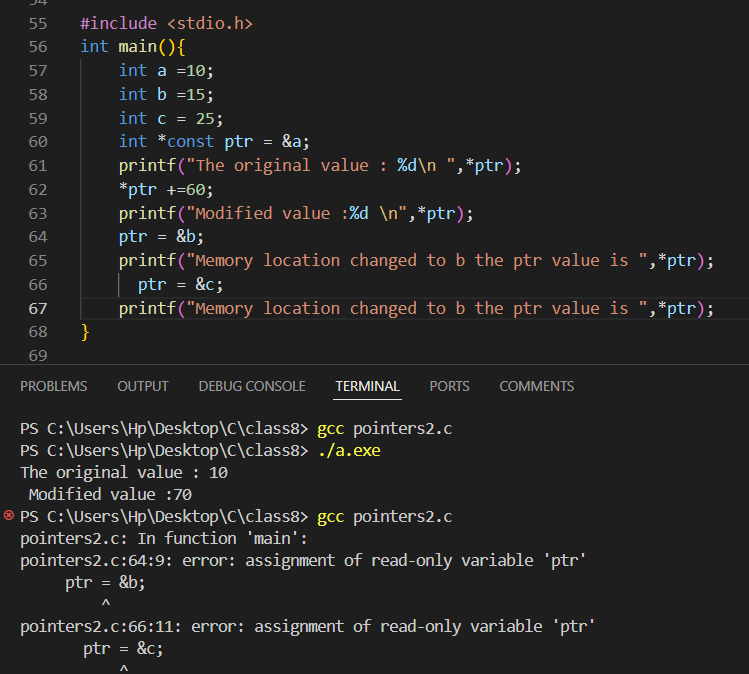
Description automatically generated

1. **Implement a program that declares a constant pointer to a constant integer. Show that neither the address stored in the pointer nor the value it points to can be changed.**

A screen shot of a computer program

Description automatically generated

1. **Develop a program that uses a constant pointer to iterate over multiple integers stored in separate variables. Show how you can modify their values through dereferencing while keeping the pointer itself constant.**



1. **Implement a program that uses pointers and decision-making statements to check if two constant integers are equal or not, printing an appropriate message based on the comparison.**

A screen shot of a computer program

Description automatically generated

1. **Create a program that uses conditional statements to determine if a constant pointer is pointing to a specific value, printing messages based on whether it matches or not.**

A screen shot of a computer program

Description automatically generated

1. **Write a program that declares two constant pointers pointing to different integer variables. Compare their addresses using relational operators and print whether one points to a higher or lower address than the other.**

A screen shot of a computer program

Description automatically generated

1. **Implement a program that uses a constant pointer within loops to iterate through multiple variables (not stored in arrays) and print their values.**

A screen shot of a computer program

Description automatically generated

1. **Develop a program that uses a constant pointer to iterate over several integer variables (not in an array) using pointer arithmetic while keeping the pointer itself constant.**

A screen shot of a computer program

Description automatically generated

**1. Machine Efficiency Calculation**

**Requirements:**

* **Input: Machine's input power and output power as floats.**
* **Output: Efficiency as a float.**
* **Function: Accepts pointers to input power and output power, calculates efficiency, and updates the result via a pointer.**
* **Constraints: Efficiency = (Output Power / Input Power) \* 100.**

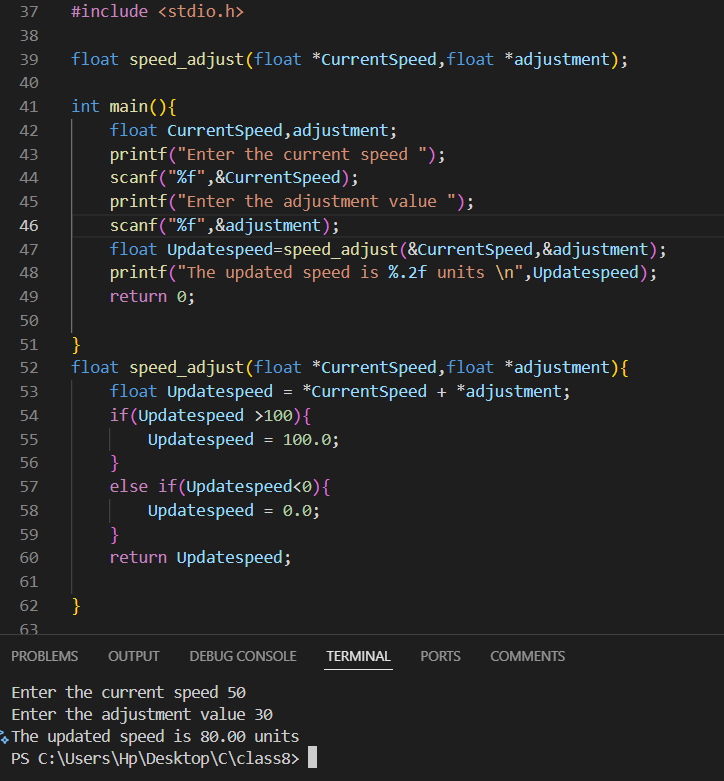
A screen shot of a computer program

Description automatically generated

**2. Conveyor Belt Speed Adjustment**

**Requirements:**

* **Input: Current speed (float) and adjustment value (float).**
* **Output: Updated speed.**
* **Function: Uses pointers to adjust the speed dynamically.**
* **Constraints: Ensure speed remains within the allowable range (0 to 100 units).**



**3. Inventory Management**

**Requirements:**

* **Input: Current inventory levels of raw materials (array of integers).**
* **Output: Updated inventory levels.**
* **Function: Accepts a pointer to the inventory array and modifies values based on production or consumption.**
* **Constraints: No inventory level should drop below zero.**

A screen shot of a computer program

Description automatically generated

**4. Robotic Arm Positioning**

**Requirements:**

* **Input: Current x, y, z coordinates (integers) and movement delta values.**
* **Output: Updated coordinates.**
* **Function: Takes pointers to x, y, z and updates them based on delta values.**
* **Constraints: Validate that the coordinates stay within the workspace boundaries.**

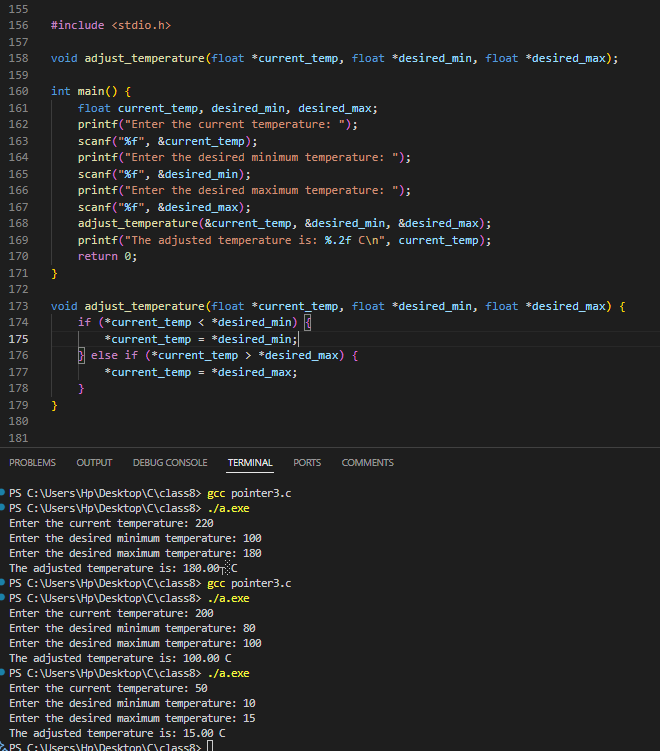
A screen shot of a computer program

Description automatically generated

**5. Temperature Control in Furnace**

**Requirements:**

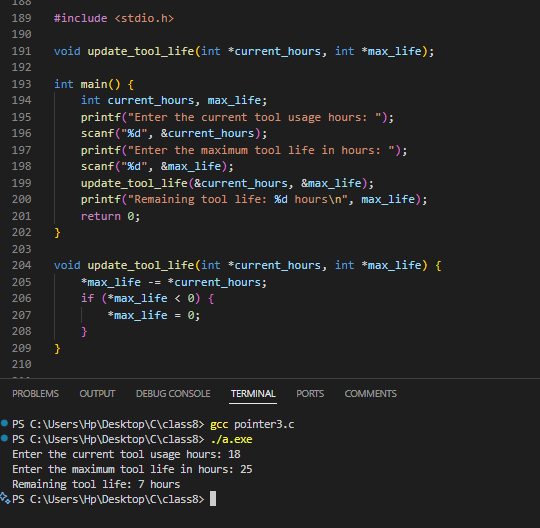
* **Input: Current temperature (float) and desired range.**
* **Output: Adjusted temperature.**
* **Function: Uses pointers to adjust temperature within the range.**
* **Constraints: Temperature adjustments must not exceed safety limits.**



**6. Tool Life Tracker**

**Requirements:**

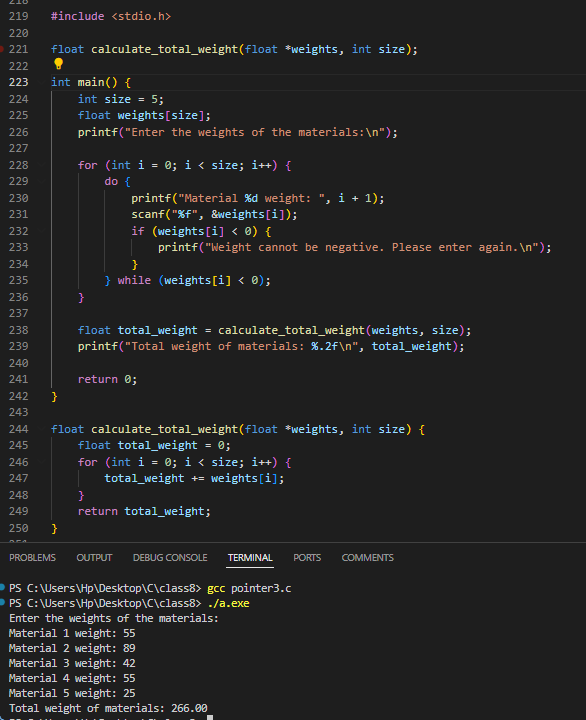
* **Input: Current tool usage hours (integer) and maximum life span.**
* **Output: Updated remaining life (integer).**
* **Function: Updates remaining life using pointers.**
* **Constraints: Remaining life cannot go below zero.**



**7. Material Weight Calculator**

**Requirements:**

* **Input: Weights of materials (array of floats).**
* **Output: Total weight (float).**
* **Function: Accepts a pointer to the array and calculates the sum of weights.**
* **Constraints: Ensure no negative weights are input.**



**8. Welding Machine Configuration**

**Requirements:**

* **Input: Voltage (float) and current (float).**
* **Output: Updated machine configuration.**
* **Function: Accepts pointers to voltage and current and modifies their values.**
* **Constraints: Validate that voltage and current stay within specified operating ranges.**

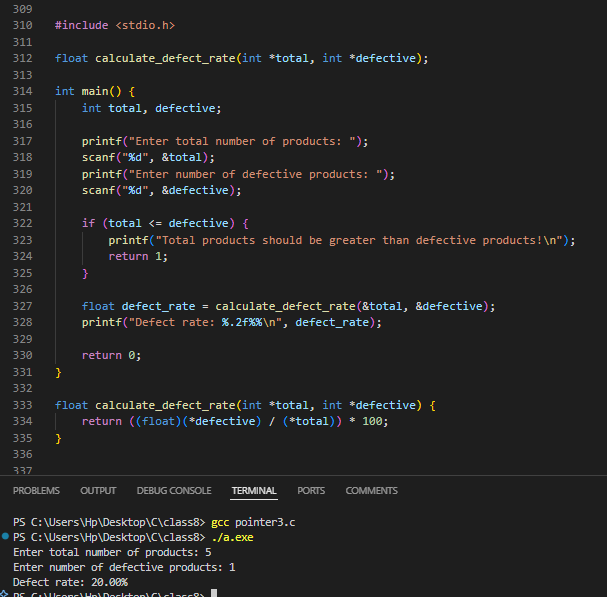
A screen shot of a computer program

Description automatically generated

**9. Defect Rate Analyzer**

**Requirements:**

* **Input: Total products and defective products (integers).**
* **Output: Defect rate (float).**
* **Function: Uses pointers to calculate defect rate = (Defective / Total) \* 100.**
* **Constraints: Ensure total products > defective products.**



**10. Assembly Line Optimization**

**Requirements:**

* **Input: Timing intervals between stations (array of floats).**
* **Output: Adjusted timing intervals.**
* **Function: Modifies the array values using pointers.**
* **Constraints: Timing intervals must remain positive.**

A screen shot of a computer program

Description automatically generated

**11. CNC Machine Coordinates**

**Requirements:**

* **Input: Current x, y, z coordinates (floats).**
* **Output: Updated coordinates.**
* **Function: Accepts pointers to x, y, z values and updates them.**
* **Constraints: Ensure updated coordinates remain within machine limits.**

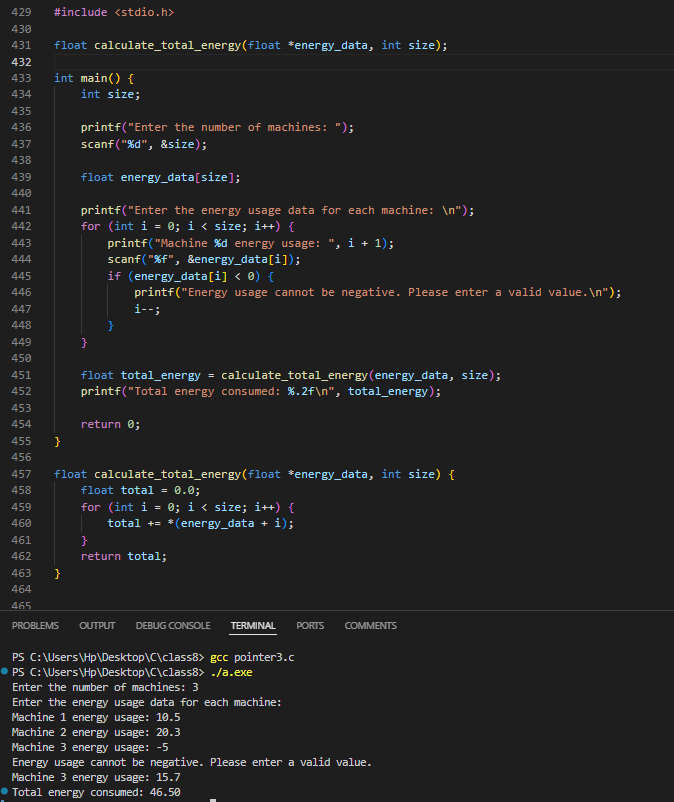
A screen shot of a computer program

Description automatically generated

**12. Energy Consumption Tracker**

**Requirements:**

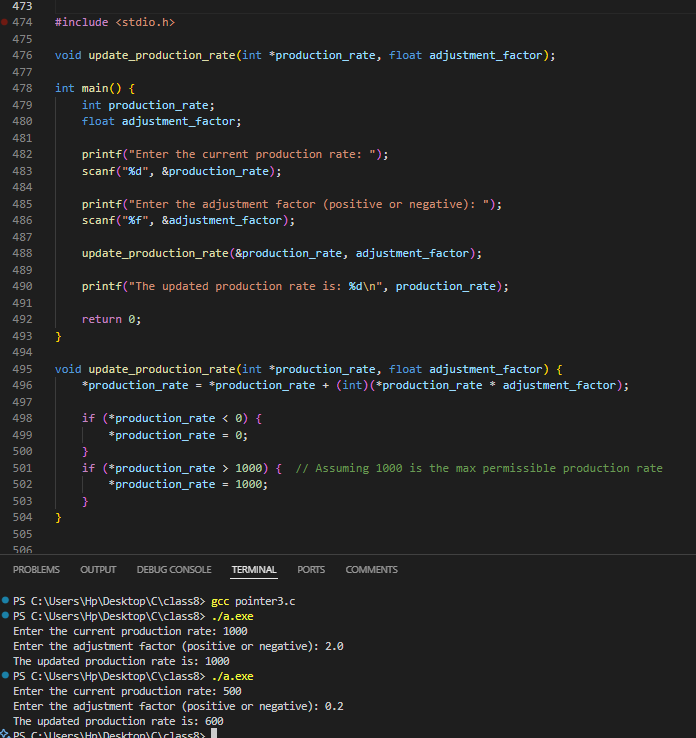
* **Input: Energy usage data for machines (array of floats).**
* **Output: Total energy consumed (float).**
* **Function: Calculates and updates total energy using pointers.**
* **Constraints: Validate that no energy usage value is negative.**



**13. Production Rate Monitor**

**Requirements:**

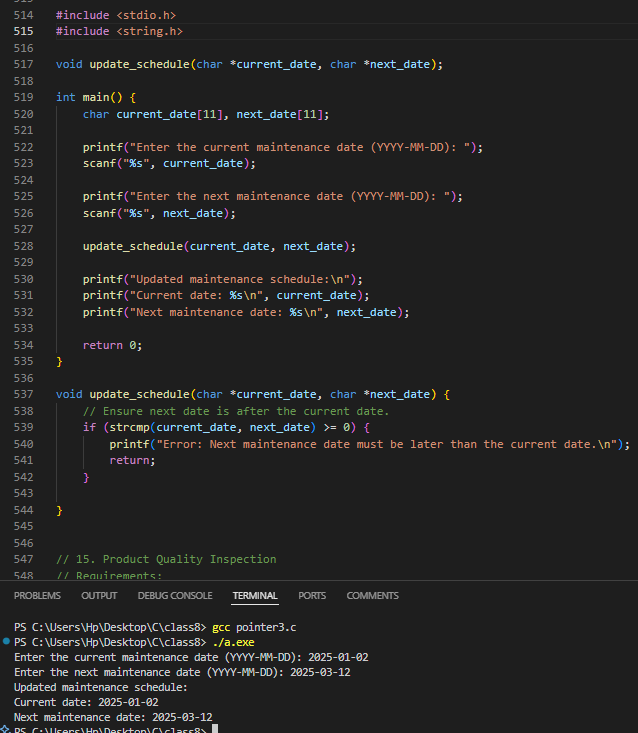
* **Input: Current production rate (integer) and adjustment factor.**
* **Output: Updated production rate.**
* **Function: Modifies the production rate via a pointer.**
* **Constraints: Production rate must be within permissible limits.**



**14. Maintenance Schedule Update**

**Requirements:**

* **Input: Current and next maintenance dates (string).**
* **Output: Updated maintenance schedule.**
* **Function: Accepts pointers to the dates and modifies them.**
* **Constraints: Ensure next maintenance date is always later than the current date.**



**15. Product Quality Inspection**

**Requirements:**

* **Input: Quality score (integer) for each product in a batch.**
* **Output: Updated quality metrics.**
* **Function: Updates quality metrics using pointers.**
* **Constraints: Ensure quality scores remain within 0-100.**

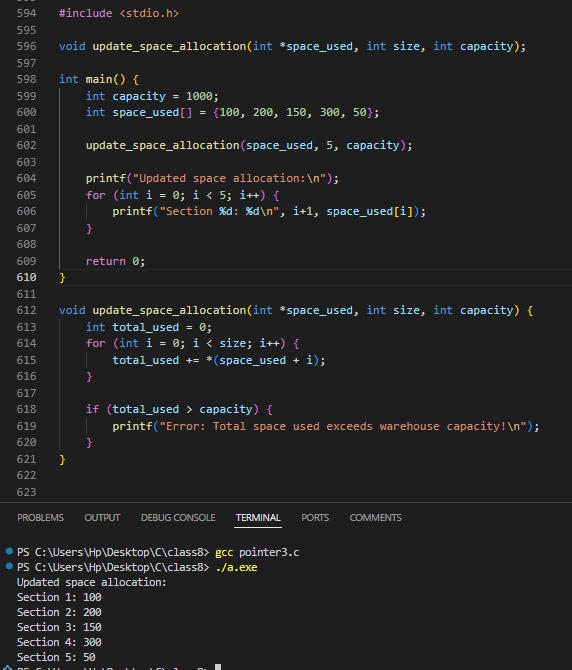
A screen shot of a computer program

Description automatically generated

**16. Warehouse Space Allocation**

**Requirements:**

* **Input: Space used for each section (array of integers).**
* **Output: Updated space allocation.**
* **Function: Adjusts space allocation using pointers.**
* **Constraints: Ensure total space used does not exceed warehouse capacity.**



**17. Packaging Machine Settings**

**Requirements:**

* **Input: Machine settings like speed (float) and wrap tension (float).**
* **Output: Updated settings.**
* **Function: Modifies settings via pointers.**
* **Constraints: Validate settings remain within safe operating limits.**

A screen shot of a computer program

Description automatically generated

**18. Process Temperature Control**

**Requirements:**

* **Input: Current temperature (float).**
* **Output: Adjusted temperature.**
* **Function: Adjusts temperature using pointers.**
* **Constraints: Temperature must stay within a specified range.**

A screen shot of a computer program

Description automatically generated

**19. Scrap Material Management**

**Requirements:**

* **Input: Scrap count for different materials (array of integers).**
* **Output: Updated scrap count.**
* **Function: Modifies the scrap count via pointers.**
* **Constraints: Ensure scrap count remains non-negative.**

A screen shot of a computer program

Description automatically generated

**20. Shift Performance Analysis**

**Requirements:**

* **Input: Production data for each shift (array of integers).**
* **Output: Updated performance metrics.**
* **Function: Calculates and updates overall performance using pointers.**
* **Constraints: Validate data inputs before calculations.**

A screenshot of a computer program

Description automatically generated