BlockWorks: Building the Future One Block at a Time.

LAUNCHING INTO COMPUTER SCIENCE

TABLE OF CONTENTS

Development 3
Overview 3
Features 3
Development 3
User Interface: 3
Viewing Blocks 5
Searching Blocks: 6
Adding Blocks 7
Deleting Blocks: 8
Sorting Blocks: 9
Data Validation: 10
Testing 12
Add Block 12
Delete Block 16
Sort - Insertion 20
Search - Serial 25
Evaluation 31
Requirements 31
Limitations and Developments 31
References 32

BLOCKWORKS DEVELOPMENT

DEVELOPMENT

OVERVIEW

BlockWorks is a Python application designed to manage and track blocks and their associated components. The system provides a robust, user-friendly interface for creating, managing and organising block data. It enables users to perform essential operations such as adding new blocks and searching through existing blocks.

FEATURES

View All Blocks: Displays all blocks with their components and quantities.

Add New Blocks: Add new blocks by specifying the name and components, and their quantities.

Delete Blocks: Remove blocks with a confirmation prompt to avoid accidental deletion.

Search Blocks: Find blocks by name or component.

Sort Blocks: Sort blocks alphabetically in ascending or descending order.

Confirmation Prompts: Confirm critical actions like adding or deleting blocks to prevent mistakes.

DEVELOPMENT

USER INTERFACE:

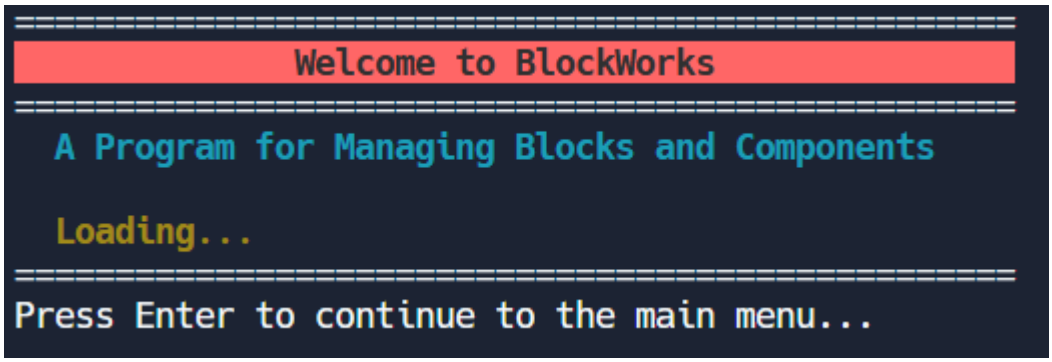
The BlockWorks interface is a simple, menu-based system. According to Nielsen (1993), a menu-based interfaces simplify the user experience by offering a straightforward and organised workflow. It works best when run in a console. It includes:

Splash Screen: Provides a welcome screen with a brief introduction. This screen sets the tone for the user experience. Pressing Enter moves the user into the system's main functionality.

Main Menu: The central hub with numbered options.1-View All Blocks, 2-Search Blocks, 3-Add Blocks, 4-Delete Blocks, 5-Sort Blocks and 6-Exit.

Action Specific Screens: Each option provides a clear guide, displays results and requests confirmations. This ensures that users are always aware of their actions improving the overall user experience.

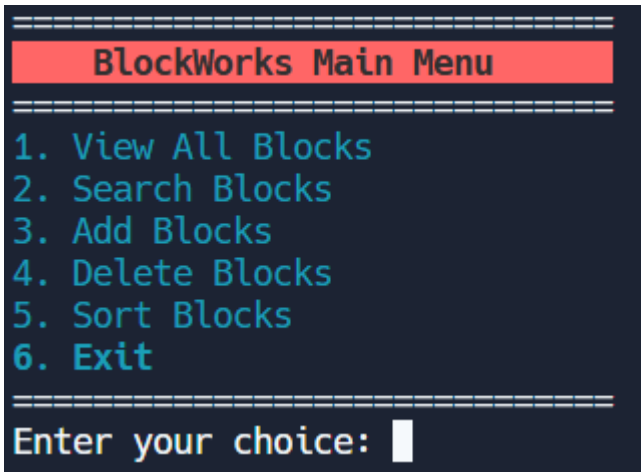
```
def splash_screen():
    """
    Displays the splash screen when the program starts.
    Shows a welcome message and some loading information.
    """
    clear_console()
    print("=" * 50)
    print("\033[1;37;41m              Welcome to BlockWorks               \033[0m")
    print("=" * 50)
    print("\033[1;36m  A Program for Managing Blocks and Components  \033[0m")
    print("\n\033[1;33m  Loading...\033[0m")
    print("=" * 50)
    input("Press Enter to continue to the main menu...")
```




```
def display_menu():
    """
    Displays the main menu to the user, where they can choose
    what action to perform in the program.
    """
    print("=" * 30)
    print("\033[1;37;41m    BlockWorks Main Menu      \033[0m")
    print("=" * 30)

    print("\033[36m1. View All Blocks\033[0m")
    print("\033[36m2. Search Blocks\033[0m")
    print("\033[36m3. Add Blocks\033[0m")
    print("\033[36m4. Delete Blocks\033[0m")
    print("\033[36m5. Sort Blocks\033[0m")
    print("\033[1;36m6. Exit\033[0m")

    print("=" * 30)
```



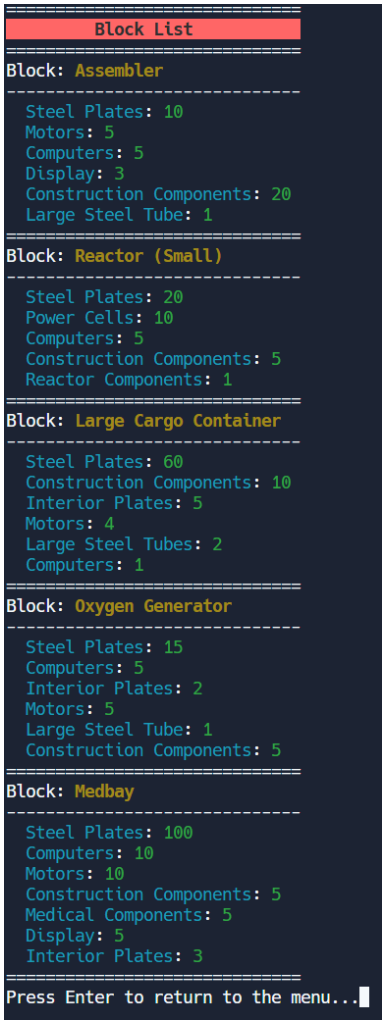
VIEWING BLOCKS

This feature displays a comprehensive list of all blocks in the system. If no blocks are available a message will display. The block data is stored in a list of dictionaries, each dictionary represents a block with the block's name as a key and its components stored as nested dictionaries. Storing data in this format offers clarity and flexibility. This structure ensures efficient memory usage and facilitates scalability.

```
def display_blocks(blocks):
    """
    Displays a list of all blocks and their components.
    If no blocks are available, shows a message.
    """
    clear_console()

    print("=" * 30)
    print("\033[1;37;41m         Block List           \033[0m")
    print("=" * 30)

    if not blocks:
      print("No blocks available.")
    else:
      for block in blocks:
        print(f"Block: \033[1;33m{block['name']}\033[0m")
        print("-" * 30)
        for component, quantity in block['components'].items():
            print(f"  \033[36m{component}\033[0m: \033[32m{quantity}\033[0m")
        print("=" * 30)
```



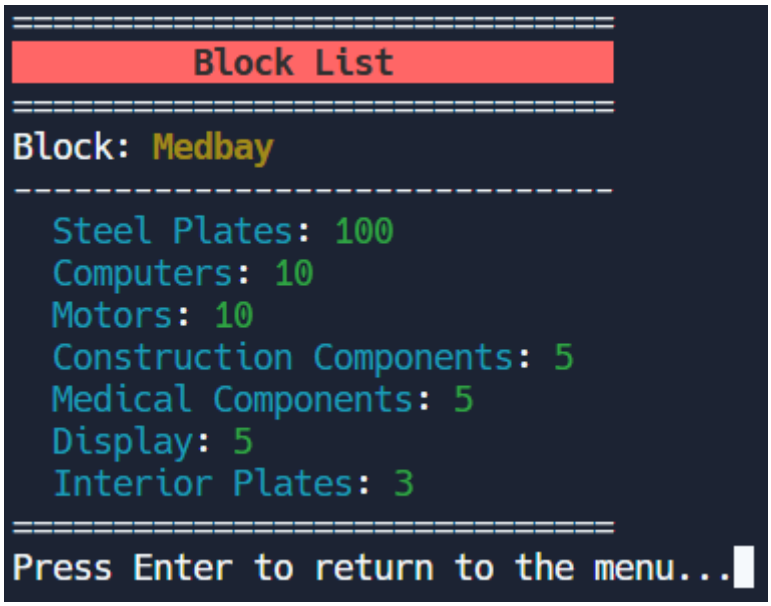
SEARCHING BLOCKS:

The search functionality enables users to search for blocks by their name. Users enter a search term and the program filters and displays blocks that match the term. To facilitate partial name searches the search algorithm was changed from binary to serial search. This change allows the program to find matches even when only part of the block name is entered. If no matches are found, the user is notified.

```
def search_block(blocks):
    """
    Prompts the user to enter a search term and displays blocks
    whose names match the search term.
    """
    search_term = input("Enter the name of the block to search: ").lower()
    found_blocks = []

    for block in blocks:
        if search_term in block['name'].lower():
            found_blocks.append(block)

    if found_blocks:
        display_blocks(found_blocks)
        input("Press Enter to return to the menu...")
    else:
        input("No matching blocks found. Press Enter to return to the menu...")
```



ADDING BLOCKS

This allows users to create a new block by providing its name and specifying the components. Users are prompted to enter quantities for each component, with input validated to ensure that quantities are positive numbers. The program prevents duplication. Users confirm whether they wish to add the block to the system. If confirmed, the block is added to the list.

```
def add_block(blocks):
    """
    Prompts the user to add a new block by entering its name and
    components. Validates input before adding the block to the list.
    """
    available_components = [
        "Steel Plates", "Motors", "Computers", "Display",
        "Construction Components", "Large Steel Tube", "Small Steel Tube", "Power Cells",
        "Reactor Components", "Interior Plates", "Medical Components"
    ]

    block_name = input("Enter the name of the new block: ").strip()

    if not block_name:
        print("Block name cannot be empty. Please try again.")
        input("Press Enter to return to the menu...")
        return blocks

    if any(block['name'].lower() == block_name.lower() for block in blocks):
        print(f"A block with the name '{block_name}' already exists. Please enter a unique
name.")
        input("Press Enter to return to the menu...")
        return blocks

    components = {}

    for component in available_components:
        while True:
            try:
                quantity = int(input(f"Enter the quantity for {component}: ").strip())

                if quantity < 0:
                    print("Quantity cannot be negative. Please try again.")
                    continue

                if quantity > 0:
                    components[component] = quantity
                break
            except ValueError:
                print("Please enter a valid number for the quantity.")

    if not components:
        print("No components entered for this block. The block will not be added.")
        input("Press Enter to return to the menu...")
        return blocks

    new_block = {
        "name": block_name,
        "components": components
    }

    display_blocks([new_block])

    if confirm_action("Do you want to add this block to the list? (yes/no): "):
        blocks.append(new_block)
        print(f"Block '{block_name}' has been added successfully.")
    else:
        print("Block was not added.")

    input("Press Enter to return to the menu...")
    return blocks
```

DELETING BLOCKS:

To delete a block by the block's name. The user must confirm the deletion. The block is removed if confirmed or the action is cancelled.

```
def delete_block(blocks):
    """
    Prompts the user to delete a block by its name. Confirms
    before deleting the block from the list.
    """
    block_name = input("Enter the exact name of the block to delete: ").strip()

    if not block_name:
        print("Block name cannot be empty. Please try again.")
        input("Press Enter to return to the menu...")
        return blocks

    found_block = None
    for block in blocks:
        if block_name.strip().lower() == block['name'].lower():
            found_block = block
            break

    if found_block:
        display_blocks([found_block])

        if confirm_action(f"Are you sure you want to delete the block
'{found_block['name']}'? (yes/no): "):
            blocks.remove(found_block)
            print(f"Block '{found_block['name']}' has been deleted.")
        else:
            print("Block not deleted.")
    else:
        print(f"No block found with the exact name '{block_name}'.")

    input("Press Enter to return to the menu...")
    return blocks
```

SORTING BLOCKS:

The sorting options allow the user to arrange blocks alphabetically by name, either ascending or descending order. The blocks are displayed accordingly after selecting the preferred sorting order. The sorting is performed using an insertion sort algorithm. This method was chosen for its simplicity and efficiency with small datasets.

```
def sort_blocks(blocks):
    """
    Prompts the user to choose between sorting blocks in ascending
    (A-Z) or descending (Z-A) order and then sorts the blocks accordingly.
    """
    print("Choose sort order:")
    print("1. Ascending (A-Z)")
    print("2. Descending (Z-A)")

    choice = input("Enter your choice: ").strip()

    if choice == "1":
        blocks = insertion_sort(blocks, ascending=True)
    elif choice == "2":
        blocks = insertion_sort(blocks, ascending=False)
    else:
        input("Invalid choice. Press Enter to return to the menu...")
        return

    display_blocks(blocks)
    input("Press Enter to return to the menu...")
```

```
def insertion_sort(blocks, ascending=True):
    """
    Sorts the blocks alphabetically by their name using the
    insertion sort algorithm. Can sort in ascending or descending order.
    """
    for i in range(1, len(blocks)):
        key = blocks[i]
        j = i - 1
        while (
            j >= 0
            and (
                (key['name'].lower() < blocks[j]['name'].lower() and ascending)
                or
                (key['name'].lower() > blocks[j]['name'].lower() and not ascending)
            )
        ):
            blocks[j + 1] = blocks[j]
            j -= 1
        blocks[j + 1] = key
    return blocks
```

DATA VALIDATION:

The program ensures users enter valid data by prompting for quantities and checking that the input is a positive integer. If the input is invalid, such as a negative number or non-numeric input, the program asks the user to try again until a valid number is provided.

```
def get_user_choice():
    """
    Prompts the user to select a choice from the menu and
    validates the input to ensure it's a valid choice.
    """
    while True:
        try:
            choice = int(input("Enter your choice: "))
            if 1 <= choice <= 6:
                return choice
            else:
                print("Invalid choice. Please choose a number between 1 and 6.")
        except ValueError:
            print("Invalid input. Please enter a number.")
```

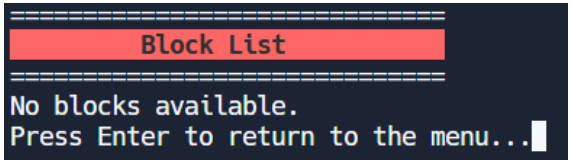
```
def confirm_action(prompt="Are you sure? (yes/no): "):
    """
    Asks the user for confirmation before performing an action.
    Returns True for 'yes' and False for 'no', ensures valid input.
    """
    while True:
        user_input = input(prompt).strip().lower()
        if user_input in ["yes", "y"]:
            return True
        elif user_input in ["no", "n"]:
            return False
        else:
            print("Invalid input. Please enter 'yes' or 'no'.")
```

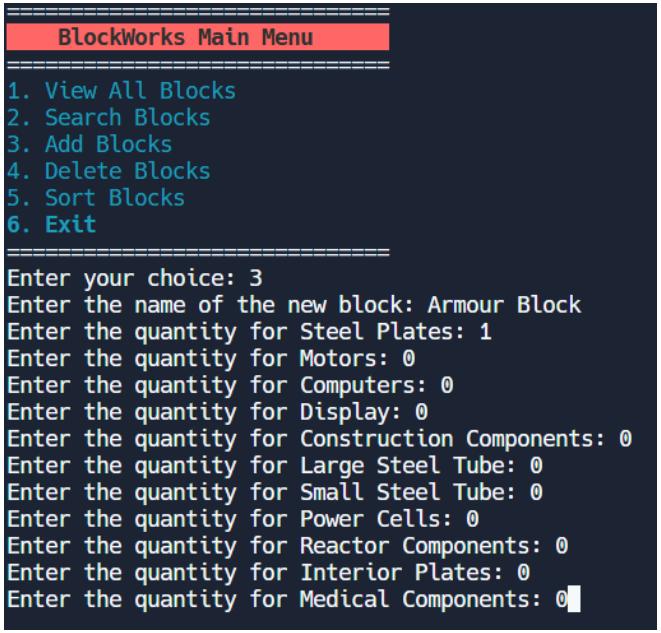
BLOCKWORKS TESTING

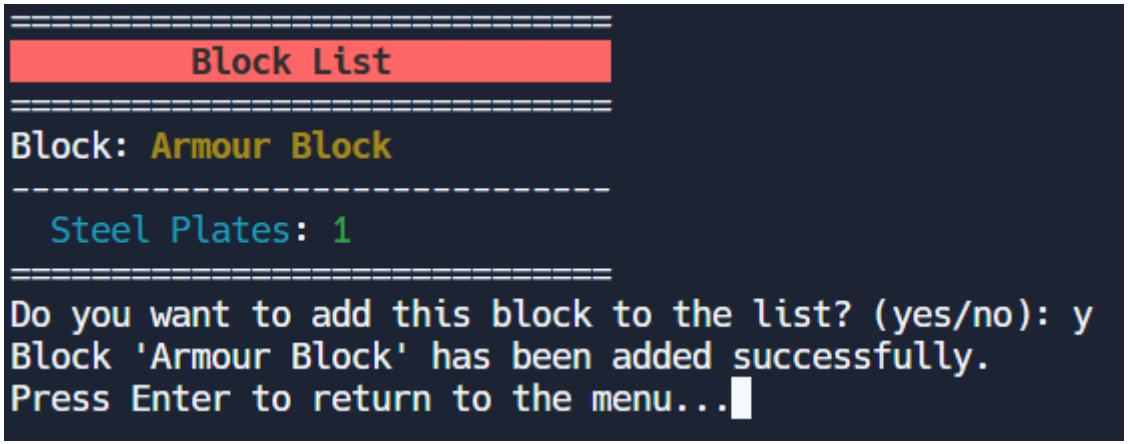
TESTING

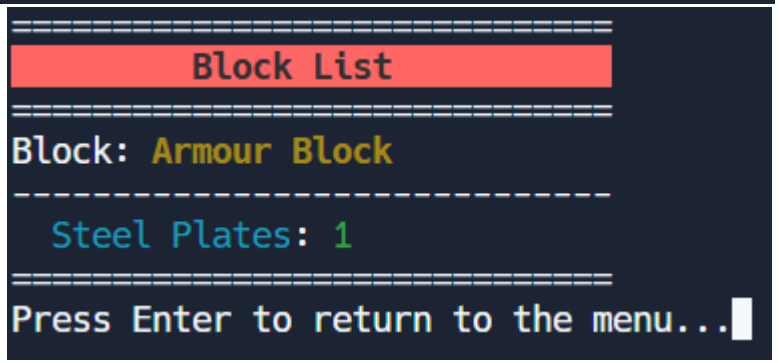
ADD BLOCK

Test	Description	Data	Expected Outcome	Actual Outcome
A1	Test adding a new block to an empty list	List = [] Block Name = "Armour Block" Components = {"Steel Plate": 1}	The new block is added to the list.	The new block is added to the list.

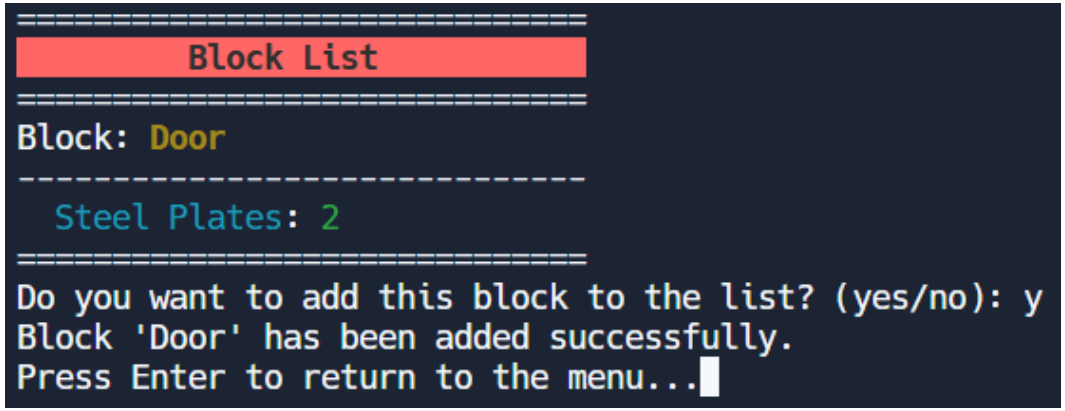


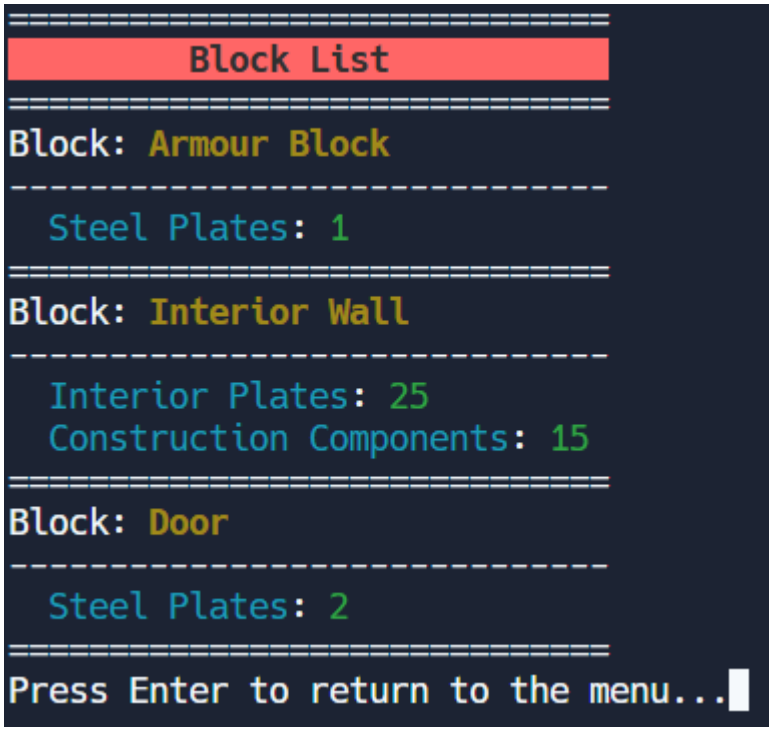




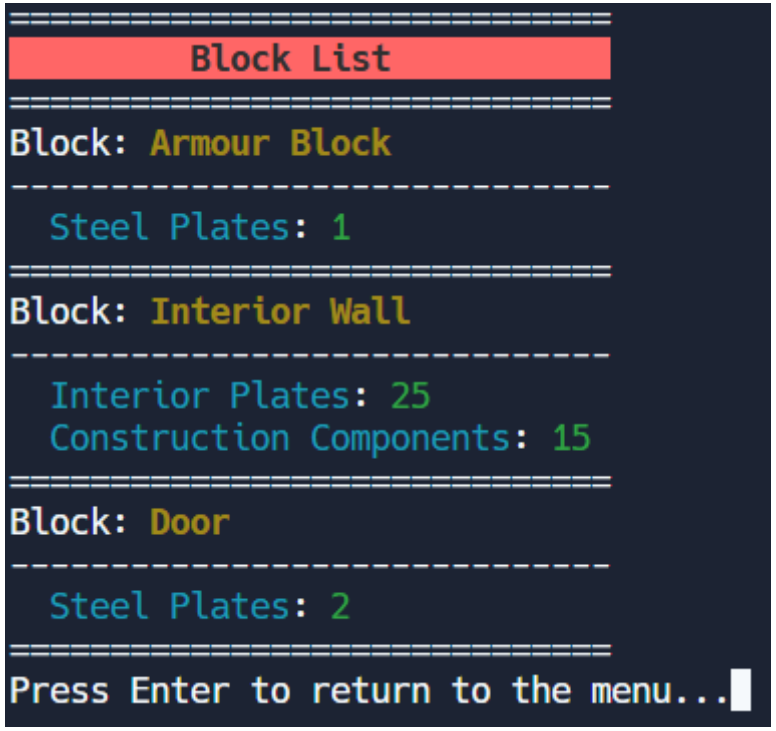


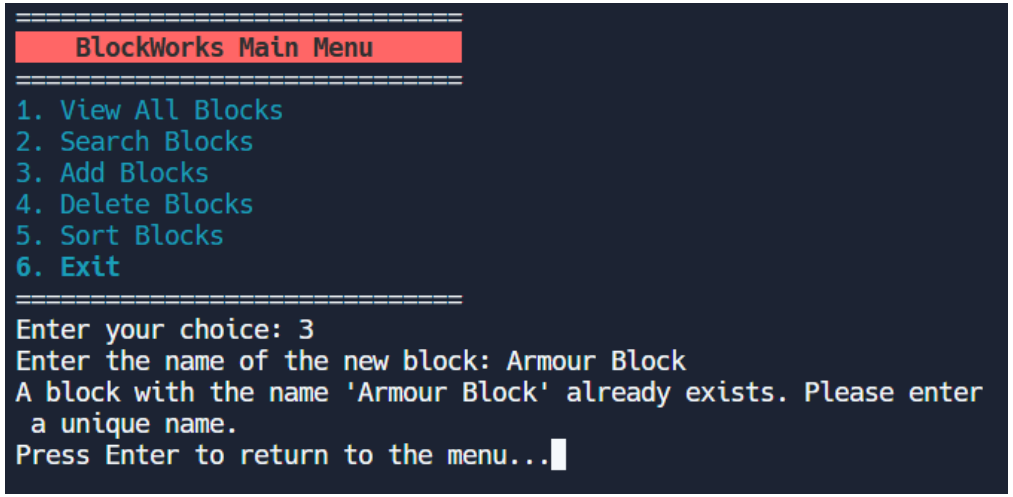
Test	Description	Data	Expected Outcome	Actual Outcome
A2	Test with a list containing blocks and a unique block name.	List = ["Armour Block", "Interior Wall"] Block Name = "Door" Components = {"Steel Plate": 2}	The new block is added to the list.	The new block is added to the list.



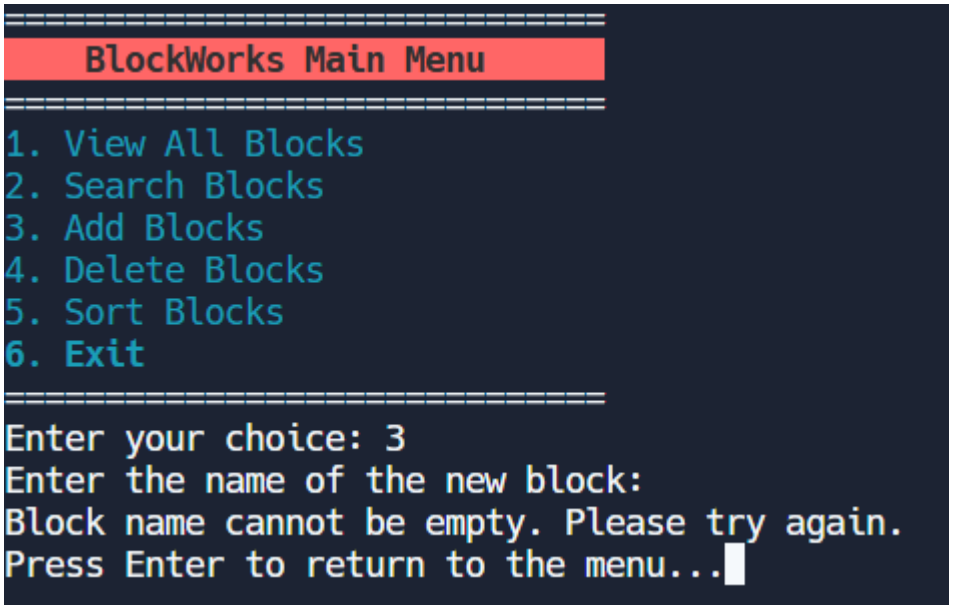


Test	Description	Data	Expected Outcome	Actual Outcome
A3	Test with a list containing blocks and a duplicate block name.	List = ["Armour Block", "Interior Wall", "Door"] Block Name = "Armour Block" Components = {"Steel Plate": 2}	The function prevents adding the block and notifies the user that the block name already exists.	The function prevents adding the block and notifies the user that the block name already exists.

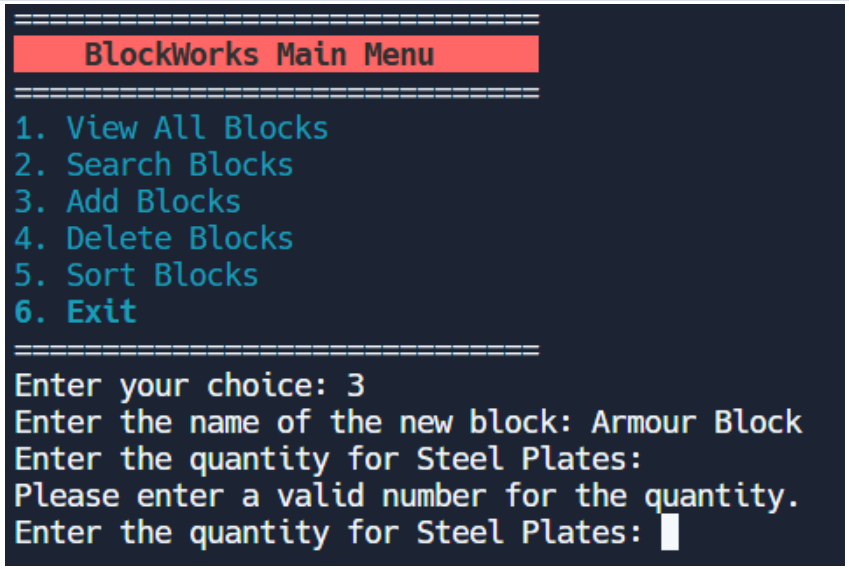


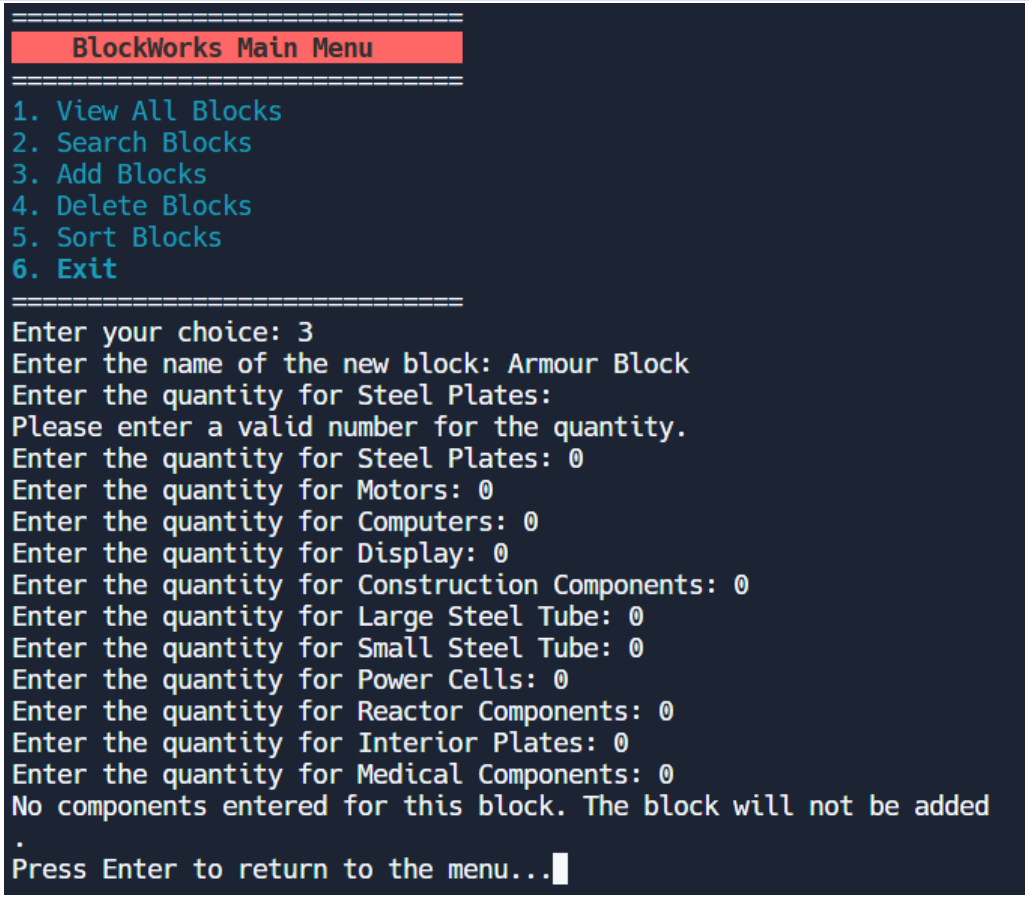


Test	Description	Data	Expected Outcome	Actual Outcome
A4	Test with an invalid or empty block name.	List = [], Block Name = "" Components = {"Steel Plate": 1}	The function notifies the user that the block name cannot be empty or invalid.	The function notifies the user that the block name cannot be empty or invalid.



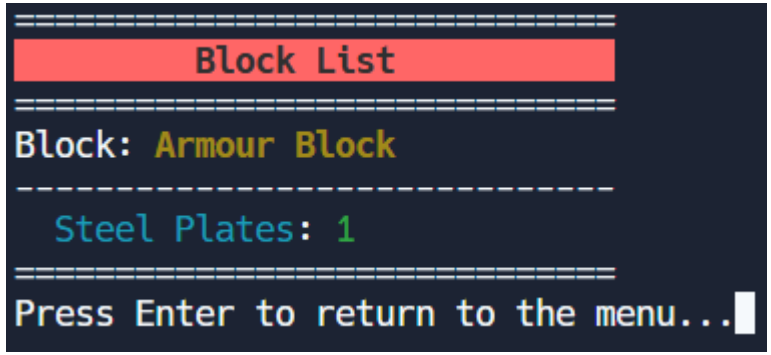
Test	Description	Data	Expected Outcome	Actual Outcome
A5	Test with no components entered.	List = [] Block Name = "Armour Block" Components = {}	The function does not add the block and notifies the user that no components were entered.	The function does not add the block and notifies the user that no components were entered.

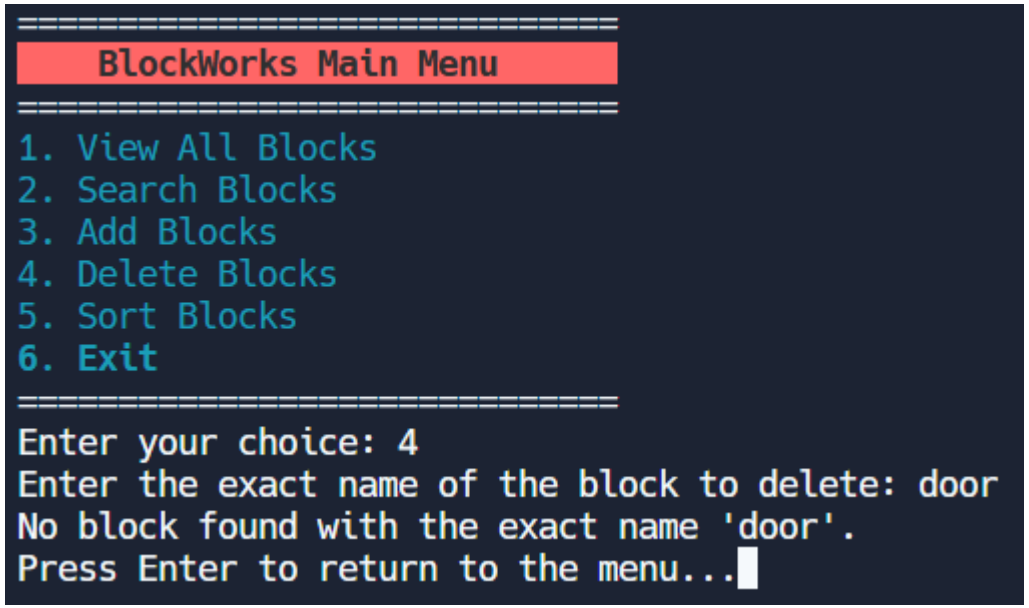




DELETE BLOCK

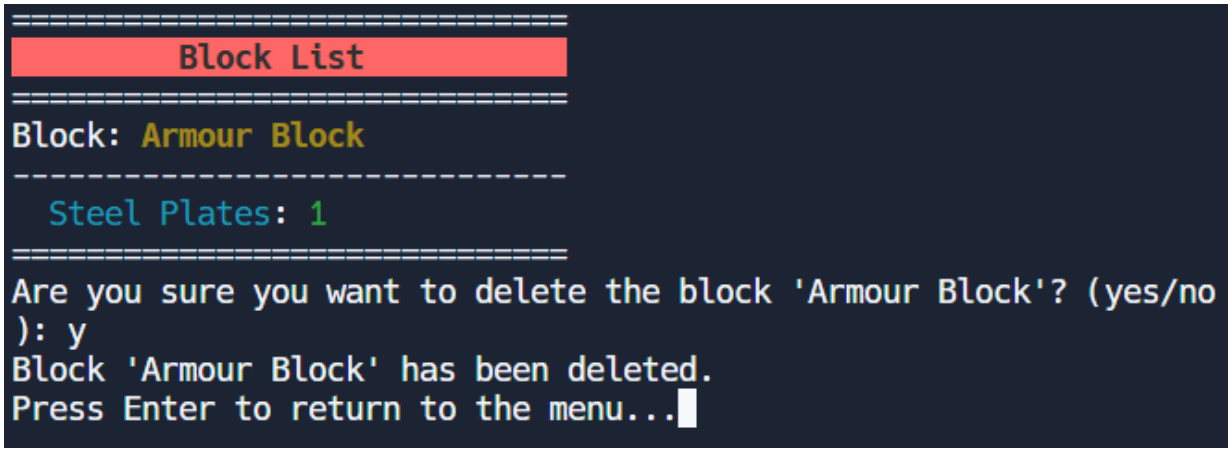
Test	Description	Data	Expected Outcome	Actual Outcome
D1	Test with a non-existent block name.	List = ["Armour Block"] Block Name = "Door"	The function informs the user that there are no blocks to delete.	The function informs the user that there are no blocks to delete.

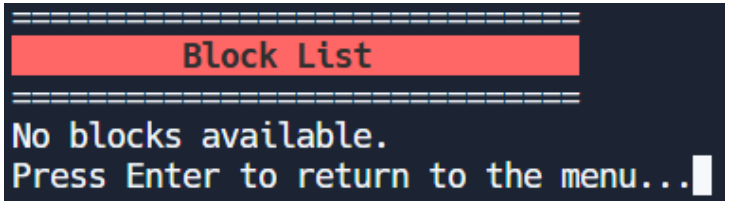




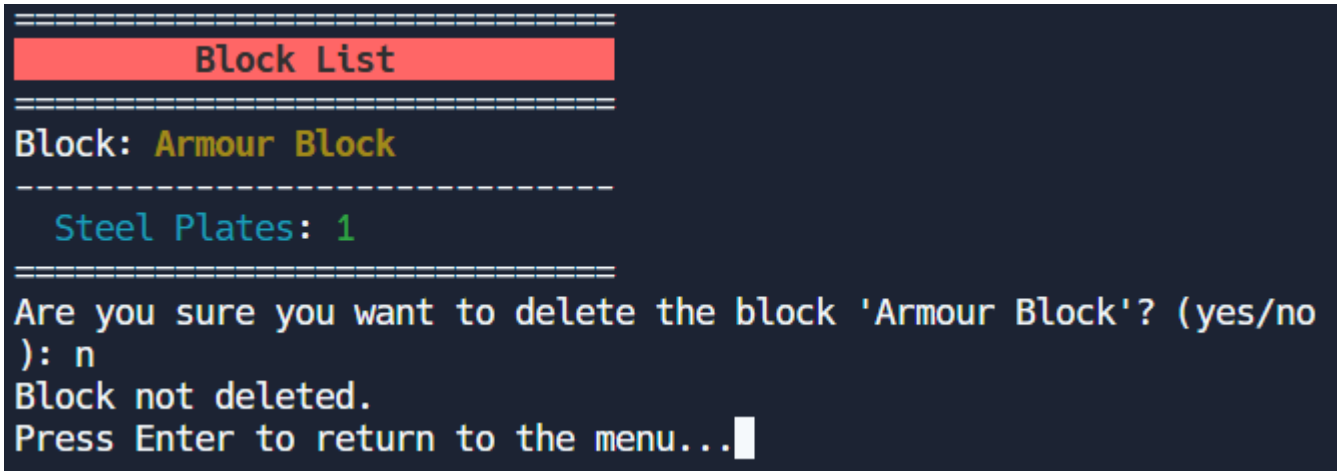
Test D2 replaced with confirmation entry

Test	Description	Data	Expected Outcome	Actual Outcome
D2	Test with multiple blocks and a matching block name.	List = ["Armour Block"] Block Name = "Armour Block"	The block is deleted from the list after user confirmation.	The block is not deletes.



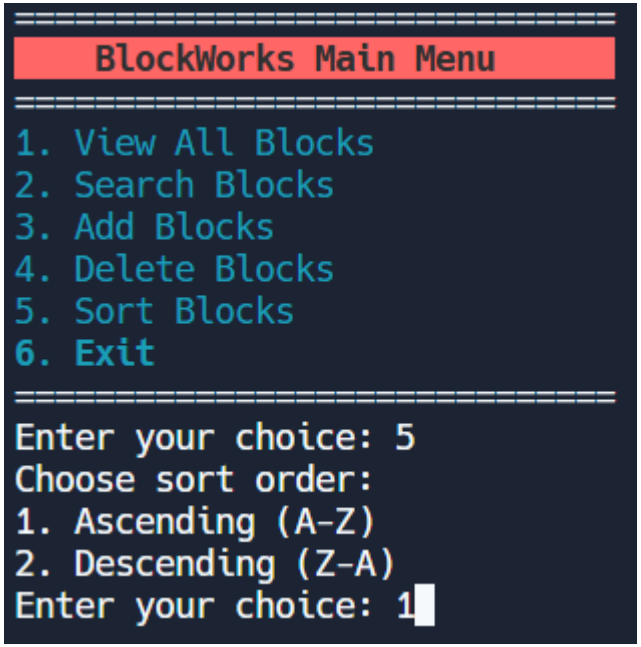


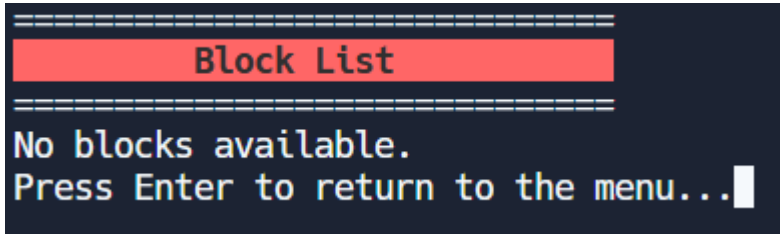
Test	Description	Data	Expected Outcome	Actual Outcome
D3	Test with user declining the deletion.	List = ["Armour Block"] Block Name = "Armour Block" User Response = "no"	The block is not deleted, and the function informs the user that the block was not deleted.	The block is not deleted, and the function informs the user that the block was not deleted.



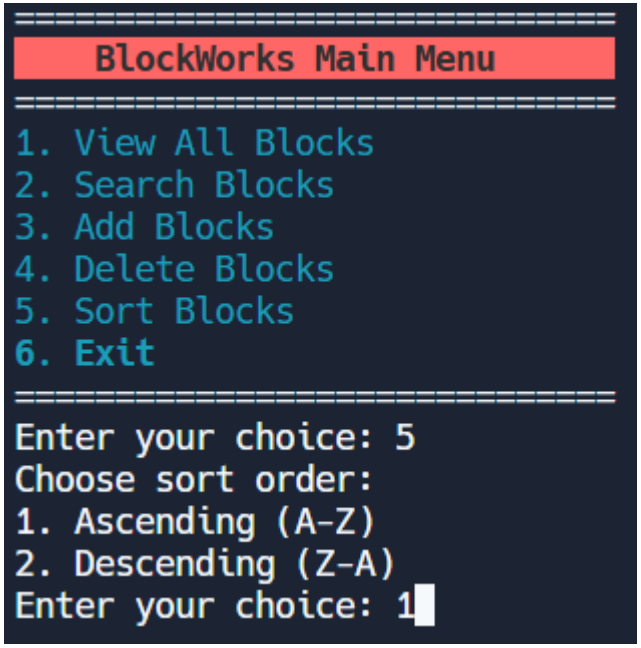
SORT - INSERTION

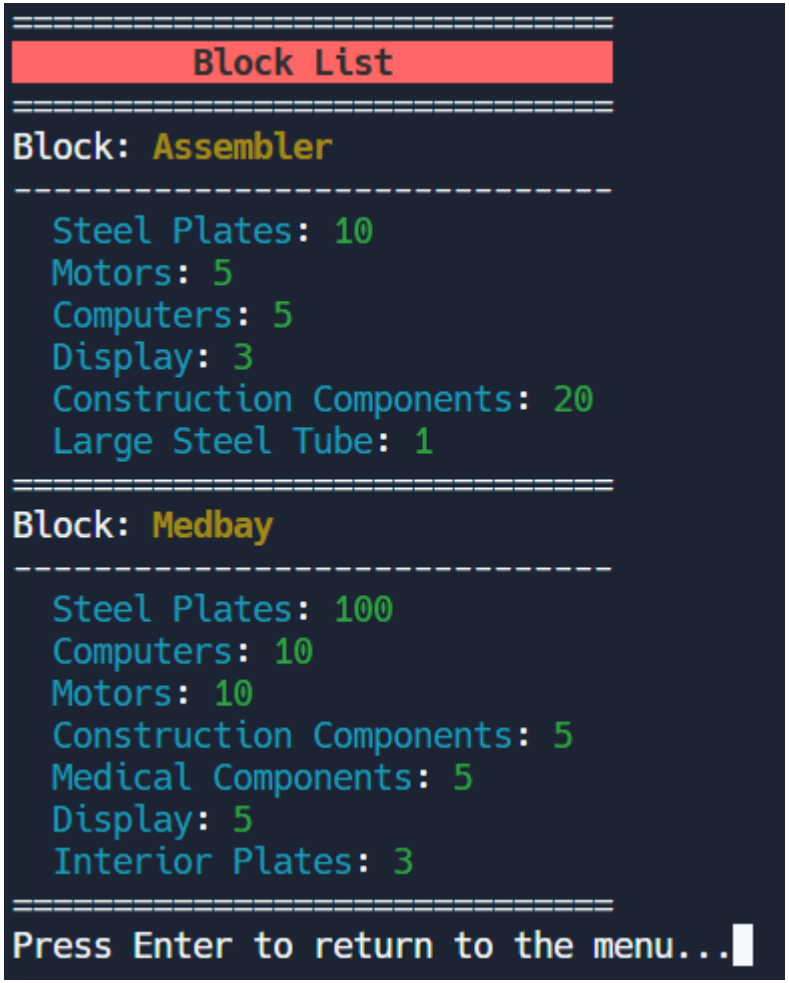
Test	Description	Data	Expected Outcome	Actual Outcome
I1	Test with an empty list.	List = [] Sort Order = "A-Z"	The function returns an empty list since there are no blocks to sort.	The function returns an empty list since there are no blocks to sort.



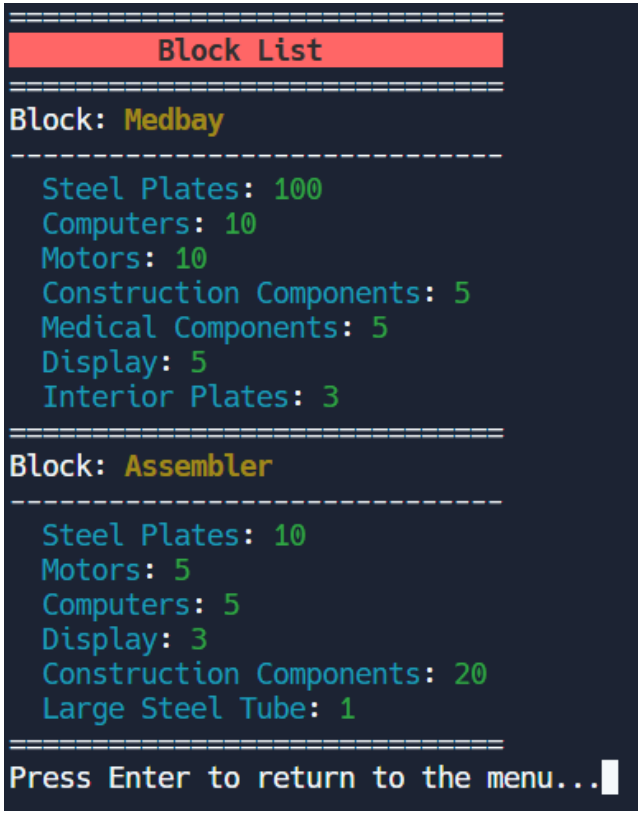


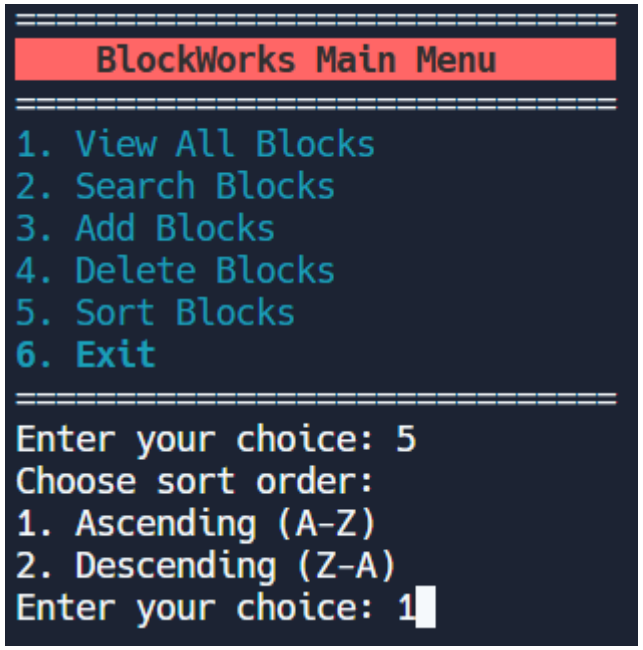
Test	Description	Data	Expected Outcome	Actual Outcome
I2	Test with a list of blocks sorted in ascending order (A-Z).	List = ["Assembler", " Medbay"] Sort Order = "A-Z"	The blocks are already sorted, so the list remains unchanged.	The blocks are already sorted, so the list remains unchanged.

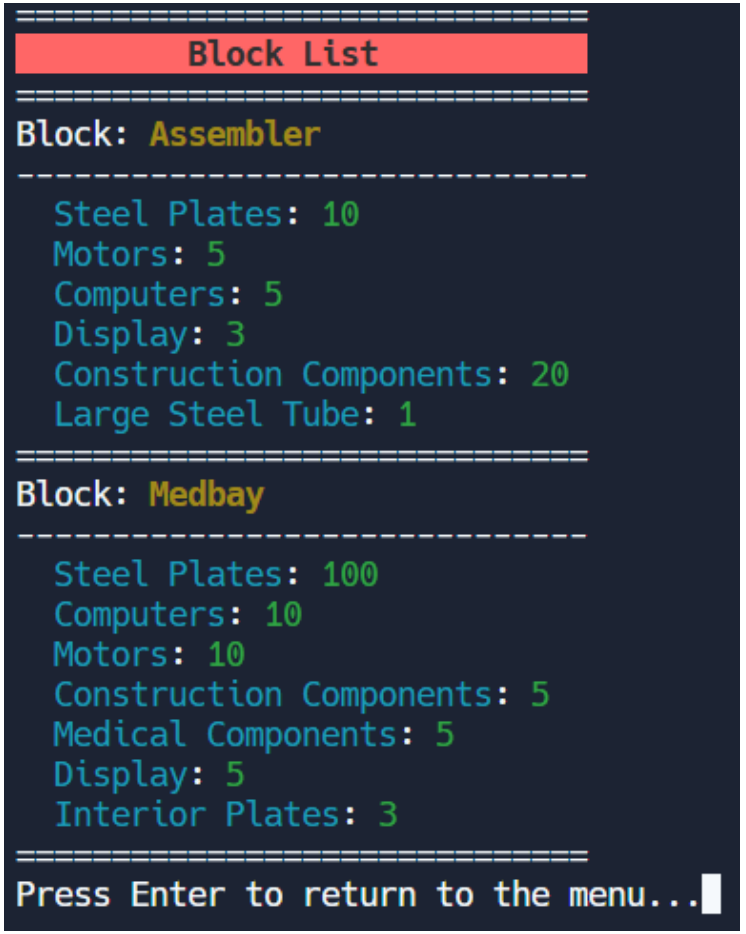




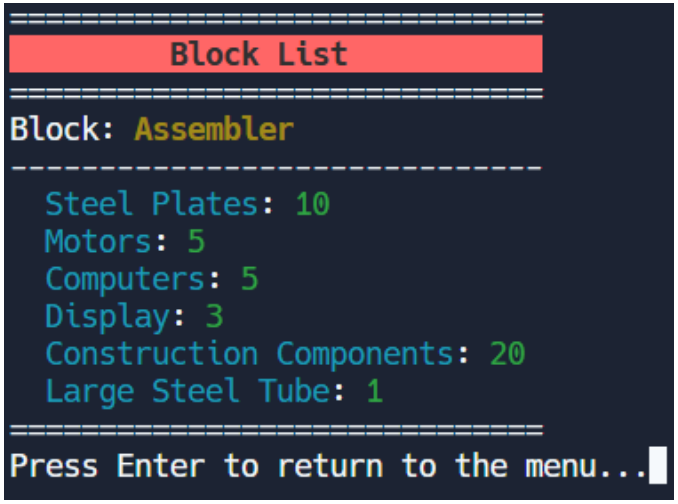
Test	Description	Data	Expected Outcome	Actual Outcome
I3	Test with a list containing blocks that are out of order.	List = ["Medbay", " Assembler"] Sort Order = "A-Z"	The function correctly sorts the blocks in ascending order.	The function correctly sorts the blocks in ascending order.

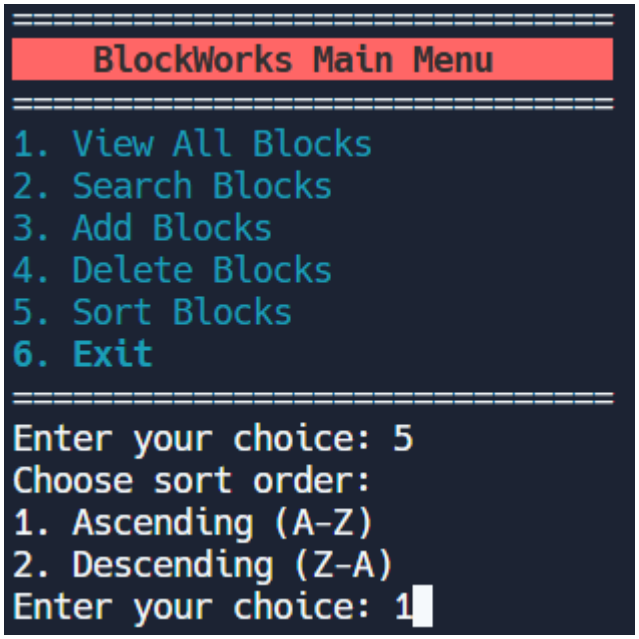


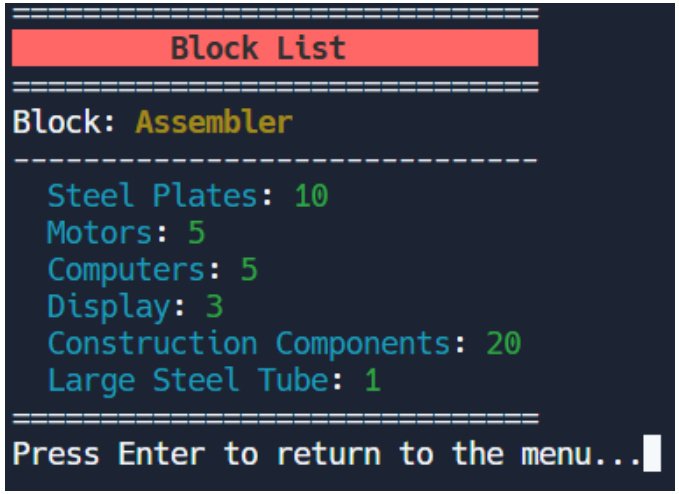




Test	Description	Data	Expected Outcome	Actual Outcome
I4	Test with a list containing a single block.	List = ["Assembler"] Sort Order = "A-Z"	The list remains unchanged since a single block does not need sorting.	The list remains unchanged since a single block does not need sorting.

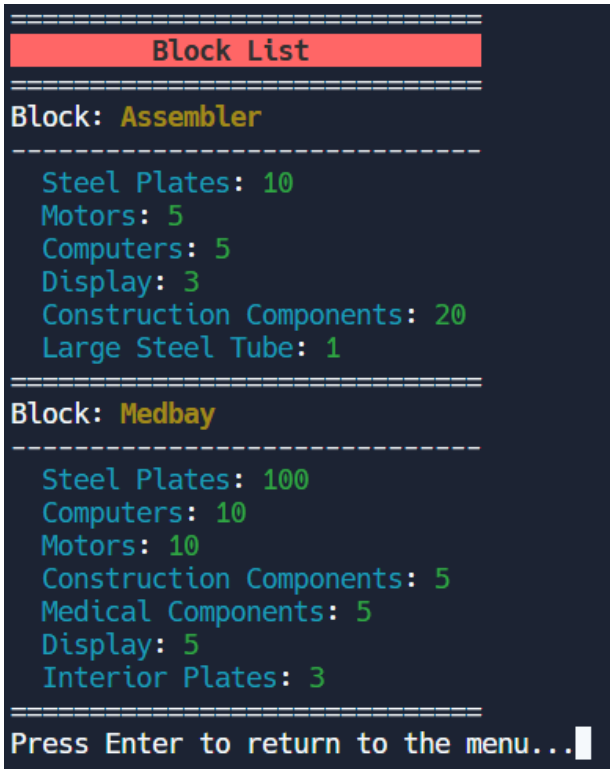


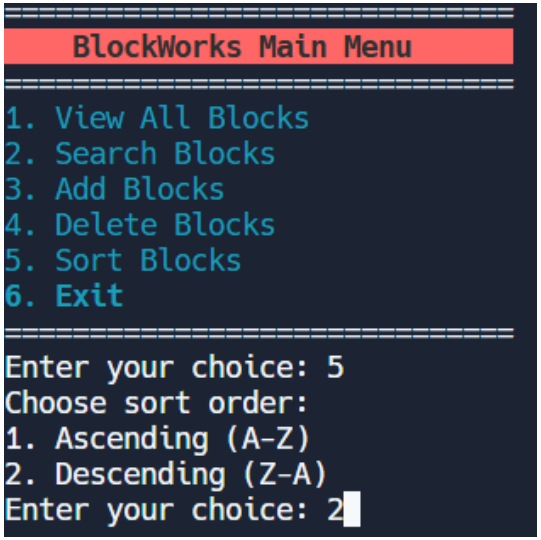


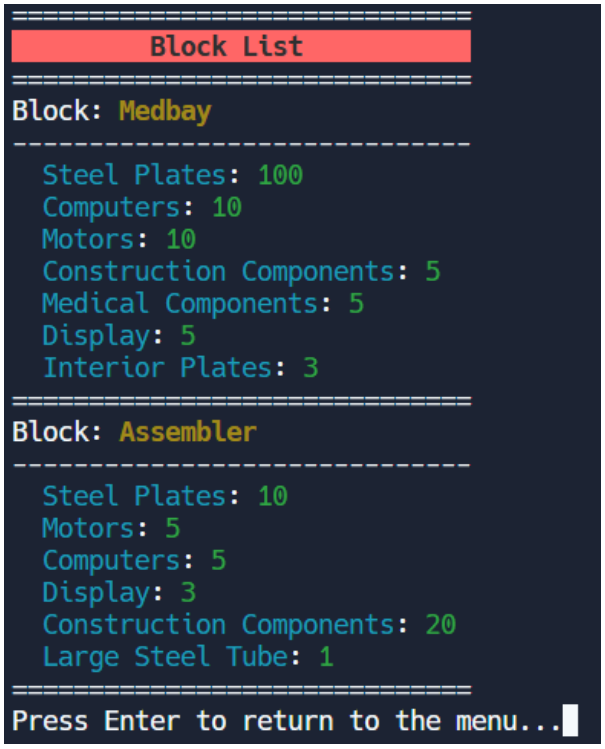


Test I5 modified to sort descending.

Test	Description	Data	Expected Outcome	Actual Outcome
I5	Test with a list containing blocks that are out of order.	List = ["Assembler", " Medbay"] Sort Order = "Z-A"	The function correctly sorts the blocks in decending order.	The function correctly sorts the blocks in decending order.

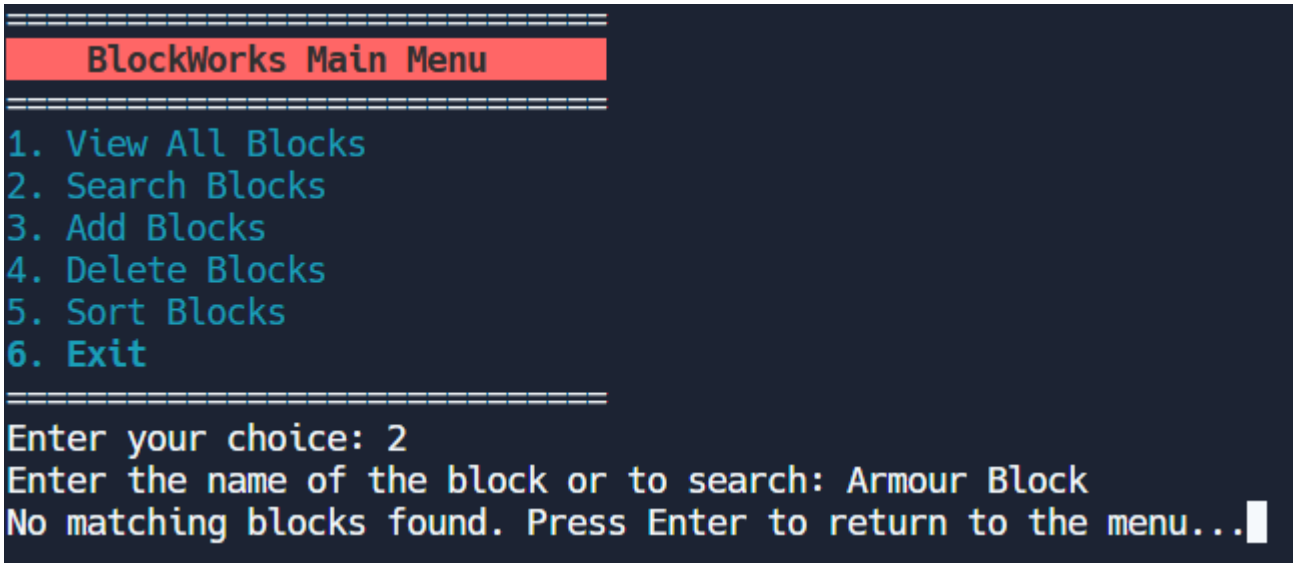




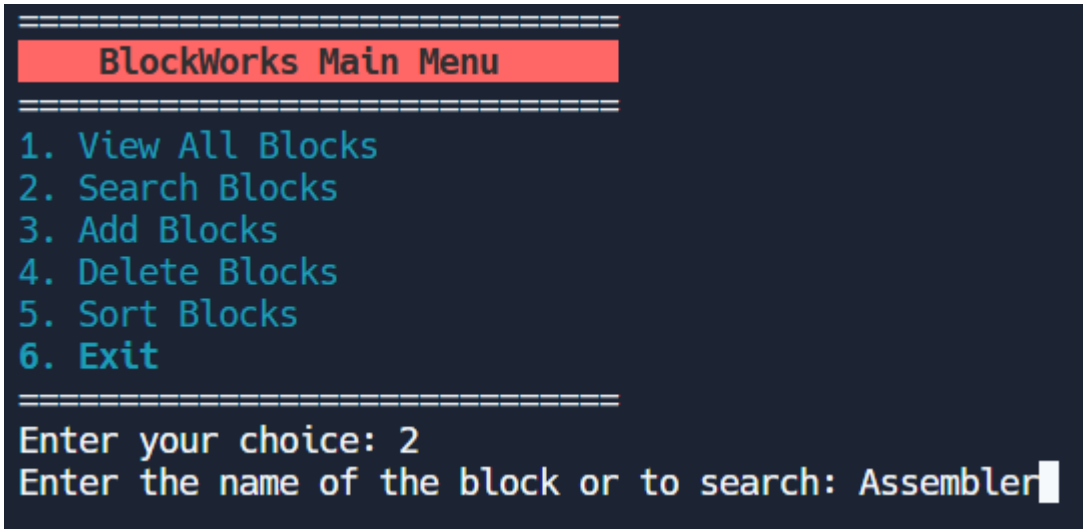


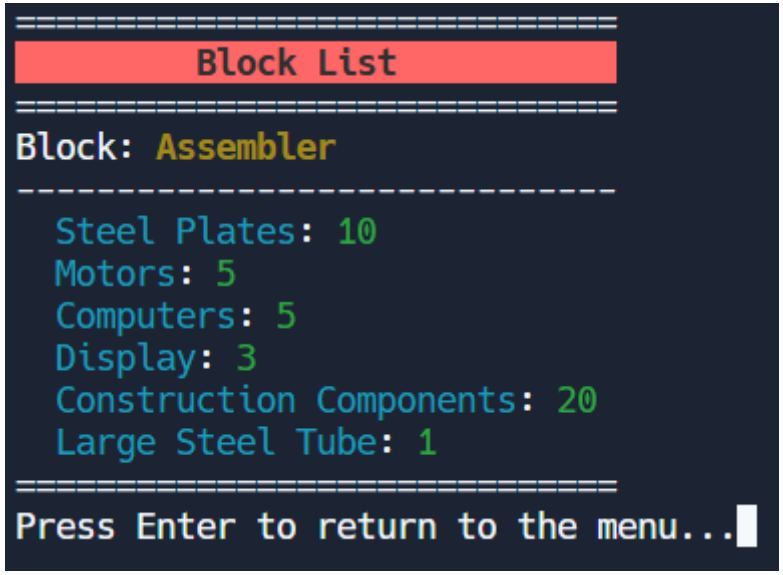
SEARCH - SERIAL

Test	Description	Data	Expected Outcome	Actual Outcome
S1	Test with an empty list.	List = [] Search Term = "Armour Block"	The function returns no results or an appropriate message indicating no blocks are available.	The function returns no results or an appropriate message indicating no blocks are available.

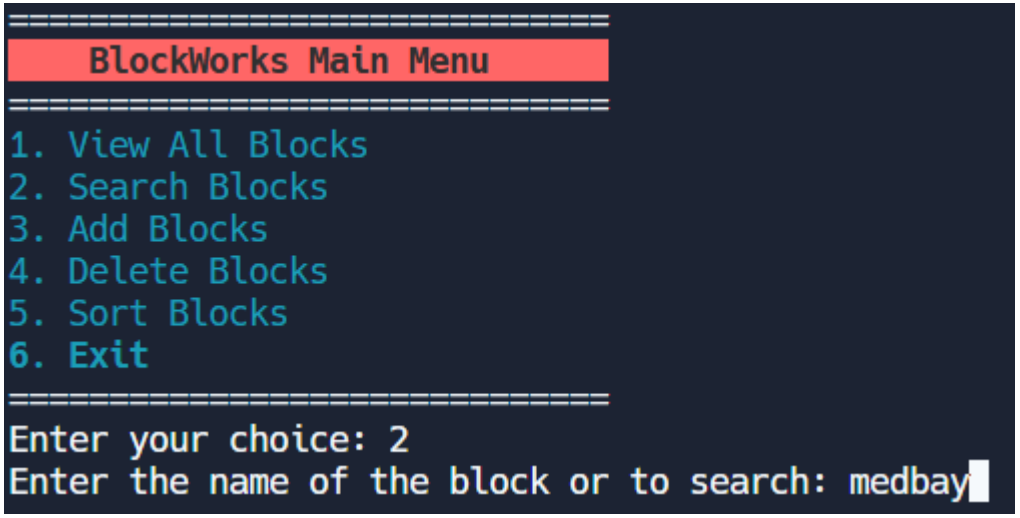


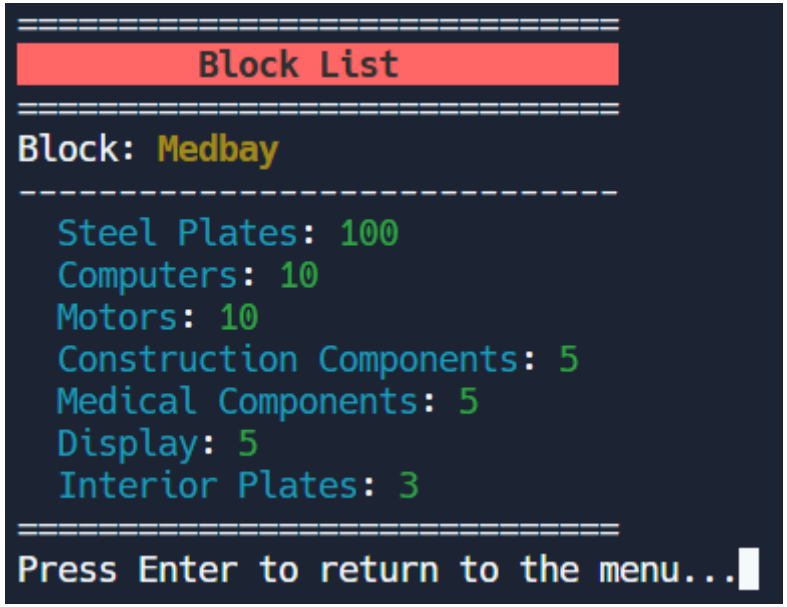
Test	Description	Data	Expected Outcome	Actual Outcome
S2	Test with a list containing blocks and an exact match search term.	List = ["Assembler", "Medbay"] Search Term = "Assembler"	The function returns the block whose name exactly matches the search term.	The function returns the block whose name exactly matches the search term.



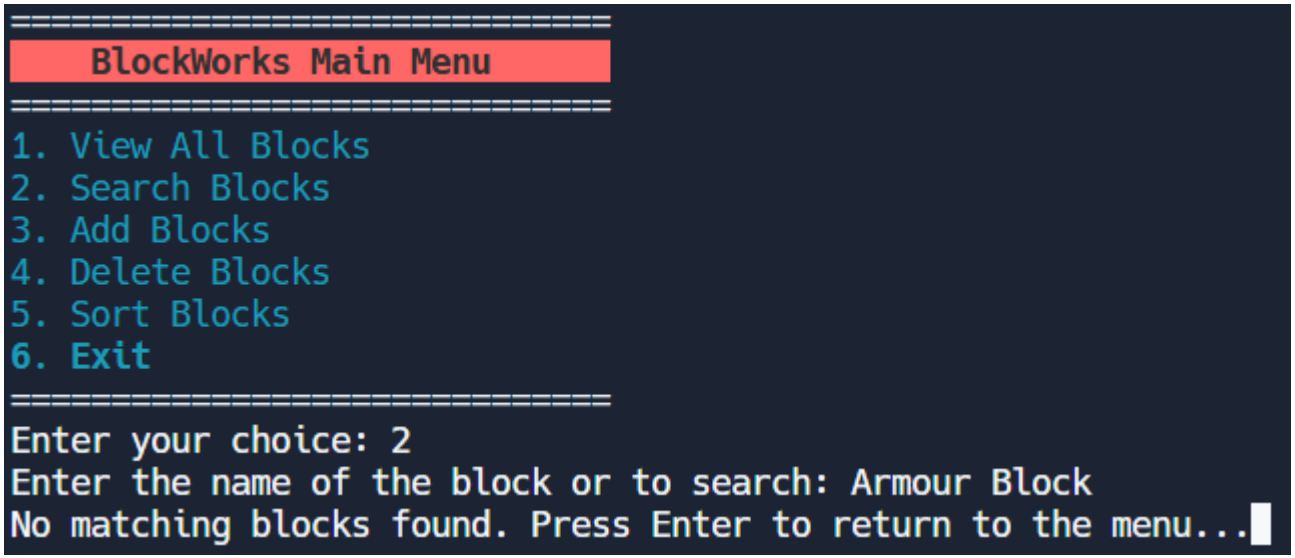


Test	Description	Data	Expected Outcome	Actual Outcome
S3	Test with a case-insensitive search term.	List = ["Assembler", "Medbay"] Search Term = "medbay"	The function correctly matches blocks regardless of case (upper/lower).	The function correctly matches blocks regardless of case (upper/lower).



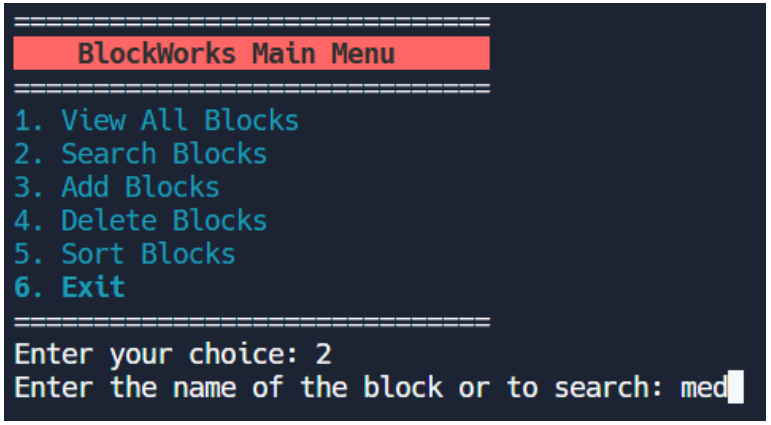


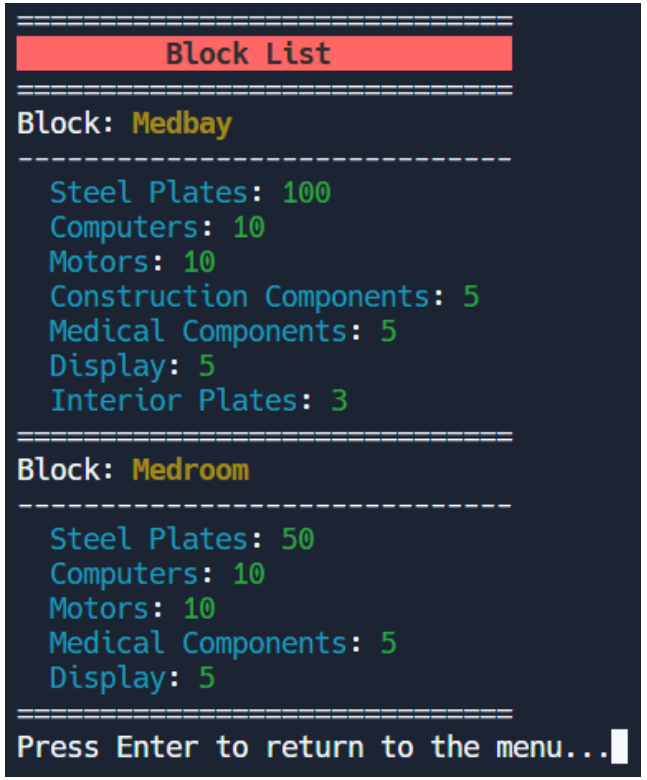
Test	Description	Data	Expected Outcome	Actual Outcome
S4	Test with a search term that does not match any block name.	List = ["Assembler", "Medbay"] Search Term = "Armour Block"	The function returns no results or an appropriate message indicating no matching blocks.	The function returns no results or an appropriate message indicating no matching blocks.



Test S5 added to test for partial match.

Test	Description	Data	Expected Outcome	Actual Outcome
S5	Test with a search term that that is a partial match.	List = ["Assembler", "Medbay", "Medroom"] Search Term = "med"	The function returns the block whose name containes the search term.	The function returns the block whose name containes the search term.





BLOCKWORKS EVALUATION

EVALUATION

REQUIREMENTS

The BlockWorks program successfully implements all required functionalities. The add, delete, and search operations work as expected, with user confirmation before deletion to prevent accidental removal. The sorting feature functions well, handles partial name searches effectively.

Enter at least five records.

This was successfully met. The "Add New Blocks" feature prompts users to enter block names and components with their quantities. Multiple blocks can be added, and the system can handle at least five records.

Delete and search records based on keyword/string entry.

This was successfully met. The deletion process allows users to remove blocks by name, and a confirmation prompt prevents accidental deletion.

Sort records in a specific order and display them on the screen.

This was successfully met. The search functionality allows keyword entry to find matching blocks by name or component. The serial search supports partial name searches, improving usability.

Select an option on the screen to perform a function.

This was successfully met. The menu system allows users to select from various options, making navigation simple and intuitive.

Get a prompt on screen before an action is performed.

This was successfully met. Confirmation prompts ensure users do not perform irreversible actions, like adding or deleting a block, unintentionally.

LIMITATIONS AND DEVELOPMENTS

A challenge during development was ensuring robust input validation for quantities. The program rejects non-numeric input and negative numbers, prompting users to re-enter valid data.

A limitation is the use of the insertion sort algorithm, which may not be efficient for larger datasets. Future versions could use advanced algorithms like quicksort or mergesort for better performance.

REFERENCES

Brayshaw, T. (2024) *Blockworks: Design. LCS: Launching into Computer Science.* Report submitted to the University of Essex Online.

Cormen, T. H., Leiserson, C. E., Stein, C., & Rivest, R. L. (2022) *INTRODUCTION TO ALGORITHMS, 4TH EDITION*. Cambridge, Mass., USA: MIT Press. Available at: https://dl.ebooksworld.ir/books/Introduction.to.Algorithms.4th.Leiserson.Stein.Rivest.Cormen.MIT.Press.9780262046305.EBooksWorld.ir.pdf.

Keen Software House (no date) *Space Engineers Official Website*. Available at: https://www.spaceengineersgame.com/ (Accessed: December 13, 2024).

Knuth, D. (2011) *The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1*. Hoboken: Pearson Education, Limited.

Nielsen, J. (1994) *Usability Engineering*. San Francisco, CA: Morgan Kaufmann.

Appendix A - Code

```
import sys
import os


def main():
    """
    Main function that controls the flow of the program.
    Displays the splash screen, initializes blocks, and
    displays the main menu for user interaction.
    """
    splash_screen()
    blocks = initialise_blocks()

    while True:
        clear_console()
        display_menu()
        choice = get_user_choice()

        if choice == 1:
            display_blocks(blocks)  # Show all blocks
            input("Press Enter to return to the menu...")
        elif choice == 2:
            search_block(blocks)  # Search for a block
        elif choice == 3:
            add_block(blocks)  # Add a new block
        elif choice == 4:
            delete_block(blocks)  # Delete an existing block
        elif choice == 5:
            sort_blocks(blocks)  # Sort blocks alphabetically
        elif choice == 6:
            print("Exiting BlockWorks. Goodbye!")  # Exit the program
            sys.exit()


def splash_screen():
    """
    Displays the splash screen when the program starts.
    Shows a welcome message and some loading information.
    """
    clear_console()
    print("=" * 50)
    print("\033[1;37;41m              Welcome to BlockWorks               \033[0m")
    print("=" * 50)
    print("\033[1;36m  A Program for Managing Blocks and Components  \033[0m")
    print("\n\033[1;33m  Loading...\033[0m")
    print("=" * 50)
    input("Press Enter to continue to the main menu...")


def display_menu():
    """
    Displays the main menu to the user, where they can choose
    what action to perform in the program.
    """
    print("=" * 30)
    print("\033[1;37;41m    BlockWorks Main Menu      \033[0m")
    print("=" * 30)

    print("\033[36m1. View All Blocks\033[0m")
    print("\033[36m2. Search Blocks\033[0m")
    print("\033[36m3. Add Blocks\033[0m")
```

```
    print("\033[36m4. Delete Blocks\033[0m")
    print("\033[36m5. Sort Blocks\033[0m")
    print("\033[1;36m6. Exit\033[0m")

    print("=" * 30)


def get_user_choice():
    """
    Prompts the user to select a choice from the menu and
    validates the input to ensure it's a valid choice.
    """
    while True:
        try:
            choice = int(input("Enter your choice: "))
            if 1 <= choice <= 6:
                return choice
            else:
                print("Invalid choice. Please choose a number between 1 and 6.")
        except ValueError:
            print("Invalid input. Please enter a number.")


def clear_console():
    """
    Clears the console based on the operating system.
    For Windows, uses 'cls'; for Linux/macOS, uses 'clear'.
    """
    if os.name == 'nt':  # If on Windows
        os.system('cls')
    else:  # If on Linux or macOS
        os.system('clear')


def confirm_action(prompt="Are you sure? (yes/no): "):
    """
    Asks the user for confirmation before performing an action.
    Returns True for 'yes' and False for 'no', ensures valid input.
    """
    while True:
        user_input = input(prompt).strip().lower()
        if user_input in ["yes", "y"]:
            return True
        elif user_input in ["no", "n"]:
            return False
        else:
            print("Invalid input. Please enter 'yes' or 'no'.")


def display_blocks(blocks):
    """
    Displays a list of all blocks and their components.
    If no blocks are available, shows a message.
    """
    clear_console()

    print("=" * 30)
    print("\033[1;37;41m         Block List           \033[0m")
    print("=" * 30)

    if not blocks:
        print("No blocks available.")
    else:
```

```
        for block in blocks:
            print(f"Block: \033[1;33m{block['name']}\033[0m")
            print("-" * 30)
            for component, quantity in block['components'].items():
                print(f"  \033[36m{component}\033[0m: \033[32m{quantity}\033[0m")
            print("=" * 30)


def search_block(blocks):
    """
    Prompts the user to enter a search term and displays blocks
    whose names match the search term.
    """
    search_term = input("Enter the name of the block or to search: ").lower()
    found_blocks = []

    for block in blocks:
        if search_term in block['name'].lower():
            found_blocks.append(block)

    if found_blocks:
        display_blocks(found_blocks)
        input("Press Enter to return to the menu...")
    else:
        input("No matching blocks found. Press Enter to return to the menu...")


def add_block(blocks):
    """
    Prompts the user to add a new block by entering its name and
    components. Validates input before adding the block to the list.
    """
    available_components = [
        "Steel Plates", "Motors", "Computers", "Display",
        "Construction Components", "Large Steel Tube", "Small Steel Tube", "Power
Cells",
        "Reactor Components", "Interior Plates", "Medical Components"
    ]

    block_name = input("Enter the name of the new block: ").strip()

    if not block_name:
        print("Block name cannot be empty. Please try again.")
        input("Press Enter to return to the menu...")
        return blocks

    if any(block['name'].lower() == block_name.lower() for block in blocks):
        print(f"A block with the name '{block_name}' already exists. Please enter a
unique name.")
        input("Press Enter to return to the menu...")
        return blocks

    components = {}

    for component in available_components:
        while True:
            try:
                quantity = int(input(f"Enter the quantity for {component}: ").strip())

                if quantity < 0:
                  print("Quantity cannot be negative. Please try again.")
                  continue
```

```
                if quantity > 0:
                    components[component] = quantity
                break
            except ValueError:
                print("Please enter a valid number for the quantity.")

    if not components:
        print("No components entered for this block. The block will not be added.")
        input("Press Enter to return to the menu...")
        return blocks

    new_block = {
        "name": block_name,
        "components": components
    }

    display_blocks([new_block])

    if confirm_action("Do you want to add this block to the list? (yes/no): "):
        blocks.append(new_block)
        print(f"Block '{block_name}' has been added successfully.")
    else:
        print("Block was not added.")

    input("Press Enter to return to the menu...")
    return blocks


def delete_block(blocks):
    """
    Prompts the user to delete a block by its name. Confirms
    before deleting the block from the list.
    """
    block_name = input("Enter the exact name of the block to delete: ").strip()

    if not block_name:
        print("Block name cannot be empty. Please try again.")
        input("Press Enter to return to the menu...")
        return blocks

    found_block = None
    for block in blocks:
        if block_name.strip().lower() == block['name'].lower():
            found_block = block
            break

    if found_block:
        display_blocks([found_block])

        if confirm_action(f"Are you sure you want to delete the block
'{found_block['name']}'? (yes/no): "):
            blocks.remove(found_block)
            print(f"Block '{found_block['name']}' has been deleted.")
        else:
            print("Block not deleted.")
    else:
        print(f"No block found with the exact name '{block_name}'.")

    input("Press Enter to return to the menu...")
    return blocks


def insertion_sort(blocks, ascending=True):
```

```
    """
    Sorts the blocks alphabetically by their name using the
    insertion sort algorithm. Can sort in ascending or descending order.
    """
    for i in range(1, len(blocks)):
        key = blocks[i]
        j = i - 1
        while (
            j >= 0
            and (
                (key['name'].lower() < blocks[j]['name'].lower() and ascending)
                or
                (key['name'].lower() > blocks[j]['name'].lower() and not ascending)
            )
        ):
            blocks[j + 1] = blocks[j]
            j -= 1
        blocks[j + 1] = key
    return blocks


def sort_blocks(blocks):
    """
    Prompts the user to choose between sorting blocks in ascending
    (A-Z) or descending (Z-A) order and then sorts the blocks accordingly.
    """
    print("Choose sort order:")
    print("1. Ascending (A-Z)")
    print("2. Descending (Z-A)")

    choice = input("Enter your choice: ").strip()

    if choice == "1":
        blocks = insertion_sort(blocks, ascending=True)
    elif choice == "2":
        blocks = insertion_sort(blocks, ascending=False)
    else:
        input("Invalid choice. Press Enter to return to the menu...")
        return

    display_blocks(blocks)
    input("Press Enter to return to the menu...")


def initialise_blocks():
    """
    Initialises the blocks with predefined data. Returns a list
    of blocks with its name and a dictionary of components and quantities.
    """
    blocks = [
        {
            "name": "Assembler",
            "components": {
                "Steel Plates": 10,
                "Motors": 5,
                "Computers": 5,
                "Display": 3,
                "Construction Components": 20,
                "Large Steel Tube": 1
            }
        },
        {
            "name": "Reactor (Small)",
```

```
            "components": {
                "Steel Plates": 20,
                "Power Cells": 10,
                "Computers": 5,
                "Construction Components": 5,
                "Reactor Components": 1
            }
        },
        {
            "name": "Large Cargo Container",
            "components": {
                "Steel Plates": 60,
                "Construction Components": 10,
                "Interior Plates": 5,
                "Motors": 4,
                "Large Steel Tubes": 2,
                "Computers": 1
            }
        },
        {
            "name": "Oxygen Generator",
            "components": {
                "Steel Plates": 15,
                "Computers": 5,
                "Interior Plates": 2,
                "Motors": 5,
                "Large Steel Tube": 1,
                "Construction Components": 5
            }
        },
        {
            "name": "Medbay",
            "components": {
                "Steel Plates": 100,
                "Computers": 10,
                "Motors": 10,
                "Construction Components": 5,
                "Medical Components": 5,
                "Display": 5,
                "Interior Plates": 3
            }
        }
    ]
    return blocks


if __name__ == '__main__':
    main()  # Run the main function to start the program
```