FAST NUCES Programming Olympics

PROGRAMMING FUNDAMENTAL CS1002 - Fall 2024

Release Date: 11th October 2024 Due Date: 29th October 2024

Total Marks 90

Congratulations, athletes! You've advanced to the next stage of the Programming Olympics for Fall 2024 by excelling in Assignment 1. Your journey in CS1002 continues with the second round of challenges.

- The rules remain simple: Your mission is to cross the finish line, but not just with speed. This time, we focus even more on elegance, creativity, and the dedication you put into your problem-solving journey.
- True champions face obstacles with integrity. Remember, the biggest challenge isn't just solving the problems—it's overcoming the temptations of using shortcuts like ChatGPT, Gemini, or relying on others' solutions. Such actions will lead to immediate disqualification.

Now, show us your best work, stay honest, and keep your eyes on the prize. Let the challenge begin!

How to submit

There are a total of 9 questions in the assignment and each question carries equal marks i.e 10. You must submit the .c files compressed in a zip folder with your student id on google classroom within the due date.

Question : 1

Write a c program to find the second smallest element in an array.
Input 5 elements in the array (value must be <9999) :
element - 0 : 0
element - 1 : 9
element - 2 : 4
element - 3 : 6
element - 4 : 5
Expected Output :
The Second smallest element in the array is : 4

```c
#include <stdio.h>

int main() {
    int arr[5];
    int first_smallest, second_smallest;
    first_smallest = second_smallest = 9999;
    printf("Input 5 elements in the array (value must be < 9999):\n");
    for (int i = 0; i < 5; i++) {
        printf("element - %d: ", i);
        scanf("%d", &arr[i]);
        if (arr[i] >= 9999) {
            printf("Please enter a value less than 9999.\n");
            return 1;
        }

        if (arr[i] < first_smallest) {
            second_smallest = first_smallest;
            first_smallest = arr[i];
                } else if (arr[i] < second_smallest && arr[i] !=
first_smallest) {
            second_smallest = arr[i];
        }
    }
    if (second_smallest == 9999) {
        printf("There is no second smallest element.\n");
    } else {
          printf("The Second smallest element in the array is: %d\n",
second_smallest);
```

```
    }

    return 0;
}
```

Question : 2

A popular beverage company is looking to optimize their marketing campaigns by understanding the frequency of characters used in their promotional slogans. They believe that analyzing these patterns can help them tailor their messaging to specific demographics and improve brand recall. You've been tasked with creating a C code function to analyze the character frequency in a list of slogans provided by the marketing team. This analysis will help identify the most common letters used in their slogans, which can inform future marketing strategies.

Input:
slogans = ["buy now", "save big", "limited offer"]

Expected Output:

- For "buy now": {'b': 1, 'u': 1, 'y': 1, ' ': 1, 'n': 1, 'o': 1, 'w': 1}
- For "save big": {'s': 1, 'a': 1, 'v': 1, 'e': 1, ' ': 1, 'b': 1, 'i': 1, 'g': 1}
- For "limited offer": {'l': 1, 'i': 1, 'm': 1, 't': 1, 'e': 2, 'd': 1, ' ': 1, 'o': 1, 'f': 1, 'r': 1}

```
#include <stdio.h>
#include <string.h>
#define MAX_SLOGANS 10
#define MAX_LENGTH 100
void    analyze_slogans(char    slogans[MAX_SLOGANS][MAX_LENGTH],    int
num_slogans) {
    for (int i = 0; i < num_slogans; i++) {
        int frequency[256] = {0};
        char slogan[MAX_LENGTH];
        strcpy(slogan, slogans[i]);
        for (int j = 0; j < strlen(slogan); j++) {
            frequency[(unsigned char)slogan[j]]++;
        }

        printf("For \"%s\": {", slogan);
        for (int k = 0; k < 256; k++) {
            if (frequency[k] > 0) {
                printf("%c: %d,",k, frequency[k]);
            }
        }
        printf("\b }\n");
    }
}

int main() {
    char slogans[MAX_SLOGANS][MAX_LENGTH] = {
        "buy now",
        "save big",
        "limited offer"
    };
    int num_slogans = 3;

    analyze_slogans(slogans, num_slogans);
    return 0;
}
```

———

Question : 3

You are working for a text-processing company that optimizes storage for large amounts of written data. The company has identified a recurring issue: words in documents are often unnecessarily lengthy due to repeated consecutive characters. Your task is to minimize these words by reducing consecutive duplicate characters to a single instance of the character.

Operations:

Compress a Word: Write a function that takes a string (a word) as input and returns a minimized version of the word, where all consecutive duplicate letters are reduced to one.

Multiple Word Compression: You will then apply this function to a list of words and return the minimized version for each word.

Add a feature to calculate how many characters were removed during the compression process.

Input: words = [“boooooook”, “coooool”, “heeeey”]

Output: [“bok”, “col”, “hey”]

———

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int compressWord(char word[], char compressed[]) {
    int length = strlen(word);
    int index = 0;
    int removedCount = 0;

    for (int i = 0; i < length; i++) {

        if (i == 0 || word[i] != word[i - 1]) {
            compressed[index++] = word[i];
        } else {
            removedCount++;
        }
    }

    compressed[index] = '\0';
    return removedCount;
}

void compressWords(char words[][100], int numWords) {
    char compressed[100];

    for (int i = 0; i < numWords; i++) {
        int removedCount = compressWord(words[i], compressed);
            printf("Original: %s, Compressed: %s, Characters Removed:
%d\n",
               words[i], compressed, removedCount);
    }
}

int main() {
    char words[3][100] = {"boooooook", "coooool", "heeeey"};
    int numWords = sizeof(words) / sizeof(words[0]);

    compressWords(words, numWords);
    return 0;
}
```

Question : 4

You are working as a software engineer at a company that manages a large database of customer transactions. Each transaction is stored as a string that contains encoded information about the customer. However, some transactions may have been duplicated due to system errors, and these duplicates appear in scrambled order. Your task is to identify and group these "scrambled" transactions together, as they represent the same customer activity. Two transactions are considered scrambled if they contain the same letters but in a different order (anagrams).You are provided with a list of transaction strings. You need to write a program that groups these scrambled transactions into separate categories, returning each group of related transactions together.

Example

Input transactions = ["eat", "tea", "tan", "ate", "nat", "bat"]

Expected Output: [['bat'], ['nat', 'tan'], ['ate', 'eat', 'tea']]

```c
#include <stdio.h>
#include <string.h>
void sortString(char str[100]) {
    int n = strlen(str);
    int i, j;
    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (str[j] > str[j + 1]) {
                char temp = str[j];
                str[j] = str[j + 1];
                str[j + 1] = temp;
            }
        }
    }
}
int areAnagrams(char str1[100], char str2[100]) {
    if (strlen(str1) != strlen(str2)) {
        return 0;
    }
    char temp1[100], temp2[100];
    int i;
    for (i = 0; str1[i] != '\0'; i++) {
        temp1[i] = str1[i];
    }
    temp1[i] = '\0';

    for (i = 0; str2[i] != '\0'; i++) {
        temp2[i] = str2[i];
    }
    temp2[i] = '\0';
    sortString(temp1);
    sortString(temp2);
    return strcmp(temp1, temp2) == 0;
}

int main() {
     char transactions[100][100] = {"eat", "tea", "tan", "ate", "nat",
"bat"};
    int n = 6;
    int grouped[100] = {0};
    int i, j;

    printf("Grouped Anagrams:\n");

    for (i = 0; i < n; i++) {
        if (grouped[i]) continue;

        printf("[");
        printf("%s", transactions[i]);
        grouped[i] = 1;

        for (j = i + 1; j < n; j++) {
                    if  (!grouped[j]  &&  areAnagrams(transactions[i],
transactions[j])) {
                printf(", %s", transactions[j]);
                grouped[j] = 1;
            }
        }
        printf("]\n");
    }
```

```c
    return 0;
}
```

Question : 5

You are tasked with creating a program that generates a histogram based on user input. The user will provide a set of values, and your program will use loops to create a visual representation of these values.

1. **Input Handling**: Write a function that takes an array of integers (the values) and the count of those integers as input.
2. **Horizontal Histogram Function**: Use loops to generate and print a histogram, where each value is represented by asterisks (*).
3. **Vertical Histogram Function**: Use loops to generate and print a vrtical histogram, where each value is represented by asterisks (*).

Expected Values:

int values[] = {3, 5, 1, 4};

int count = 4;

Horizontal Histogram

```
Value 1: ***
Value 2: *****
Value 3: *
Value 4: ****
```

Vertical Histogram

```
  *
  *   *
* *   *
* *   *
* * * *
3 5 1 4
```

```c
#include <stdio.h>
void horizontal_histogram(int values[], int count) {
    printf("Horizontal Histogram:\n");
    for (int i = 0; i < count; i++) {
        printf("Value %d: ", values[i]);
        for (int j = 0; j < values[i]; j++) {
            printf("*");
        }
        printf("\n");
    }
}

void vertical_histogram(int values[], int count) {
    int max_value = values[0];
    for (int i = 1; i < count; i++) {
        if (values[i] > max_value) {
            max_value = values[i];
        }
    }
    printf("\nVertical Histogram:\n");
    for (int i = max_value; i > 0; i--) {
        for (int j = 0; j < count; j++) {
            if (values[j] >= i) {
                printf("*");
            } else {
                printf(" ");
            }
            printf(" ");
        }
        printf("\n");
    }
    for (int i = 0; i < count; i++) {
        printf("%d ", values[i]);
    }
    printf("\n");
}

int main() {
    int values[] = {3, 5, 1, 4};
    int count = sizeof(values) / sizeof(values[0]);
    horizontal_histogram(values, count);
    vertical_histogram(values, count);
    return 0;
```

```
}
```

——

Question : 6

Two friends, A and B, are playing the game of matchsticks. In this game, a group of N matchsticks is placed on the table. The players can pick any number of matchsticks from 1 to 4 (both inclusive) during their chance. The player who takes the last match stick wins the game. If A starts first, how many matchsticks should he pick on his 1st turn such that he is guaranteed to win the game or determine if it's impossible for him to win? Write a function which returns -1 if it's impossible for A to win the game, else return the number of matchsticks he should pick on his 1st turn such that he is guaranteed to win.

——

```
#include <stdio.h>
int firstMove(int N) {
    if (N % 5 == 0) {
        return -1;
    }
    return N % 5;
}

void playGame(int N) {
    int matchsticks = N;
    int pick, turn = 1;
    int optimalPick = firstMove(N);
    if (optimalPick == -1) {
         printf("It is impossible for A to guarantee a win on the first
move, but let's play!\n");
    } else {
          printf("A should pick %d matchstick(s) on the first turn to
guarantee a win.\n", optimalPick);
        matchsticks -= optimalPick;
        printf("Remaining matchsticks: %d\n", matchsticks);
        turn = 2;  // Now it's B's turn
    }
    while (matchsticks > 0) {
        if (turn == 1) {
            printf("A's turn. Pick matchsticks (1 to 4): ");
            scanf("%d", &pick);
        } else {
            printf("B's turn. Pick matchsticks (1 to 4): ");
            scanf("%d", &pick);
        }

        if (pick < 1 || pick > 4) {
             printf("Invalid pick. Please choose a number between 1 and 
4.\n");
            continue;
        }
        matchsticks -= pick;
        if (matchsticks <= 0) {

            if (turn == 1) {
                printf("A wins!\n");
            } else {
                printf("B wins!\n");
            }
            break;
        }

        // Print remaining matchsticks
        printf("Remaining matchsticks: %d\n", matchsticks);

        // Switch turns: 1 for A, 2 for B
        turn = (turn == 1) ? 2 : 1;
    }
}

int main() {
    int N;
    printf("Enter the number of matchsticks: ");
    scanf("%d", &N);

    playGame(N);

    return 0;
}
```

Question : 7

One of the master coders designed a subscript block that stores random characters in a multidimensional array. You are asked to design a program that will find a given string in a multidimensional array of characters. The search for characters can be present and operational from left to right and top to down only. The program should create a 6 x 5 2D array and populate it with random alphabet characters. After that, the program should print it in a tabular form as shown below. Search the user-entered string in the 2D array, if it is present then add a point to the score, if it is not available then subtract one. Print the score at every input. The program stops asking and re-populates the 2D array with new random characters when the user enters "END" as the string.

[NOTE: After generating random numbers, the last row should have your student ID's last four digits]

E	D	D	F	R
A	F	V	A	Q
T	E	B	S	T
L	J	G	T	T
1	2	3	4	Q

Search Str= "FAST"

Output:

FAST is present Score 1

Search Str= "EAT"

EAT is present Score 2

Search Str= "GREAT"

GREAT is not present Score 1

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

void generateArray(char arr[6][5], int studentID) {
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 5; j++) {
            arr[i][j] = 'A' + (rand() % 26);
        }
    }
    char id_str[5];
    sprintf(id_str, "%04d", studentID % 10000);
    for (int j = 0; j < 4; j++) {
        arr[5][j] = id_str[j];
    }
    arr[5][4] = 'Q';
}
void printArray(char arr[6][5]) {
    for (int i = 0; i < 6; i++) {
        for (int j = 0; j < 5; j++) {
            printf("%c\t", arr[i][j]);
        }
        printf("\n");
    }
}
int searchString(char arr[6][5], char *str) {
    int len = strlen(str);
    for (int i = 0; i < 6; i++) {
        for (int j = 0; j <= 5 - len; j++) {
            int found = 1;
            for (int k = 0; k < len; k++) {
                if (arr[i][j + k] != str[k]) {
                    found = 0;
                    break;
                }
            }
            if (found) return 1;
        }
    }

    for (int j = 0; j < 5; j++) {
        for (int i = 0; i <= 6 - len; i++) {
            int found = 1;
            for (int k = 0; k < len; k++) {
                if (arr[i + k][j] != str[k]) {
                    found = 0;
                    break;
                }
            }
            if (found) return 1;
        }
    }

    return 0;
}

int main() {
    srand(time(0));
    char arr[6][5];
    int studentID;

    printf("Enter the last 4 digits of your student ID: ");
    scanf("%d", &studentID);

    int score = 0;
    char input[100];

    while (1) {
        generateArray(arr, studentID);
        printf("\nGenerated 2D Array:\n");
        printArray(arr);

        while (1) {
                    printf("\nEnter a string to search (or 'END' to
regenerate): ");
            scanf("%s", input);

            if (strcmp(input, "END") == 0) break;

            if (searchString(arr, input)) {
                score++;
                printf("%s is present. Score: %d\n", input, score);
            } else {
                score--;
                      printf("%s is not present. Score: %d\n", input,
score);
```

```
            }
        }
    }

    return 0;
}
```

Question : 8

Write a C program that contains the following functions for different numerical conversions:

1. int BinaryToDecimal(int number); Converts a binary number to its decimal equivalent.
2. int DecimalToBinary(int number); Converts a decimal number to its binary equivalent.
3. void DecimalToHexadecimal(int number); Converts a decimal number to its hexadecimal equivalent and prints it.
4. void HexadecimalToDecimal(string hexNumber); Converts a hexadecimal number to its decimal equivalent and prints it.
5. void BinaryToHexadecimal(int number); Converts a binary number to its hexadecimal equivalent and prints it.
6. void HexadecimalToBinary(string hexNumber); Converts a hexadecimal number to its binary equivalent and prints it.

Each function should take an appropriate input and return or display the converted value. Ensure that the program handles invalid inputs gracefully. Your program must display the menu which function the user wants to call.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

// Function Prototypes
int BinaryToDecimal(int number);
int DecimalToBinary(int number);
void DecimalToHexadecimal(int number);
void HexadecimalToDecimal(char hexNumber[]);
void BinaryToHexadecimal(int number);
void HexadecimalToBinary(char hexNumber[]);
int isValidBinary(int number);
int isValidHexadecimal(char hexNumber[]);
```

```
//  Main  function  to  display  the  menu  and  call  the  appropriate
conversion function
int main() {
    int choice, decimalNumber, binaryNumber;
    char hexNumber[20];

    while (1) {
        printf("\n==== Numerical Conversion Menu ====\n");
        printf("1. Binary to Decimal\n");
        printf("2. Decimal to Binary\n");
        printf("3. Decimal to Hexadecimal\n");
        printf("4. Hexadecimal to Decimal\n");
        printf("5. Binary to Hexadecimal\n");
        printf("6. Hexadecimal to Binary\n");
        printf("7. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter a binary number: ");
                scanf("%d", &binaryNumber);
                if (isValidBinary(binaryNumber)) {
                                              printf("Decimal:  %d\n",
BinaryToDecimal(binaryNumber));
                } else {
                    printf("Invalid binary number!\n");
                }
                break;

            case 2:
                printf("Enter a decimal number: ");
                scanf("%d", &decimalNumber);
                                              printf("Binary:   %d\n",
DecimalToBinary(decimalNumber));
                break;

            case 3:
```

```c
                printf("Enter a decimal number: ");
                scanf("%d", &decimalNumber);
                printf("Hexadecimal: ");
                DecimalToHexadecimal(decimalNumber);
                printf("\n");
                break;

            case 4:
                printf("Enter a hexadecimal number: ");
                scanf("%s", hexNumber);
                if (isValidHexadecimal(hexNumber)) {
                    printf("Decimal: ");
                    HexadecimalToDecimal(hexNumber);
                } else {
                    printf("Invalid hexadecimal number!\n");
                }
                break;

            case 5:
                printf("Enter a binary number: ");
                scanf("%d", &binaryNumber);
                if (isValidBinary(binaryNumber)) {
                    printf("Hexadecimal: ");
                    BinaryToHexadecimal(binaryNumber);
                    printf("\n");
                } else {
                    printf("Invalid binary number!\n");
                }
                break;

            case 6:
                printf("Enter a hexadecimal number: ");
                scanf("%s", hexNumber);
                if (isValidHexadecimal(hexNumber)) {
                    printf("Binary: ");
                    HexadecimalToBinary(hexNumber);
                    printf("\n");
                } else {
                    printf("Invalid hexadecimal number!\n");
                }
                break;

            case 7:
                exit(0);

            default:
                printf("Invalid choice! Please try again.\n");
        }
    }
    return 0;
}

// Function to check if a number is a valid binary number
int isValidBinary(int number) {
    while (number != 0) {
        if (number % 10 > 1) return 0;  // Only 0 or 1 are allowed
        number /= 10;
    }
    return 1;
}

// Function to check if a string is a valid hexadecimal number
int isValidHexadecimal(char hexNumber[]) {
    for (int i = 0; hexNumber[i] != '\0'; i++) {
        if (!((hexNumber[i] >= '0' && hexNumber[i] <= '9') ||
              (hexNumber[i] >= 'A' && hexNumber[i] <= 'F') ||
              (hexNumber[i] >= 'a' && hexNumber[i] <= 'f'))) {
            return 0;  // Invalid character found
        }
    }
    return 1;
}

// Function to convert a binary number to decimal
int BinaryToDecimal(int number) {
    int decimal = 0, base = 1;
    while (number > 0) {
        int lastDigit = number % 10;
        decimal += lastDigit * base;
        base *= 2;
        number /= 10;
    }
    return decimal;
}

// Function to convert a decimal number to binary
int DecimalToBinary(int number) {
    int binary = 0, place = 1;
    while (number > 0) {
        int remainder = number % 2;
        binary += remainder * place;
        place *= 10;
        number /= 2;
    }
    return binary;
}

// Function to convert a decimal number to hexadecimal
void DecimalToHexadecimal(int number) {
    char hex[20];
    int index = 0;

    while (number > 0) {
        int remainder = number % 16;
        if (remainder < 10) {
            hex[index++] = '0' + remainder;
        } else {
            hex[index++] = 'A' + (remainder - 10);
        }
        number /= 16;
    }

    // Print the hexadecimal number in reverse order
    for (int i = index - 1; i >= 0; i--) {
        printf("%c", hex[i]);
    }
}
```

```c
// Function to convert a hexadecimal number to decimal
void HexadecimalToDecimal(char hexNumber[]) {
    int length = strlen(hexNumber);
    int base = 1;  // 16^0
    int decimal = 0;

    for (int i = length - 1; i >= 0; i--) {
        if (hexNumber[i] >= '0' && hexNumber[i] <= '9') {
            decimal += (hexNumber[i] - '0') * base;
        } else if (hexNumber[i] >= 'A' && hexNumber[i] <= 'F') {
            decimal += (hexNumber[i] - 'A' + 10) * base;
        } else if (hexNumber[i] >= 'a' && hexNumber[i] <= 'f') {
            decimal += (hexNumber[i] - 'a' + 10) * base;
        }
        base *= 16;
    }
    printf("%d\n", decimal);
}

// Function to convert a binary number to hexadecimal
void BinaryToHexadecimal(int number) {
    int decimal = BinaryToDecimal(number);
    DecimalToHexadecimal(decimal);
}

// Function to convert a hexadecimal number to binary
void HexadecimalToBinary(char hexNumber[]) {
    int decimal = 0;

    // Convert hex to decimal first
    for (int i = 0; hexNumber[i] != '\0'; i++) {
        if (hexNumber[i] >= '0' && hexNumber[i] <= '9') {
            decimal = decimal * 16 + (hexNumber[i] - '0');
        } else if (hexNumber[i] >= 'A' && hexNumber[i] <= 'F') {
            decimal = decimal * 16 + (hexNumber[i] - 'A' + 10);
        } else if (hexNumber[i] >= 'a' && hexNumber[i] <= 'f') {
            decimal = decimal * 16 + (hexNumber[i] - 'a' + 10);
        }
    }

    // Convert decimal to binary
    printf("%d", DecimalToBinary(decimal));
}
```

Question : 9

Write a C program to simulate a simple grid-based adventure game using a 2D array.

Game Description:

The game consists of a 5x5 grid.

Each cell can contain:

- o An empty space (' ').
- o An item ('I') that the player can collect.
- o An obstacle ('X') that the player cannot pass through.
- o The player's position, represented by 'P'.

Example Input:

char grid[5][5] = {

{' ', ' ', 'I', 'X', ' '},

{' ', 'X', ' ', ' ', ' '},

{'I', ' ', 'X', 'X', ' '},

{' ', ' ', ' ', 'I', 'X'},

{' ', 'X', ' ', ' ', 'P'}

};

Requirements:

1. Display the grid and prompt the player for a move (W: up, S: down, A: left, D: right, Q: quit).
2. Update the player's position based on valid moves.
3. Collect items when the player moves to a cell containing an item (remove item from grid).
4. Prevent movement into obstacles ('X').
5. Continue until the player chooses to quit.
6. The grid is fixed at 5x5.
7. Only valid moves are allowed.
8. Handle invalid input gracefully.

```
----------------------------------------------------------------------
#include <stdio.h>
void displayGrid(char grid[5][5]);
void  movePlayer(char  grid[5][5],  int  *playerX,  int  *playerY,  char
direction, int *itemsCollected);
int isValidMove(char grid[5][5], int newX, int newY);

int main() {
    char grid[5][5] = {
        {' ', ' ', 'I', 'X', ' '},
        {' ', 'X', ' ', ' ', ' '},
        {'I', ' ', 'X', 'X', ' '},
        {' ', ' ', ' ', 'I', 'X'},
        {' ', 'X', ' ', ' ', 'P'}
    };

    int playerX = 4, playerY = 4;
    int itemsCollected = 0;
    char input;
    printf("Welcome to the Grid Adventure Game!\n");
      printf("Controls:  W  (up),  S  (down),  A  (left),  D  (right),  Q
(quit)\n\n");

    while (1) {
        displayGrid(grid);
        printf("Items Collected: %d\n", itemsCollected);
        printf("Enter your move: ");
        scanf(" %c", &input);
        if (input == 'Q' || input == 'q') {
               printf("Thanks  for  playing!  You  collected  %d  items.\n",
itemsCollected);
            break;
        }
        movePlayer(grid, &playerX, &playerY, input, &itemsCollected);
    }
    return 0;
}
void displayGrid(char grid[5][5]) {
    printf("\n");
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 5; j++) {
            printf("%c ", grid[i][j]);
        }
        printf("\n");
    }
    printf("\n");
}
void  movePlayer(char  grid[5][5],  int  *playerX,  int  *playerY,  char
direction, int *itemsCollected) {
    int newX = *playerX, newY = *playerY;
    if (direction == 'W' || direction == 'w') newX--;
    else if (direction == 'S' || direction == 's') newX++;
    else if (direction == 'A' || direction == 'a') newY--;
    else if (direction == 'D' || direction == 'd') newY++;
    else {
        printf("Invalid input! Use W, A, S, D to move.\n");
        return;
    }

    if (isValidMove(grid, newX, newY)) {
        if (grid[newX][newY] == 'I') {
            (*itemsCollected)++;
            printf("You collected an item!\n");
        }
            grid[*playerX][*playerY]  =  '  ';   //  Clear  the  previous
position
        *playerX = newX;
        *playerY = newY;
        grid[newX][newY] = 'P';  // Update the new position
    } else {
```

```c
        printf("Invalid move! You can't move there.\n");
    }
}
int isValidMove(char grid[5][5], int newX, int newY) {
      return  (newX  >=  0  &&  newX  <  5  &&  newY  >=  0  &&  newY  <  5  &&
grid[newX][newY] != 'X');
}
```
