|  |  |
| --- | --- |
| **Department of Computer Science**  **Bahria University** | **CSC-221: Data structures & Algorithms**  **Semester03 (Spring 2024)** |



**ASSIGNMENT 02**

Marks: 05

Submission Date: 25-March-2024

Abdullah

# NAME:

BS (CS) 3A

# CLASS: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

02-131222-099

# REG #: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Marks Obtained:**

**Instructions.**

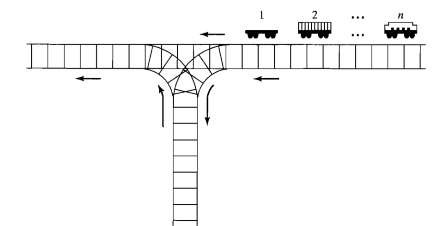
1. Follow same format for assignment submission.

1. Copied/Plagiarized answers will be marked zero.

1. Output must be attached with the code

**Scenario CLO4, PLO3, C6**

Consider the following railroad switching network:



Railroad cars numbered 1,2, ..., n on the right track are to be permuted and moved along on the left track. A car may be moved directly onto the left track, or it may be shunted onto the siding to be removed at a later time and placed on the left track. The siding thus operates like a stack, a push operation moving a car from the right track onto the siding and a pop operation moving the "top" car from the siding onto the left track.

1. For *n = 3*, find all possible permutations of cars that can be obtained (on the left track) by a sequence of these operations. For example, push 1, push 2, move 3, pop 2, pop 1 arranges them in the order 3, 2,1. Are any permutations not possible?

1. Find all possible permutations for n = 4. What permutations (if any) are not possible?

1. Repeat (b) for *n = 5.*

**Question**

1. **Design** an algorithm for the above scenario. (2 marks)

1. **Create** a code in C++ that implements the above scenario. Test your program as the value of n mentioned above. (3 marks)

**Solution:**

**Algorithm:**

1. **Function Definitions:**

- Define functions `pushToSiding`, `popFromSiding`, `top`, and `generatePermutations` to manage the cars on the siding and tracks.

- `pushToSiding`: Adds a car to the siding.

- `popFromSiding`: Removes a car from the siding.

- `top`: Retrieves the top car from the siding.

- `generatePermutations`: Uses recursion to create all possible arrangements of cars.

2. **Permutation Generation:**

Begin the `generatePermutations` function with the following parameters:

- `rightTrack[]`: Array representing cars initially on the right track.

- `leftTrack[]`: Array representing cars on the left track.

- `numRight`: Number of cars on the right track.

- `numLeft`: Number of cars on the left track.

- `siding[]`: Array representing cars on the siding.

- `sidingTop`: Index indicating the top of the siding stack.

**Base Case:**

- If there are no cars left on both tracks and the siding, print the arrangement of cars on the left track.

**Recursive Cases:**

- If there are cars remaining on the right track, push a car onto the siding, recurse, then restore the arrangement.

- If there are cars on the siding, pop a car onto the left track, recurse, then restore the arrangement.

3. **Main Function:**

- Prompt the user to input the number of cars (n).

- Validate the input to ensure it's within a valid range.

- Call the `findPossibleArrangements` function with the user's input.

4. **Arrangement Count:**

- Inside `findPossibleArrangements`, initialize arrays and variables needed for permutation generation.

- Call the `generatePermutations` function to create all possible arrangements.

- Print the total count of arrangements generated.

**Code:**

#include <iostream>

using namespace std;

const int MAX\_CARS = 10;

void pushToSiding(int car, int siding[], int& ss) {

    siding[++ss] = car;

}

void popFromSiding(int siding[], int& ss) {

    if (ss >= 0) {

        ss--;

    }

}

int top(const int siding[], int ss) {

    if (ss >= 0) {

        return siding[ss];

    }

    return -1;

}

void generatePermutations(int rightTrack[], int leftTrack[], int& rs, int& ls, int siding[], int& ss, int& totalPermutations) {

    if (rs == 0 && ss == -1) {

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

            cout << leftTrack[i] << " ";

        }

        cout << endl;

        totalPermutations++;

        return;

    }

    if (rs > 0) {

        pushToSiding(rightTrack[--rs], siding, ss);

        generatePermutations(rightTrack, leftTrack, rs, ls, siding, ss, totalPermutations);

        rightTrack[rs++] = top(siding, ss);

        popFromSiding(siding, ss);

    }

    if (ss != -1) {

        leftTrack[ls++] = top(siding, ss);

        popFromSiding(siding, ss);

        generatePermutations(rightTrack, leftTrack, rs, ls, siding, ss, totalPermutations);

        pushToSiding(leftTrack[--ls], siding, ss);

    }

}

void findPossiblePermutations(int n) {

    int rightTrack[MAX\_CARS];

    int siding[MAX\_CARS];

    int leftTrack[MAX\_CARS];

    int totalPermutations = 0;

    for (int i = 0; i < n; ++i) {

        rightTrack[i] = i + 1;

    }

    int rs = n;

    int ls = 0;

    int ss = -1;

    generatePermutations(rightTrack, leftTrack, rs, ls, siding, ss, totalPermutations);

    cout << "Total permutations: " << totalPermutations << endl;

}

int main() {

    int num;

    cout << "Enter the number of cars (maximum " << MAX\_CARS << "): ";

    cin >> num;

    if (num <= 0 || num > MAX\_CARS) {

        cout << "Please enter a number between 1 and " << MAX\_CARS << "." << endl;

        return 0;

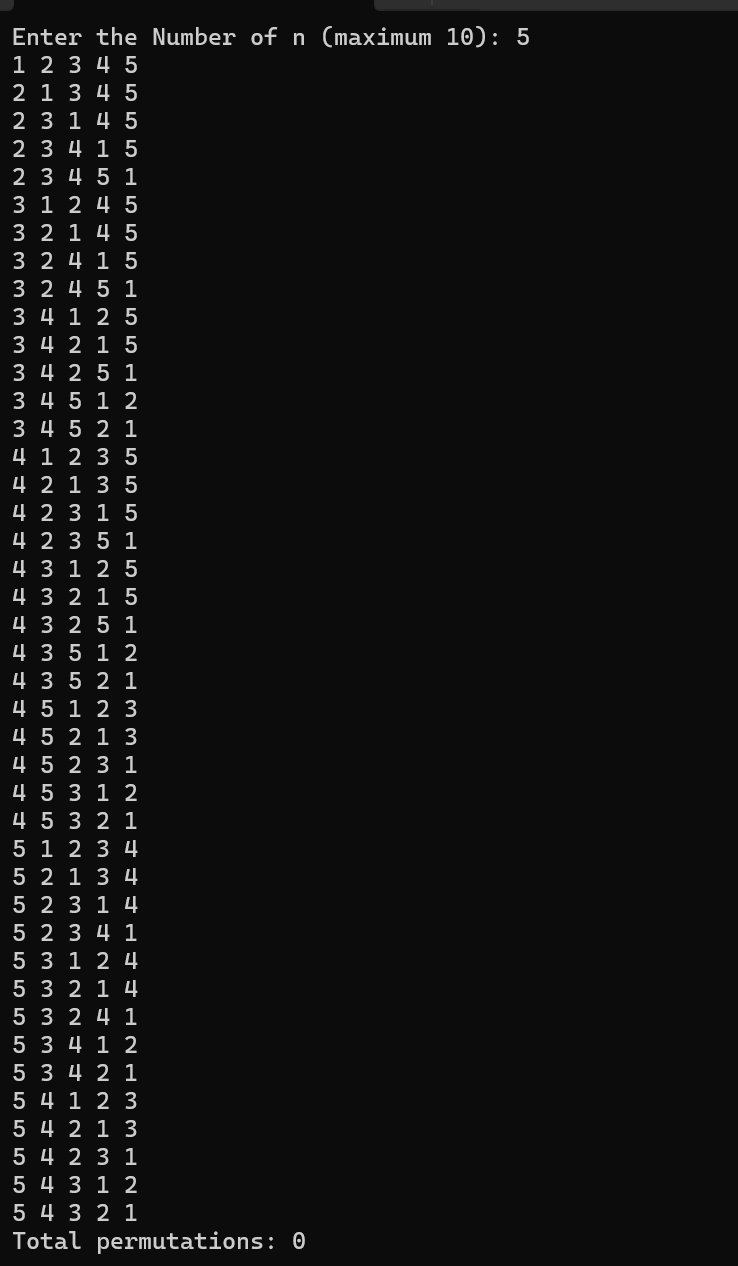
    }

    findPossiblePermutations(num);

    return 0;

}

**Output:**

****

**A screenshot of a computer

Description automatically generated**