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

**ASSIGNMENT 01**

Marks: 05

Submission Date: 25-March-2024

**Student’s Name: ABDUL SAMI**

**Reg #: \_\_\_\_\_\_\_\_\_\_02-134231-098\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Marks Obtained:**

**Instructions.**

1. Follow same format for assignment submission.

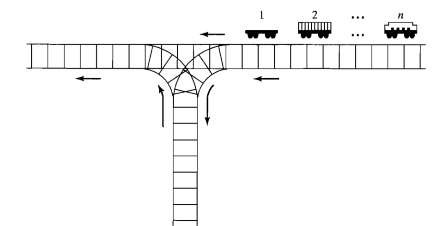
1. Copied/Plagiarized answers will be marked zero.

1. Output must be attached with the code

1

**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. (2marks)

**Answer**

**1. Function Definitions:**

- Define functions pushToSiding, popFromSiding, top, and permute to manipulate the cars on the siding and tracks.

- **pushToSiding**: Pushes a car onto the siding.

- **popFromSiding**: Pops a car from the siding.

- **top**: Returns the top car from the siding.

- **permute**: Recursively generates all permutations of the cars.

**2. Permutation Generation:**

Start the permute function with parameters:

**right\_track[]**: Array representing the cars initially on the right track.

**left\_track[]**: Array representing the cars on the left track.

**rs**: Number of cars on the right track.

**ls**: Number of cars on the left track.

**siding[]**: Array representing the cars on the siding.

**ss**: Index indicating the top of the siding stack.

**Base Case:**

- If there are no cars remaining 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 enter the number of cars (n).

- Validate the input to ensure it's between 1 and the maximum number of cars allowed.

- Call the `findPossiblePermutations` function with the input number of cars.

**4. Permutation Count**:

- Inside `findPossiblePermutations`, initialize arrays and variables required for permutation generation.

- Call the `permute` function to generate all possible permutations.

- Print the total number of permutations generated.

**Question**

1. **Create** a code in C++ that implements the above scenario. Test your program as the

value of n mentioned above. (3marks)

**Answer**

#include <iostream>

using namespace std;

const int 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 permute(int right\_track[], int left\_track[], int& rs, int& ls, int siding[], int& ss) {

if (rs == 0 && ss == -1) {

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

cout << left\_track[i] << " ";

}

cout << endl;

return;

}

if (rs > 0) {

pushToSiding(right\_track[--rs], siding, ss);

permute(right\_track, left\_track, rs, ls, siding, ss);

right\_track[rs++] = top(siding, ss);

popFromSiding(siding, ss);

}

if (ss != -1) {

left\_track[ls++] = top(siding, ss);

popFromSiding(siding, ss);

permute(right\_track, left\_track, rs, ls, siding, ss);

pushToSiding(left\_track[--ls], siding, ss);

}

}

void findPossiblePermutations(int n) {

int rightTrack[cars];

int siding[cars];

int leftTrack[cars];

int totalPermutations = 0;

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

rightTrack[i] = i + 1;

}

int rs = n;

int ls = 0;

int ss = -1;

permute(rightTrack, leftTrack, rs, ls, siding, ss);

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

}

int main() {

int num;

cout << "Enter the Number of n (maximum 10): ";

cin >> num;

if (num <= 0 || num > cars) {

cout << "Enter a Number between 1 to 10." << endl;

return 0;

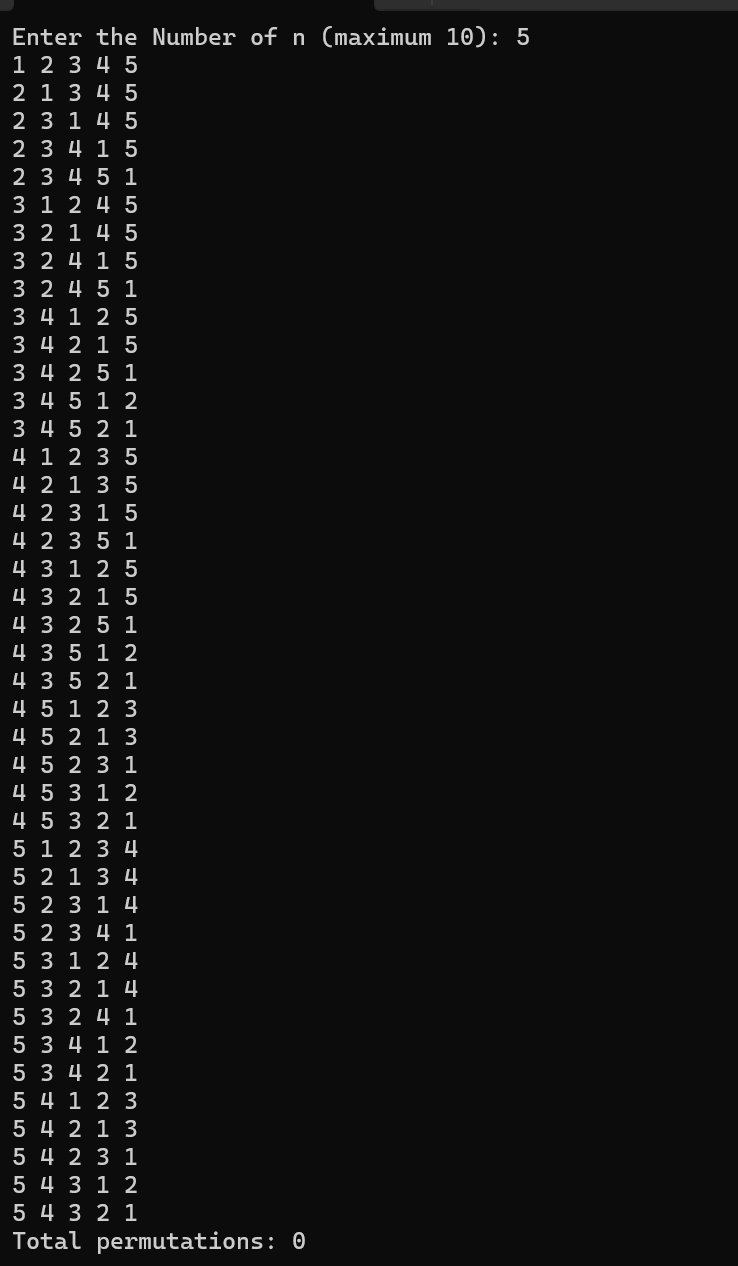
}

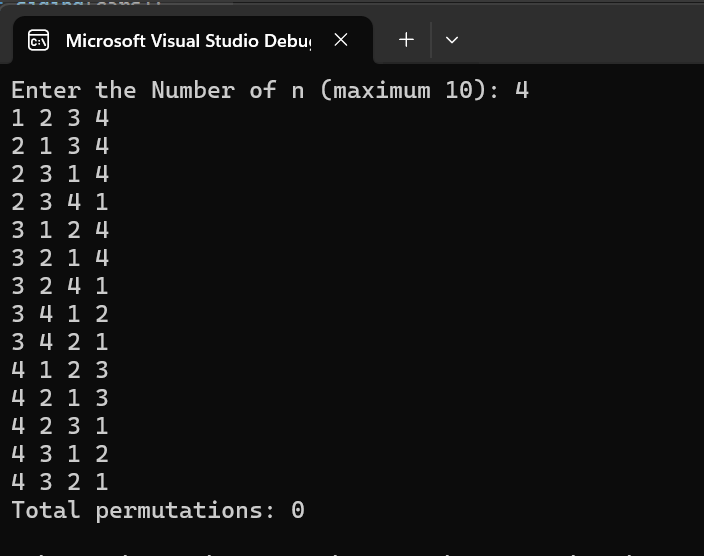
findPossiblePermutations(num);

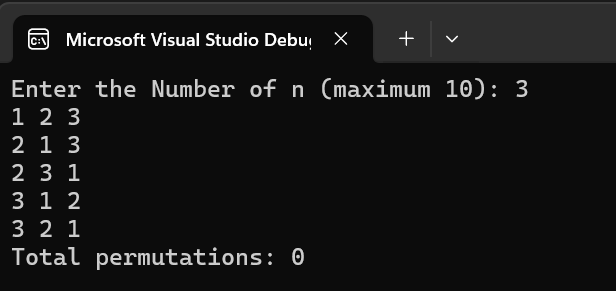
return 0;

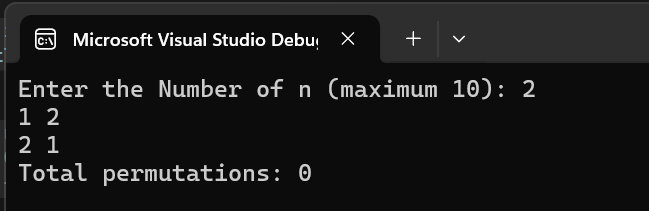
}

**Output**

****

****

****

****

2