Experiment - 2

Student Name: Gaganjot Singh **UID:22BCS14843**
Branch: BE-CSE **Section/Group:**22BCS_JT_ 802-B
Semester: 5th **Date of Performance:**2 August 2024
Subject Name: Advanced Programming Lab **Subject Code:**22CSP-314

Question 2.1 - Equal Stack:

1. **Aim:** You have three stacks of cylinders where each cylinder has the same diameter,but they may vary in height. You can change the height of a stack by removing and discarding its topmost cylinder any number of times. Find the maximum possible height of the stacks such that all of the stacks are exactly the same height. This meansyou must remove zero or more cylinders from the top of zero or more of the three stacks until they are all the same height, then return the height.

2. **Objective:**
The objective of the code is to understand stack manipulation and array handling bycalculating the maximum possible height where three stacks can be made equal by removing cylinders from the top of the stacks in C++, including implementing a method to adjust the heights until all stacks are equal.

3. **Algorithm:** Start Initialization:

- Input: Three stacks of cylinders represented as arrays of integers.Output: Maximum possible
- equal height of the three stacks after removingcylinders from the top.
- User Interaction:
  - Prompt the user to enter the number of cylinders in each stack.
  - Read the number of cylinders into variables n1, n2, and n3.
  - Initialize arrays h1, h2, and h3 of sizes n1, n2, and n3 respectively.
  - Prompt the user to enter the heights of the cylinders for each stack.
  - Read each height into the respective arrays h1, h2, and h3.Height
- Calculation:
  - Initialize three variables sum1, sum2, and sum3 to store the total height of eachstack.
  - For each height in arrays h1, h2, and h3, add the height to sum1, sum2, andsum3 respectively.
- Height Equalization:
  - Initialize three indices i1, i2, and i3 to 0, representing the top cylinder of eachstack.
  - While the heights sum1, sum2, and sum3 are not equal:
    - If sum1 is the highest, remove the top cylinder from h1 by subtracting itsheight from sum1 and increment i1.
    - If sum2 is the highest, remove the top cylinder from h2 by subtracting itsheight from sum2 and increment i2.
    - If sum3 is the highest, remove the top cylinder from h3 by subtracting itsheight from sum3 and increment i3.
- Output:

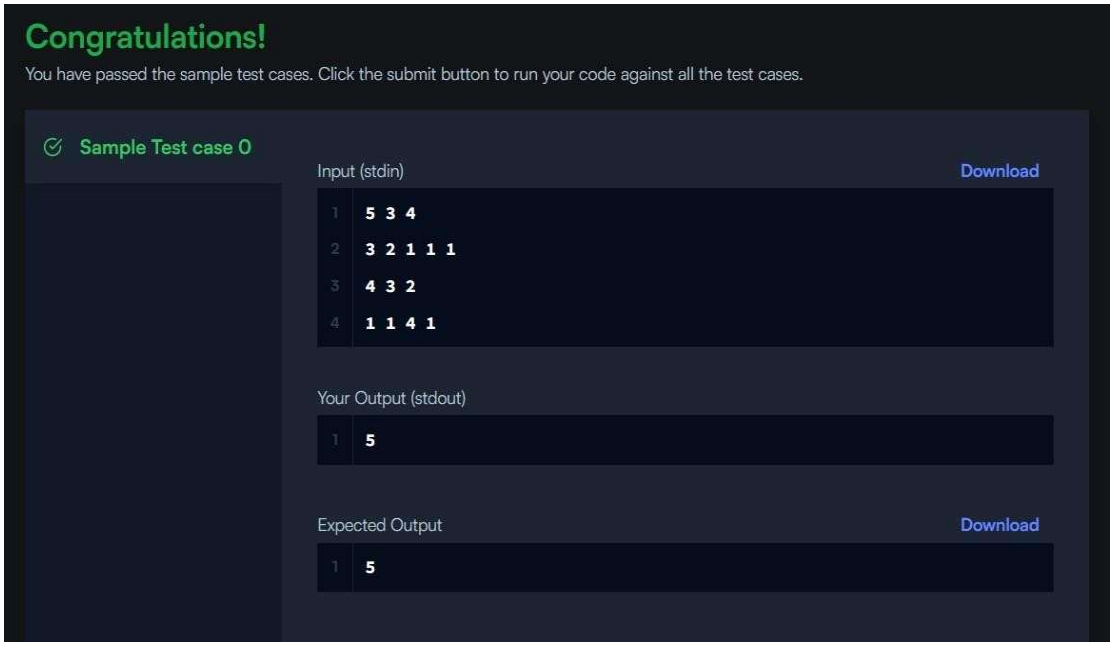
- Once the heights are equal, output the value of sum1, sum2, or sum3 as themaximum possible equal height.

4. Implementation/Code:

```
#include <iostream>
#include <vector>

using namespace std;

int equalStacks(vector<int>& h1, vector<int>& h2, vector<int>& h3) {int sum1 = 0,
    sum2 = 0, sum3 = 0;

    for (int height : h1) sum1 += height; for
    (int height : h2) sum2 += height; for (int
    height : h3) sum3 += height;

    int i1 = 0, i2 = 0, i3 = 0;

    while (!(sum1 == sum2 && sum2 == sum3)) {if (sum1
        >= sum2 && sum1 >= sum3) {
            sum1 -= h1[i1++];
        } else if (sum2 >= sum1 && sum2 >= sum3) {sum2 -=
            h2[i2++];
        } else if (sum3 >= sum1 && sum3 >= sum2) {sum3 -=
            h3[i3++];
        }
    }

    return sum1;
}

int main() {
    int n1, n2, n3;
    cin >> n1 >> n2 >> n3;

    vector<int> h1(n1);
    for (int i = 0; i < n1; ++i) {cin
        >> h1[i];
    }

    vector<int> h2(n2);
    for (int i = 0; i < n2; ++i) {cin
        >> h2[i];
    }
```

```
vector<int> h3(n3);
for (int i = 0; i < n3; ++i) {cin
    >> h3[i];
}
cout << equalStacks(h1, h2, h3) << endl;return 0;
 }
```

5. Output:



6. Time Complexity:

The time complexity of the **equalStacks** method is O(n1+n2+n3), where n1, n2, and n3 are the lengths of the three input arrays. This is because both the initial sumcalculations and the while loop may iterate through all elements of each array.

7. Learning Outcomes:

- I understood how to equalize the heights of three stacks by removing elementsfrom the top in C++.
- I learned to initialize and manage multiple vectors and populate them with userinput.
- I practiced performing array manipulations by iterating through elements tocompute and adjust sums.
- I became familiar with handling user input for multiple vectors and displayingthe results interactively.
- I gained experience in implementing stack operations and managing timecomplexity efficiently.

Question 2 - Queues: A Tale of Two Stacks:

1. **Aim:** A queue is an abstract data type that maintains the order in which elements were added to it, allowing the oldest elements to be removed from the front and newelements to be added to the rear. This is called a First-In-First-Out (FIFO) data structure because the first element added to the queue (i.e., the one that has been waiting the longest) is always the first one to be removed.

2. **Objective:**
The objective of the code is to manage a queue using a LinkedList, handling enqueue,dequeue, and front-element queries efficiently based on user input.

3. **Algorithm:**

i. Set up initial data:

- Create a std::deque to represent the queue.
- Use cin to read input.

ii. Read the number of queries:

- Process each query based on its type.

iii. Handle each query:

- If the query type is 1, add the value to the rear of the queue using push_back.
- If the query type is 2, remove the element from the front of the queue usingpop_front if the queue is not empty.
- If the query type is 3, print the front element of the queue using front if thequeue is not empty.

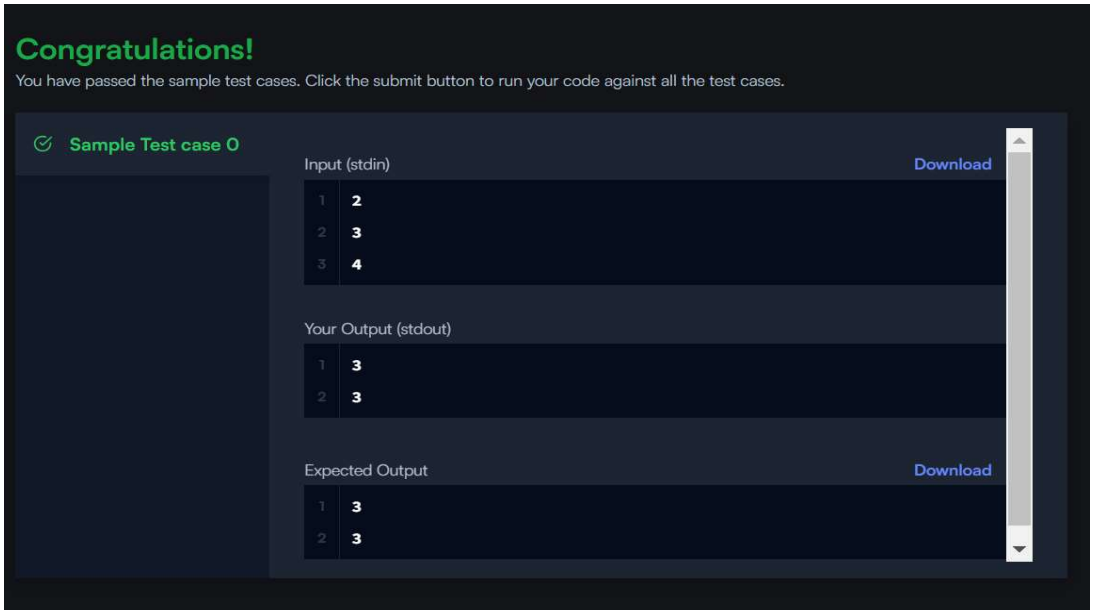
iv. **Finish processing all queries**.

4. **Implementation/Code:**

```
#include <iostream>
#include <deque> using
namespace std;int
main() {
    deque<int> queue;int
    n;
    cin >> n;
    for (int i = 0; i < n; i++) {int
        queryType;
        cin >> queryType;

        if (queryType == 1) {int
            value;
            cin >> value;
```

```
            queue.push_back(value);
        } else if (queryType == 2) {if
            (!queue.empty()) {
                queue.pop_front();
            }
        } else if (queryType == 3) {if
            (!queue.empty()) {
                cout << queue.front() << endl;
            }
        }
    }

    return 0;
}
```

5. Output:



6. Time Complexity:

The time complexity for each query operation in the code is O(1). The overall complexity depends on the number of queries, making it O(n), where n is the numberof queries.

7. Learning Outcomes:

- I learned how to implement a queue in C++ using std::deque to manage differentoperations based on user queries.
- I practiced efficiently handling queue operations such as enqueueing, dequeueing,and accessing the front element.
- I gained experience in reading and processing user input to dynamically performqueue operations.

- I improved my skills in managing and outputting results for various query types whilemaintaining the FIFO order of the queue.
- I understood how to handle multiple queries and perform operations on a queue withconsistent time complexity