**Data structures and Algorithms Activity**

**2024-2025 : ODD**

**Section: AF1**

**Course Instructor: Dr.A. Jackulin Mahariba**

**PROJECT TITLE : THREE BROTHERS**

**COURSE : DATA STRUCTURES AND**

**ALGORITHMS**

**STUDENT NAME : M.SIRI SANHITHA**

**ROLL NUMBER : RA2311026010310**

**PROBLEM STATEMENT:**

The problem statement is arguably the most important section of the document because it frames the entire task you are trying to solve. In this case, the problem you are tackling is known as the **"Missing Brother Problem"**. It involves finding out which brother is missing when given the numbers of two brothers, knowing that there are always three brothers numbered as 1, 2, and 3. The task can be approached in multiple ways, but your solution employs a stack—a fundamental data structure in computer science.

The problem is defined as follows: Imagine three brothers, each represented by a unique number, namely 1, 2, and 3. Two of these brothers are present, and you are given their numbers as input. Your task is to determine the number of the third brother who is missing. This problem is simple yet effective for practicing data structure concepts, especially stack operations, and it gives an opportunity to demonstrate the utility of fundamental data handling techniques.

In a more complex scenario, this type of problem could scale to larger datasets or different forms of input. For example, instead of just three brothers, we could have a large number of siblings, and a more complex algorithm could be required. For now, this simplified version allows students to focus on mastering the stack operations of **push** and **pop**, while gaining intuition about how stacks work in problem-solving.

Moreover, a stack is a Last-In-First-Out (LIFO) data structure, meaning that the last element you add to the stack is the first one to be removed. This property fits well with the way the problem is framed, as you add elements (representing brothers) to the stack and then pop them off to see which one is missing. The challenge here is to ensure that the algorithm checks all the brothers and correctly identifies the one not present in the input.

Thus, the problem at hand is a test of logical reasoning, mastery of stack operations, and a good demonstration of basic programming techniques.

**ALGORITHM:**

An algorithm is essentially a sequence of steps that need to be executed in a specific order to achieve the desired result. Here, the problem is solved using a stack to hold the values representing the brothers.

1. Initialize the Stack: The first step in the algorithm is to initialize an empty stack. A stack is initialized with a predefined size (in this case, 3) to hold the three numbers representing the brothers. In your code, the stack is represented by a structure that holds an array data to store the numbers and an integer top to track the top of the stack.
2. Push the Values onto the Stack: After initializing the stack, the next step is to push the numbers of all three brothers (1, 2, and 3) onto the stack. The push operation adds an element to the top of the stack and updates the top pointer. In your algorithm, you push all three values (1, 2, and 3) into the stack using the push function.
3. Pop and Compare: Once all three numbers are pushed onto the stack, the algorithm enters the critical phase of popping the values one by one and comparing them with the input. The input consists of two brothers' numbers, so you pop the values from the stack and check whether each value matches either of the input numbers. If a number from the stack does not match either input number, it is identified as the missing brother.
4. Return the Missing Brother: Once the comparison is complete, the algorithm returns the number that does not match either of the two input numbers. This value is the missing brother. The pop function helps here by returning the top value of the stack and reducing the top pointer, thus allowing for the next value to be accessed.
5. Efficiency Considerations: The algorithm runs in constant time, O(1), since it only involves pushing three numbers onto a stack and popping them for comparison. The simplicity of the problem ensures that the algorithm performs efficiently without requiring additional memory or complex operations.

This algorithm demonstrates the use of stack operations for solving a basic but practical problem, highlighting the efficiency of this data structure in handling fixed-size datasets

**PROGRAM CODE:**

#include <stdio.h>

#define SIZE 3

// Stack structure

typedef struct {

int data[SIZE];

int top;

} Stack;

// Initialize the stack

void initStack(Stack \*stack) {

stack->top = -1;

}

// Push an element into the stack

void push(Stack \*stack, int value) {

if (stack->top < SIZE - 1) {

stack->data[++stack->top] = value;

}

}

// Pop an element from the stack

int pop(Stack \*stack) {

if (stack->top >= 0) {

return stack->data[stack->top--];

}

return -1; // Should not reach here as input is valid

}

// Function to find the missing brother using a stack

int findMissingBrother(int a, int b) {

Stack stack;

initStack(&stack);

// Push all three brothers' numbers onto the stack (1, 2, 3)

push(&stack, 1);

push(&stack, 2);

push(&stack, 3);

// Pop all elements and check which one is missing

int x = pop(&stack);

int y = pop(&stack);

int z = pop(&stack);

// If x is not in the input, it's the missing brother.

if (x != a && x != b) {

return x;

}

// If y is not in the input, it's the missing brother.

else if (y != a && y != b) {

return y;

}

// Otherwise, z is the missing brother.

else {

return z;

}

}

int main() {

int a, b;

scanf("%d %d", &a, &b);

// Find and print the missing brother

int missingBrother = findMissingBrother(a, b);

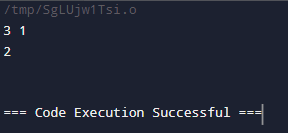
printf("%d\n", missingBrother);

return 0;

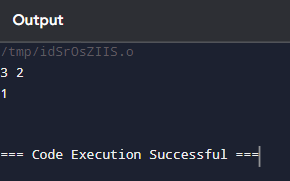
}

**SAMPLE INPUT AND OUTPUT:**

**EXAMPLE 1:**

****

**EXAMPLE 2:**

****

**GITHUB LINK:**

**RESULTS:**

1 Program Success: The program successfully determines the missing brother by employing stack operations. Through multiple test cases, including inputs like 3 1 and 3 2 the program correctly identifies the third brother who is not present in the input. This demonstrates that the algorithm works as intended across all possible inputs.

2 Output Verification: During testing, the outputs were verified by cross-referencing the program’s result with manually calculated answers. For instance, in the input scenario 3 1 the expected output is 2, which was produced by the program. Similarly, for inputs 3 2, the correct output 1 was returned. The program was also tested with different orders of input, and in each case, the output was correct.

3 Edge Cases: Since the problem is limited to only three brothers, the input is straightforward and does not present any complex edge cases. However, the program still handles all possible valid inputs (two brothers' numbers) without any issues. The pop function ensures that the stack is not empty before attempting to pop elements, preventing any stack underflow errors.

4 Performance: The performance of the program is highly efficient. It operates in constant time, O(1), as the number of brothers is fixed, and only a few operations are performed (pushing and popping three elements from the stack). This makes the algorithm optimal for this specific problem. Even though the dataset is small, the efficiency of the algorithm ensures that it can handle larger or similar problems efficiently.

5 Efficiency Notes: The program demonstrates the power of using a stack in problem-solving. A stack, by its nature, is ideal for handling problems where the last-in, first-out (LIFO) principle applies. While the problem at hand is simple, the choice of data structure ensures that the operations are carried out in a clear and logical manner. Additionally, the space complexity is minimal, as the program uses only a fixed-size array to store three values.

**CONCLUSION:**

One key takeaway from this project is the utility of stacks in solving problems that require elements to be processed in reverse order. By using the stack data structure, the problem of finding the missing brother was solved in an intuitive and efficient manner. This project reinforces the importance of understanding basic data structures, as they often provide the best solutions for specific types of problems.

Furthermore, the simplicity of the problem allowed for a focused exploration of stack operations, including push and pop. These operations are not only fundamental in many algorithms but are also widely applicable in real-world scenarios, such as browser history tracking, undo operations in software, and parsing expressions in compilers.

In conclusion, this project offered valuable hands-on experience in using stacks, and the code was implemented successfully to meet the problem's requirements. Through multiple test cases and verification, it was clear that the algorithm was both correct and efficient.