K. J. Somaiya College of Engineering, Mumbai-77
(A Constituent college of Somaiya Vidyavihar University)

Batch:B2
Roll No: 16010122151

Experiment No. 1

Grade: AA / AB / BB / BC / CC / CD /DD

Title: Implementation of Abstract Data Type

Objective: Implementation of ADT without using any standard library function

Expected Outcome of Experiment:

CO	Outcome
CO 1	Explain the different data structures used in problem solving.

Books/ Journals/ Websites referred:

Abstract:-

(Define ADT. Why are they important in data structures?)

Ans-Abstract Data Types (ADTs) stores data

and allow various operations on the data to

access and change it.

- A mathematical model, together with various operations defined on the model
- An ADT is a collection of data and associated operations for manipulating that data

ADTs provide a way to encapsulate data and operations into a single unit, making it easier to manage and modify the data structure.

Abstract Data Type for Rational Number

Value Definition- abstract TypeDef<integer,integer>RATIONALType

Operator Definition-

- abstract RATIONALType makerational<a,b>
- abstract RATIONALtype add<a,b>
- abstract RATIONALType mult<a, b>
- abstract ReturnType Equal<a,b>

[for chosen data type write value definition and operator definition)

Implementation Details:

1. Explain the Importance of the approach followed by you

Ans-The approach followed in the C program to handle rational numbers using abstract data types is beneficial for several reasons:

1. **Modularity and Reusability**: By creating a `Rational` struct and defining functions for arithmetic operations, we encapsulate the logic related to rational numbers within a single unit. This modular approach promotes reusability and allows us to use the same code for handling rational numbers in different parts of the program or in other programs.

2. **Abstraction**: Abstract data types hide the implementation details of data structures and expose only the essential functionalities to the user. In this case, the `Rational` struct abstracts the representation of rational numbers, allowing users to interact with them using intuitive functions like `add`, `subtract`, `multiply`, and `divide`, without needing to know the internal implementation.

3. **Readability and Maintainability**: By organizing the code with abstract data types and functions, the program becomes more readable and easier to maintain. It separates concerns and promotes a clean, understandable code structure.

4. **Data Integrity**: Using abstract data types, we can enforce rules and constraints on the data. In this case, we can ensure that rational numbers are always represented as proper fractions with a non-zero denominator, avoiding potential issues that might arise if improper fractions or zero denominators were used.

5. **Code Clarity**: The use of abstract data types and functions makes the code more concise and easier to understand. The main function focuses on high-level logic, while the low-level details of rational number manipulation are abstracted into separate functions.

6. **Error Handling**: Abstract data types allow us to handle potential errors or edge cases in a controlled manner. For example, we can check for division by zero when performing the `divide` operation and handle it gracefully.

7. **Flexibility and Extensibility**: The approach taken allows for easy expansion and modification of the program. If additional functionality is required or improvements need to be made, it can be done within the relevant functions, without affecting other parts of the code.

Overall, using abstract data types to handle rational numbers in this C program provides a well-structured, readable, and maintainable codebase, while encapsulating the complexity of handling rational numbers in a clean and modular manne

Program code and Output screenshots:

Code-

```
#include <stdio.h>

typedef struct {
    int numerator;
    int denominator;
} Rational;

Rational createRational(int num, int den) {
    Rational rational;
    rational.numerator = num;
    rational.denominator = den;
    return rational;
}

Rational add(Rational a, Rational b) {
    Rational result;
    result.numerator = a.numerator * b.denominator + b.numerator * a.denominator;
    result.denominator = a.denominator * b.denominator;
    return result;
}

Rational subtract(Rational a, Rational b) {
```

```
    Rational result;
    result.numerator = a.numerator * b.denominator - b.numerator * a.denominator;
    result.denominator = a.denominator * b.denominator;
    return result;
}

Rational multiply(Rational a, Rational b) {
    Rational result;
    result.numerator = a.numerator * b.numerator;
    result.denominator = a.denominator * b.denominator;
    return result;
}

Rational divide(Rational a, Rational b) {
    Rational result;
    result.numerator = a.numerator * b.denominator;
    result.denominator = a.denominator * b.numerator;
    return result;
}

void printRational(Rational r) {
    printf("%d / %d\n", r.numerator, r.denominator);
}

int main() {
```

```
int num1, den1, num2, den2;

printf("Enter the numerator and denominator of Rational Number 1 (e.g., 2 3): ");
scanf("%d %d", &num1, &den1);

printf("Enter the numerator and denominator of Rational Number 2 (e.g., 5 7): ");
scanf("%d %d", &num2, &den2);

Rational rational1 = createRational(num1, den1);
Rational rational2 = createRational(num2, den2);

printf("Rational Number 1: ");
printRational(rational1);

printf("Rational Number 2: ");
printRational(rational2);

printf("Addition: ");
printRational(add(rational1, rational2));

printf("Subtraction: ");
printRational(subtract(rational1, rational2));

printf("Multiplication: ");
printRational(multiply(rational1, rational2));
```

```
    printf("Division: ");
    printRational(divide(rational1, rational2));

    return 0;
}
```

Output-

```
Enter the numerator and denominator of Rational Number 1 24
24
97
Enter the numerator and denominator of Rational Number 2 35
35
84
Rational Number 1: 24 / 97
Rational Number 2: 35 / 84
Addition: 5411 / 8148
Subtraction: -1379 / 8148
Multiplication: 840 / 8148
Division: 2016 / 3395
```

Conclusion:- I understood abstract data types from this experinment..