**Batch: B4 Roll No.: 16010122221**

**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:**

**https://www.geeksforgeeks.org/abstract-data-types/**

**Abstract**:-

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

**Abstract Data type (ADT)** is a type (or class) for objects whose behaviour is defined by a set of values and a set of operations. The definition of ADT only mentions what operations are to be performed but not how these operations will be implemented. It does not specify how data will be organized in memory and what algorithms will be used for implementing the operations. It is called “**abstract**” because it gives an implementation- independent view. The process of providing only the essentials and hiding the details is known as **abstraction**.

They are important in data structures because of the following reasons:

● Representation Independence: Most of the program becomes independent of the abstract data type's representation, so that representation can be improved without breaking the entire program.

● Modularity: With representation independence, the different parts of a program become less dependent on other parts and on how those other parts are implemented.

● Interchangeability of Parts: Different implementations of an abstract data type may have different performance characteristics. With abstract data types, it becomes easier for each part of a program to use an implementation of its data types that will be more efficient for that particular part of the program.

**Abstract Data Type for Rational Number.**

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

#include <stdio.h>

typedef struct {

int numerator;

int denominator;

} RationalNumber;

int gcd(int a, int b) {

return b == 0 ? a : gcd(b, a % b);

}

RationalNumber simplify(RationalNumber number) {

int divisor = gcd(number.numerator, number.denominator);

number.numerator /= divisor;

number.denominator /= divisor;

return number;

}

RationalNumber performOperation(RationalNumber num1, RationalNumber num2, char operator) {

RationalNumber result;

switch (operator) {

case '+':

result.numerator = num1.numerator \* num2.denominator + num2.numerator \* num1.denominator;

result.denominator = num1.denominator \* num2.denominator;

break;

case '-':

result.numerator = num1.numerator \* num2.denominator - num2.numerator \* num1.denominator;

result.denominator = num1.denominator \* num2.denominator;

break;

case '\*':

result.numerator = num1.numerator \* num2.numerator;

result.denominator = num1.denominator \* num2.denominator;

break;

}

return simplify(result);

}

int isEqual(RationalNumber num1, RationalNumber num2) {

return (num1.numerator \* num2.denominator == num2.numerator \* num1.denominator);

}

void printRational(RationalNumber rational) {

printf("%d/%d", rational.numerator, rational.denominator);

}

int main() {

RationalNumber num1 = {3, 5};

RationalNumber num2 = {2, 5};

char operator = '+';

RationalNumber result = performOperation(num1, num2, operator);

printf("Result: ");

printRational(result);

printf("\n");

if (isEqual(num1, num2))

printf("The numbers are equal.\n");

else

printf("The numbers are not equal.\n");

return 0;

}

**Implementation Details:**

1. **Enlist all the Steps followed and various options explored**

Learnt the use of structure implementation , Operator Overloading, ,Object concepts, etc. Most importantly, leant Abstract Data Types aka ADT with the comparer function the assigned data type.

1. **Explain your program logic and methods used.**

The Rational ADT takes in two values and performs basic mathematical operations on it using the following methods:

* **bool Set(int N, int D):** Sets the given values for numerator and denominator
* **rational Add(rational OtherOne) :** Adds two rational numbers
* **rational Subtract(rational OtherOne) :** Subtracts two rational numbers
* **rational Multiply(rational OtherOne) :** Multiplies two rational numbers
* **rational Divide(rational OtherOne)** : Divides two rational numbers
* **rational Simplify() :** Simplifies the given rational number
* **bool Read(void):** Reads the input values from the user
* **void Write(void):** Displays/Prints the value stored in the object

The String(Pstring) ADT takes in one value i.e character array and performs basic string functions on it using the following methods:

* **int length(void):** Returns the length of the string object.
* **void concat(pstring s1, pstring s2):** Concatenates two given strings and stores it inside the object of the third string.
* **void copy(pstring s):** Copies given string and stores it inside the object of the respective string.
* **bool compare(pstring s):** Returns true or false based on if the given string is equal to the string object.
* **void display(void):** Displays the string object
* **void Read(void):** Reads the String from the user
* **void Set(char \*s, int len):** Sets the string to given character array and also stores it’s length.

The Complex Number ADT takes in two values i.e Real numbers and Imaginary Number and performs basic mathematical operations on it using the following methods:

* **bool Set(double r, double i):** Sets the given values for real and imaginary
* **complex Add(complex OtherOne) :** Adds two complex numbers
* **complex Subtract(complex OtherOne) :** Subtracts two complex numbers
* **complex Multiply(complex OtherOne) :** Multiplies two complex numbers
* **complex Divide(complex OtherOne)** : Divides two complex numbers
* **bool Read(void):** Reads the input values from the user
* **void Write(void):** Displays/Prints the value stored in the object

**3. Explain the Importance of the approach followed by you**

The approach of creating a rational number is important because we learnt using ADT to implement different types of data structures by defining a set of functions or rules operating on them. We can use this concept to create more alike data types. The same goes for the Pstring (String) ADT which was created.

**Program code and Output screenshots:**

#include <stdio.h>

typedef struct {

int numerator;

int denominator;

} RationalNumber;

int abs(int n) {

return n >= 0 ? n : -n;

}

int gcd(int a, int b) {

a = abs(a);

b = abs(b);

while (b != 0) {

int temp = b;

b = a % b;

a = temp;

}

return a;

}

void simplify(RationalNumber \*number) {

int divisor = gcd(number->numerator, number->denominator);

number->numerator /= divisor;

number->denominator /= divisor;

}

RationalNumber add(RationalNumber num1, RationalNumber num2) {

RationalNumber result;

result.numerator = num1.numerator \* num2.denominator + num2.numerator \* num1.denominator;

result.denominator = num1.denominator \* num2.denominator;

simplify(&result);

return result;

}

RationalNumber subtract(RationalNumber num1, RationalNumber num2) {

RationalNumber result;

result.numerator = num1.numerator \* num2.denominator - num2.numerator \* num1.denominator;

result.denominator = num1.denominator \* num2.denominator;

simplify(&result);

return result;

}

RationalNumber multiply(RationalNumber num1, RationalNumber num2) {

RationalNumber result;

result.numerator = num1.numerator \* num2.numerator;

result.denominator = num1.denominator \* num2.denominator;

simplify(&result);

return result;

}

int isEqual(RationalNumber num1, RationalNumber num2) {

return (num1.numerator \* num2.denominator == num2.numerator \* num1.denominator);

}

int main() {

RationalNumber num1 = {3, 5};

RationalNumber num2 = {2, 5};

RationalNumber sum = add(num1, num2);

printf("Sum: %d/%d\n", sum.numerator, sum.denominator);

RationalNumber diff = subtract(num1, num2);

printf("Difference: %d/%d\n", diff.numerator, diff.denominator);

RationalNumber product = multiply(num1, num2);

printf("Product: %d/%d\n", product.numerator, product.denominator);

if (isEqual(num1, num2))

printf("The numbers are equal.\n");

else

printf("The numbers are not equal.\n");

return 0;

}

A screenshot of a computer

Description automatically generated

**Conclusion:-**

Hence Abstract Data Type is implemented and learnt using Rational Numbers, String functions and Complex Numbers ADT.