Batch:___B2_______ Roll No.:16010121110

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/c-plus-plus/?ref=shm

Abstract:-

Abstract Data type (ADT) is a type (or class) for objects whose behavior is defined by a set of values and a set of operations. In order to simplify the process of solving problems, we can create data structures along with their operations, and such data structures that are not in-built are known as Abstract Data Type (ADT). *(Abstract data types.*

GeeksforGeeks. (2022, July 18). Retrieved September 7, 2022, from https://www.geeksforgeeks.org/abstract-data-types/

)

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

ADT's are used to explain custom data structures to the user. They are a way to represent the data structures in plain english. This can be implemented in any language. ADTs are language independent. This is why they are important.

Abstract Data Type for Rational Numbers

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

ADT- rational number

value defination

integer numerator

integer denominator

preconditions-

denominator must not be 0

Operator defination.

abstract rationaltype add(rationaltype a, rationaltype b){

preconditions none

preconditions -

function returs a new rationaltype.

new numerator value is (a.numerator)*(b.denominator)+(b.numerator)*(a.denominator)

new denominator (a.denominator)*(b.denominator)

}

abstract rationaltype sub(rationaltype a, rationaltype b){

preconditions none

preconditions -

function returs a new rationaltype.

new numerator value is (a.numerator)*(b.denominator)-(b.numerator)*(a.denominator)

new denominator (a.denominator)*(b.denominator)

}

abstract rationaltype multiply(rationaltype a, rationaltype b){

preconditions none

preconditions -

function returs a new rationaltype.

new numerator value is (a.numerator)*(b.numerator)

new denominator (a.denominator)*(b.denominator)

}

abstract rationaltype divide(rationaltype a, rationaltype b){

preconditions none

preconditions -

function returs a new rationaltype.

new numerator value is (a.numerator)*(b.denominator)

new denominator (a.denominator)*(b.numerator)

}

abstract boolean equate(rationaltype a, rationaltype b){

preconditions none

preconditions -

returns true iff a.numerator==b.numerator AND a.denominator==b.denominator else false

Implementation Details:

1. Enlist all the Steps followed and various options explored .
Made class rational then made variables numerator and denominator made static functions.
2. Explain your program logic and methods used.
OOP logic used. Methods according to ATD
3. Explain the Importance of the approach followed by you.
Represents OOP principles.

Program code and Output screenshots:

/***

ADT- rational number

value defination

integer numerator

integer denominator

preconditions-

denominator must not be 0

Operator defination.

abstract rationaltype add(rationaltype a, rationaltype b){

preconditions none

preconditions -

function returs a new rationaltype.

new numerator value is (a.numerator)*(b.denominator)+(b.numerator)*(a.denominator)

new denominator (a.denominator)*(b.denominator)

}

abstract rationaltype sub(rationaltype a, rationaltype b){

preconditions none

preconditions -

function returs a new rationaltype.

new numerator value is (a.numerator)*(b.denominator)-(b.numerator)*(a.denominator)

new denominator (a.denominator)*(b.denominator)

}

abstract rationaltype multiply(rationaltype a, rationaltype b){

preconditions none

preconditions -

function returs a new rationaltype.

new numerator value is (a.numerator)*(b.numerator)

new denominator (a.denominator)*(b.denominator)

}

abstract rationaltype divide(rationaltype a, rationaltype b){

preconditions none

preconditions -

function returs a new rationaltype.

new numerator value is (a.numerator)*(b.denominator)

new denominator (a.denominator)*(b.numerator)

}

abstract boolean equate(rationaltype a, rationaltype b){

preconditions none

preconditions -

returns true iff a.numerator==b.numerator AND a.denominator==b.denominator

else false

}

```
**************************************************************************/

#include <iostream>

using namespace std;

class rational{
  public:
 int numerator;
 int denominator;
 public: rational(int p, int q){
  if (q==0){
     throw std::invalid_argument( "received infinity value" );
  }
    this->numerator=p;
   this->denominator=q;
```

```
    }
    static rational* add(rational a, rational b){
        return new rational((a.numerator)*(b.denominator)+(b.numerator)*(a.denominator),(a.denominator)*(b.denominator));
    }
    static rational* sub(rational a, rational b){
        return new rational((a.numerator)*(b.denominator)-(b.numerator)*(a.denominator),(a.denominator)*(b.denominator));
    }
    static rational* multiply(rational a, rational b){
        return new rational((a.numerator)*(b.numerator),(a.denominator)*(b.denominator));
    }

    static rational* divide(rational a, rational b){
        return new rational((a.numerator)*(b.denominator),(a.denominator)*(b.numerator));
    }
    static bool equate(rational a, rational b){
        if(a.numerator==b.numerator&a.denominator==b.denominator){
            return true;
        }
```

```
        else{
            return false;
        }
    }
};
int main()
{
    cout<<"Hello World\n";
    rational *a = new rational(2,3);
    rational *b = new rational(3,4);
    rational *c = rational::divide(*a,*b);
    cout<<c->numerator<<"\n";
    cout<<rational::equate(*a,*b);
    return 0;
}
/*
a/b

preconditions - b must not be zero

*/
```

Hello World

8

0

Conclusion:-

Thus we have implemented ADTs in C++ without standard libraries. We used ADT to make code for rational numbers. ADTs are very useful for representing data structures.