Somaiya Vidyavihar University
K. J. Somaiya College of Engineering

Batch: A3 **Roll No.: 16010121045**

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 Numbers

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

rationalclass.h

```
#ifndef RATIONALCLASS_H
#define RATIONALCLASS_H
#include <stdbool.h>

class rational
{

public:
   // Constructors
   rational(void);

   rational(int N, int D);

   // Set the value. Fails if D is not natural
   bool Set(int N, int D);

   // Functions
   rational Add(rational OtherOne);
   rational Subtract(rational OtherOne);
   rational Multiply(rational OtherOne);
   rational Divide(rational OtherOne);
   rational Simplify();
```

```
    // Read. Fails if D is not natural
    bool Read(void);

    // Write
    void Write(void);

private:
    int Numerator,
        Denominator;
};
#endif
```

rationalclass.cpp

```
#include <bits/stdc++.h>
#include "rationalclass.h"
using namespace std;
// Constructors
rational::rational(void)
{

    Set(0, 1);
}

rational::rational(int N, int D)
{

    Set(N, D);
}
// Set the value. Fails if D is not natural
```

```
bool rational::Set(int N, int D)
{

   if (D >= 1)
   {
      Numerator = N;
      Denominator = D;
      return (true);
   }
   else
      return (false);
}

// Add
rational rational::Add(rational OtherOne)
{

   rational Answer;

   Answer.Set(Numerator * OtherOne.Denominator + Denominator *
OtherOne.Numerator,
            Denominator * OtherOne.Denominator);

   return (Answer);
}
// Substraction
rational rational::Subtract(rational OtherOne)
{
```

```
    rational Answer;

    Answer.Set(Numerator * OtherOne.Denominator - Denominator *
OtherOne.Numerator,
            Denominator * OtherOne.Denominator);

    return (Answer);
}
// Multiplication
rational rational::Multiply(rational OtherOne)
{

    rational Answer;
    Answer.Set(Numerator * OtherOne.Numerator,
            Denominator * OtherOne.Denominator);
    return (Answer);
}
// Division
rational rational::Divide(rational OtherOne)
{

    rational Answer;
    Answer.Set(Numerator * OtherOne.Denominator,
            Denominator * OtherOne.Numerator);
    return (Answer);
}
// Read. Fails if D is not natural
bool rational::Read(void)
{
```

```
    int NewNumerator,
        NewDenominator;

    cin >> NewNumerator >> NewDenominator;

    return (Set(NewNumerator, NewDenominator));
}
// Write
void rational::Write(void)
{
    int gcd = __gcd(Numerator, Denominator);
    cout << Numerator / gcd << "/" << Denominator / gcd;
}
```

useRational.cpp

```
#include <bits/stdc++.h>
#include "rationalclass.h"
#include "rationalclass.cpp"
using namespace std;
int main(void)
{

    rational R1,
        R2,
        Sum,
        Multi,
        Sub,
        Divide;
```

```
cout << "Enter N and D for new R1 :";
if (!R1.Read())
    cout << "Doh, must have a positive denominator" << endl;

cout << "Enter N and D for new R2 :";
if (!R2.Read())
    cout << "Doh, must have a positive denominator" << endl;

cout << "R1 is ";
R1.Write();
cout << " and R2 is ";
R2.Write();
cout << endl;

Sum = R1.Add(R2);
cout << "Sum is ";
Sum.Write();
cout << endl;

Sub = R1.Subtract(R2);
cout << "Subtraction is ";
Sub.Write();
cout << endl;

Multi = R1.Multiply(R2);
cout << "Multiplication is ";
Multi.Write();
cout << endl;
```

```
    Divide = R1.Divide(R2);
    cout << "Division is ";
    Divide.Write();
    cout << endl;
    return (0);
}
```

Abstract Data Type for String and String Functions

pstring.h

```
#ifndef RATIONALCLASS_H
#define RATIONALCLASS_H
#include <stdbool.h>
class pstring
{
public:
    // Constructors
    pstring(void);
    pstring(char *s);

    // String Functions
    int length(void);
    void concat(pstring s1, pstring s2);
    void copy(pstring s);
    bool compare(pstring s);

    // display
    void display(void);
    // Read
```

```
    void Read(void);
    // Setter
    void Set(char *s, int len);

private:
    char str[1000];
    int len;
};
#endif
```

pstring.cpp

```
#include <bits/stdc++.h>
#include "pstring.h"
using namespace std;
// Constructors
pstring::pstring(void){
    char temp[1000];
    Set(temp,0);
}

pstring::pstring(char* s){
    int c=0;
    while(str[c]!='\0')
        c++;
    Set(s,c);
}

// Setter
void pstring::Set(char* s,int l){
    for(int i=0;i<l;i++)
```

```
    str[i]=s[i];
    len=l;
}

// length
int pstring::length(){
    return len;
}

// Concatenate
void pstring::concat(pstring s1,pstring s2){
    char temp[s1.len+s2.len];
    for(int i=0;i<s1.len;i++)
        temp[i]=s1.str[i];
    for(int i=s1.len;i<s1.len+s2.len;i++)
        temp[i]=s2.str[i-s1.len];
    Set(temp,s1.len+s2.len);
}

// Copy
void pstring::copy(pstring s1){
    Set(s1.str,s1.len);
}

bool pstring::compare(pstring s1){
    if(len==s1.len)
    {
        for(int i=0;i<len;i++)
            if(s1.str[i]!=str[i])
```

```
            return false;
        return true;

    }
    return false;
}

// Display
void pstring::display(void){
    for(int i=0;i<len;i++)
        cout<<str[i];
}

// Read
void pstring::Read(void){
    char temp[1000];
    cin>>temp;
    int c=0;
    while(temp[c]!='\0')
        c++;
    Set(temp,c);
}
```

Useptring.cpp

```
#include <bits/stdc++.h>
#include "pstring.h"
#include "pstring.cpp"
using namespace std;
```

```
int main(void) {


pstring str1,str2,str3,str4;


cout << "Enter String1: ";
str1.Read();
cout << "Enter String2: ";
str2.Read();


str1.display();
cout<<" length is "<<str1.length()<<endl;
str2.display();
cout<<" length is "<<str2.length()<<endl;


str3.concat(str1,str2);
cout<<"The concatenation is: ";
str3.display();
cout<<endl;


str4.copy(str1);
cout<<"Copied String1 To String4: ";
str4.display();
cout<<endl;


cout<<"Comparing String1 and String4"<<endl;
if(str1.compare(str4))
   cout<<"The are equal"<<endl;
else
   cout<<"They are not equal"<<endl;
```

```
cout<<"Comparing String2 and String4"<<endl;
if(str2.compare(str4))
    cout<<"The are equal"<<endl;
else
    cout<<"They are not equal"<<endl;
return(0);
}
```

Abstract Data Type for Complex Numbers

compnumclass.h

```
#ifndef COMPNUMCLASS_H
#define COMPNUMCLASS_H
#include <stdbool.h>

class compnum
{

public:
    // Constructors
    compnum(void);
    compnum(double r, double i);

    // Set the value
    bool Set(double r, double i);
```

```
    // Functions
    compnum Add(compnum OtherOne);
    compnum Subtract(compnum OtherOne);
    compnum Multiply(compnum OtherOne);
    compnum Divide(compnum OtherOne);

    // Read
    bool Read(void);

    // Write
    void Write(void);

private:
    double real, img;
};
#endif
```

compnumclass.cpp

```
#include <bits/stdc++.h>
#include "compnumclass.h"
using namespace std;
// Constructors
compnum::compnum(void)
{
    Set(0, 0);
}
```

```
compnum::compnum(double r, double i)
{
    Set(r, i);
}
// Set the value. Fails if D is not natural
bool compnum::Set(double r, double i)
{
    real = r;
    img = i;
    return true;
}

// Add
compnum compnum::Add(compnum OtherOne)
{
    compnum Answer;
    Answer.Set(real + OtherOne.real, img + OtherOne.img);
    return (Answer);
}
// Substraction
compnum compnum::Subtract(compnum OtherOne)
{
    compnum Answer;
    Answer.Set(real - OtherOne.real, img - OtherOne.img);
    return (Answer);
}
// Multiplication
compnum compnum::Multiply(compnum OtherOne)
```

```
{
    compnum Answer;
    double tempr = (real * OtherOne.real) - (img * OtherOne.img);
    double tempi = (real * OtherOne.img) - (img * OtherOne.real);
    Answer.Set(tempr, tempi);
    return (Answer);
}
// Division
compnum compnum::Divide(compnum OtherOne)
{
    compnum Answer;
    double den = (OtherOne.real * OtherOne.real) + (OtherOne.img *
OtherOne.img);
    double tempr = (real * OtherOne.real) + (img * OtherOne.img);
    double tempi = (img * OtherOne.real) - (real * OtherOne.img);
    Answer.Set(tempr / den, tempi / den);
    return (Answer);
}
// Read. Fails if D is not natural
bool compnum::Read(void)
{
    double tempr, tempi;
    cin >> tempr >> tempi;
    return (Set(tempr, tempi));
}
// Write
void compnum::Write(void)
{
    cout << endl
```

```
        << "Real Part is: " << real << " Imaginary Part is: " << img << "i" << endl;
}
```

usecompnum.cpp

```
#include <bits/stdc++.h>
#include "compnumclass.h"
#include "compnumclass.cpp"
using namespace std;
int main(void)
{
    compnum N1, N2, Sum, Multi, Sub, Divide;

    cout << "Enter Real and Imaginary for new N1 : ";
    if (!N1.Read())
        cout << "Something went wrong" << endl;
    cout << "Enter Real and Imaginary for new N2 : ";
    if (!N2.Read())
        cout << "Something went wrong" << endl;

    cout << "N1 is ";
    N1.Write();
    cout << "N2 is ";
    N2.Write();
    cout << endl;

    Sum = N1.Add(N2);
    cout << "Sum is ";
    Sum.Write();
    cout << endl;
```

```
    Sub = N1.Subtract(N2);
    cout << "Subtraction is ";
    Sub.Write();
    cout << endl;

    Multi = N1.Multiply(N2);
    cout << "Multiplication is ";
    Multi.Write();
    cout << endl;

    Divide = N1.Divide(N2);
    cout << "Division is ";
    Divide.Write();
    cout << endl;
    return (0);
}
```

Implementation Details:

1. Enlist all the Steps followed and various options explored

Ans: 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.

2. Explain your program logic and methods used.

Ans: 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.

Ans: 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:
Rational Number

```
cd "/Users/pargat/Documents/COLLEGE/DS/Ex
pargat@Pargats-MacBook-Air DS % cd "/User
s.cpp -o useRationals && "/Users/pargat/D
Enter N and D for new R1 :5 3
Enter N and D for new R2 :10 3
R1 is 5/3 and R2 is 10/3
Sum is 5/1
Subtraction is -5/3
Multiplication is 50/9
Division is 1/2
pargat@Pargats-MacBook-Air Rational %
```

String ADT

```
pargat@Pargats-MacBook-Air Rational % c
ing.cpp -o usepstring && "/Users/pargat
Enter String1: Hello
Enter String2: World
Hello length is 5
World length is 5
The concatenation is: HelloWorld
Copied String1 To String4: Hello
Comparing String1 and String4
The are equal
Comparing String2 and String4
They are not equal
pargat@Pargats-MacBook-Air String %
```

Complex Numbers

```
pargat@Pargats-MacBook-Air Complex % cd "/Users/pargat/
COLLEGE/DS/Code/Exp1/Complex/"useComplex
Enter Real and Imaginary for new N1 : 10 5
Enter Real and Imaginary for new N2 : 5 2
N1 is
Real Part is: 10 Imaginary Part is: 5i
N2 is
Real Part is: 5 Imaginary Part is: 2i

Sum is
Real Part is: 15 Imaginary Part is: 7i

Subtraction is
Real Part is: 5 Imaginary Part is: 3i

Multiplication is
Real Part is: 40 Imaginary Part is: -5i

Division is
Real Part is: 2.06897 Imaginary Part is: 0.172414i
```

Conclusion:-

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