|  |  |  |  |
| --- | --- | --- | --- |
| **Course Name:** | **Data Structures Laboratory** | **Semester:** | **III** |
| **Date of Performance:** | **26 / 09 / 2023** | **Batch No:** | **A - 3** |
| **Faculty Name:** | **Prof. Om Goswami** | **Roll No:** | **16014022050** |
| **Faculty Sign & Date:** |  | **Grade/Marks:** | **\_\_\_ / 25** |

**Experiment No: 1**

**Title: Abstract Data Type**

|  |
| --- |
| **Aim and Objective of the Experiment:** |
| **Implement complex numbers/ rational numbers using abstract data type (ADT) and define various mathematical functions for:**   1. **Addition of Complex/rational numbers** 2. **Subtraction of Complex/rational numbers** 3. **Multiplication of Complex/rational numbers** 4. **Equality check of Complex numbers/rational numbers**   **Display appropriate results of all the functionalities using a menu driven approach.** |

|  |
| --- |
| **COs to be achieved:** |
| **CO1:** Understand and implement the different data structures used in problem solving  **CO2:** Apply linear and non-linear data structure in application development |

|  |
| --- |
| **Books/Journals/Websites referred:** |
| 1. [Abstract Data Types - GeeksforGeeks](https://www.geeksforgeeks.org/abstract-data-types/) 2. Data Structures by Reema Thareja |

|  |
| --- |
| **Tools required:** |
| DEV C/C++ compiler/ Code blocks C compiler |

|  |
| --- |
| **Theory:** |
| **Abstract Data Type:**  An abstract data type (ADT) is the way we look at a data structure, focusing on what it does and ignoring how it does its job. An abstract data type can be a structure considered without regard to its implementation. It can be thought of as a ‘description’ of the data in the structure with a list of operations that can be performed on the data within that structure. |

|  |
| --- |
| **Implementation details:** |
| 1. **Enlist all the Steps followed and various options explored.**   **Step 1:** Define the abstract data types (ADTs) for complex numbers and rational numbers using structs.  **Step 2:** Implement mathematical functions for addition, subtraction, multiplication, and equality checks for both complex and rational numbers.  **Step 3:** Create a menu-driven approach using a loop where the user can select different operations.  **Step 4:** Implement each operation inside a switch case based on user input.  **Step 5:** Take user input for complex and rational numbers where necessary.  **Step 6:** Display the results of the operations.  **Step 7:** Allow the user to exit the program or continue with other operations.  **Step 8:** Repeat steps 4-7 until the user chooses to exit.   1. **Explain your program logic and methods used.**  * The program starts by defining two ADTs, one for complex numbers (Complex) and another for rational numbers (Rational), both using structs to represent their components. * It then implements mathematical functions for each operation: * For complex numbers: addition, subtraction, multiplication, and equality checks. * For rational numbers: addition, subtraction, multiplication, and equality checks. * The program uses a menu-driven approach where the user can select which operation to perform. * Inside the menu loop, the program takes user input for the required numbers based on the operation chosen. * After performing the operation, it displays the result. * The program continues to execute the user's chosen operations until the user decides to exit.  1. **Explain the Importance of the approach followed by you.**  * **Modularity:** This approach is modular and separates the concerns of defining ADTs, implementing operations, and handling user interactions. This makes the code easier to understand, maintain, and extend. * **Reusability:** By defining ADTs and implementing operations as functions, you can reuse these components in other parts of your code or in other projects that require similar functionality. * **User-Friendly:** The menu-driven approach provides a user-friendly interface, allowing users to interact with the program easily without needing to understand the code's internal details. * **Clarity:** The program's structure and use of functions make it clear and organized. Each function has a well-defined purpose, which improves code readability. * **Error Handling:** While not explicitly shown in the provided code, you can easily incorporate error handling to handle cases like division by zero or invalid inputs, enhancing the program's robustness. * **Scalability:** This approach is scalable, making it straightforward to add more operations or expand the program's capabilities in the future. |

|  |
| --- |
| **C/C++ Code implemented:** |
| #include <stdio.h>  #include <stdlib.h>  // Define the complex number ADT  typedef struct {      double real;      double imaginary;  } Complex;  // Define the rational number ADT  typedef struct {      int numerator;      int denominator;  } Rational;  // Function to add two complex numbers  Complex addComplex(Complex c1, Complex c2) {      Complex result;      result.real = c1.real + c2.real;      result.imaginary = c1.imaginary + c2.imaginary;      return result;  }  // Function to subtract two complex numbers  Complex subtractComplex(Complex c1, Complex c2) {      Complex result;      result.real = c1.real - c2.real;      result.imaginary = c1.imaginary - c2.imaginary;      return result;  }  // Function to multiply two complex numbers  Complex multiplyComplex(Complex c1, Complex c2) {      Complex result;      result.real = c1.real \* c2.real - c1.imaginary \* c2.imaginary;      result.imaginary = c1.real \* c2.imaginary + c1.imaginary \* c2.real;      return result;  }  // Function to check equality of two complex numbers  int isEqualComplex(Complex c1, Complex c2) {      return c1.real == c2.real && c1.imaginary == c2.imaginary;  }  // Function to add two rational numbers  Rational addRational(Rational r1, Rational r2) {      Rational result;      result.numerator = r1.numerator \* r2.denominator + r2.numerator \* r1.denominator;      result.denominator = r1.denominator \* r2.denominator;      return result;  }  // Function to subtract two rational numbers  Rational subtractRational(Rational r1, Rational r2) {      Rational result;      result.numerator = r1.numerator \* r2.denominator - r2.numerator \* r1.denominator;      result.denominator = r1.denominator \* r2.denominator;      return result;  }  // Function to multiply two rational numbers  Rational multiplyRational(Rational r1, Rational r2) {      Rational result;      result.numerator = r1.numerator \* r2.numerator;      result.denominator = r1.denominator \* r2.denominator;      return result;  }  // Function to check equality of two rational numbers  int isEqualRational(Rational r1, Rational r2) {      return (r1.numerator \* r2.denominator) == (r2.numerator \* r1.denominator);  }  int main() {      printf("ketaki mahajan / A-3 / 16014022050");      int choice;        do {          printf("\n\nMenu:\n");          printf("1. Add Complex Numbers\n");          printf("2. Subtract Complex Numbers\n");          printf("3. Multiply Complex Numbers\n");          printf("4. Check Equality of Complex Numbers\n");          printf("5. Add Rational Numbers\n");          printf("6. Subtract Rational Numbers\n");          printf("7. Multiply Rational Numbers\n");          printf("8. Check Equality of Rational Numbers\n");          printf("0. Exit\n");          printf("Enter your choice: ");          scanf("%d", &choice);            switch (choice) {              case 1: // Add Complex Numbers              {                  Complex c1, c2, result;                  printf("Enter the real and imaginary parts of the first complex number: ");                  scanf("%lf %lf", &c1.real, &c1.imaginary);                  printf("Enter the real and imaginary parts of the second complex number: ");                  scanf("%lf %lf", &c2.real, &c2.imaginary);                  result = addComplex(c1, c2);                  printf("Result: %lf + %lfi\n", result.real, result.imaginary);                  break;              }              case 2: // Subtract Complex Numbers              {                  Complex c1, c2, result;                  printf("Enter the real and imaginary parts of the first complex number: ");                  scanf("%lf %lf", &c1.real, &c1.imaginary);                  printf("Enter the real and imaginary parts of the second complex number: ");                  scanf("%lf %lf", &c2.real, &c2.imaginary);                  result = subtractComplex(c1, c2);                  printf("Result: %lf + %lfi\n", result.real, result.imaginary);                  break;              }              case 3: // Multiply Complex Numbers              {                  Complex c1, c2, result;                  printf("Enter the real and imaginary parts of the first complex number: ");                  scanf("%lf %lf", &c1.real, &c1.imaginary);                  printf("Enter the real and imaginary parts of the second complex number: ");                  scanf("%lf %lf", &c2.real, &c2.imaginary);                  result = multiplyComplex(c1, c2);                  printf("Result: %lf + %lfi\n", result.real, result.imaginary);                  break;              }              case 4: // Check Equality of Complex Numbers              {                  Complex c1, c2;                  printf("Enter the real and imaginary parts of the first complex number: ");                  scanf("%lf %lf", &c1.real, &c1.imaginary);                  printf("Enter the real and imaginary parts of the second complex number: ");                  scanf("%lf %lf", &c2.real, &c2.imaginary);                  if (isEqualComplex(c1, c2)) {                      printf("Complex numbers are equal.\n");                  } else {                      printf("Complex numbers are not equal.\n");                  }                  break;              }              case 5: // Add Rational Numbers              {                  Rational r1, r2, result;                  printf("Enter the numerator and denominator of the first rational number: ");                  scanf("%d %d", &r1.numerator, &r1.denominator);                  printf("Enter the numerator and denominator of the second rational number: ");                  scanf("%d %d", &r2.numerator, &r2.denominator);                  result = addRational(r1, r2);                  printf("Result: %d/%d\n", result.numerator, result.denominator);                  break;              }              case 6: // Subtract Rational Numbers              {                  Rational r1, r2, result;                  printf("Enter the numerator and denominator of the first rational number: ");                  scanf("%d %d", &r1.numerator, &r1.denominator);                  printf("Enter the numerator and denominator of the second rational number: ");                  scanf("%d %d", &r2.numerator, &r2.denominator);                  result = subtractRational(r1, r2);                  printf("Result: %d/%d\n", result.numerator, result.denominator);                  break;              }              case 7: // Multiply Rational Numbers              {                  Rational r1, r2, result;                  printf("Enter the numerator and denominator of the first rational number: ");                  scanf("%d %d", &r1.numerator, &r1.denominator);                  printf("Enter the numerator and denominator of the second rational number: ");                  scanf("%d %d", &r2.numerator, &r2.denominator);                  result = multiplyRational(r1, r2);                  printf("Result: %d/%d\n", result.numerator, result.denominator);                  break;              }              case 8: // Check Equality of Rational Numbers              {                  Rational r1, r2;                  printf("Enter the numerator and denominator of the first rational number: ");                  scanf("%d %d", &r1.numerator, &r1.denominator);                  printf("Enter the numerator and denominator of the second rational number: ");                  scanf("%d %d", &r2.numerator, &r2.denominator);                  if (isEqualRational(r1, r2)) {                      printf("Rational numbers are equal.\n");                  } else {                      printf("Rational numbers are not equal.\n");                  }                  break;              }              case 0:                  printf("Exiting program.\n");                  break;              default:                  printf("Invalid choice. Please enter a valid option.\n");                  break;          }      } while (choice != 0);      return 0;  } |

|  |
| --- |
| **Output/ program results after execution:** |
|  |

|  |
| --- |
| **Post Lab Subjective/Objective type Questions:** |
| 1. **Write a program to insert and delete a number from a given location in an array.**   #include <stdio.h>  // Function to display the elements of an array  void displayArray(int arr[], int size) {      printf("Array: ");      for (int i = 0; i < size; i++) {          printf("%d ", arr[i]);      }      printf("\n");  }  // Function to insert a number at a specified location in an array  int insertElement(int arr[], int size, int location, int value) {      if (location < 0 || location > size) {          printf("Invalid location for insertion.\n");          return size;      }      // Shift elements to the right to make space for the new value      for (int i = size; i > location; i--) {          arr[i] = arr[i - 1];      }      // Insert the new value at the specified location      arr[location] = value;      return size + 1; // Increase the size of the array  }  // Function to delete an element from a specified location in an array  int deleteElement(int arr[], int size, int location) {      if (location < 0 || location >= size) {          printf("Invalid location for deletion.\n");          return size;      }      // Shift elements to the left to overwrite the deleted value      for (int i = location; i < size - 1; i++) {          arr[i] = arr[i + 1];      }      return size - 1; // Decrease the size of the array  }  int main() {      printf("ketaki mahajan / A-3 / 16014022050");      int arr[100]; // Assuming a maximum array size of 100      int size, choice, location, value;      printf("\nEnter the size of the array: ");      scanf("%d", &size);      printf("Enter %d elements: ", size);      for (int i = 0; i < size; i++) {          scanf("%d", &arr[i]);      }      displayArray(arr, size);      do {          printf("\nMenu:\n");          printf("1. Insert a number\n");          printf("2. Delete a number\n");          printf("0. Exit\n");          printf("Enter your choice: ");          scanf("%d", &choice);          switch (choice) {              case 1:                  printf("Enter the location where you want to insert: ");                  scanf("%d", &location);                  printf("Enter the value to insert: ");                  scanf("%d", &value);                  size = insertElement(arr, size, location, value);                  displayArray(arr, size);                  break;              case 2:                  printf("Enter the location of the element you want to delete: ");                  scanf("%d", &location);                  size = deleteElement(arr, size, location);                  displayArray(arr, size);                  break;              case 0:                  printf("Exiting program.\n");                  break;              default:                  printf("Invalid choice. Please enter a valid option.\n");                  break;          }      } while (choice != 0);      return 0;  } |

|  |
| --- |
| **Conclusion:** |
| In conclusion, we learned about the importance of abstract data types (ADTs) in structuring complex and rational numbers within programs. These examples underscored the significance of modularity through well-encapsulated functions and the benefits of a menu-driven user interface for user-friendly interactions. |

|  |
| --- |
| **Signature of faculty in-charge with Date:** |