

| **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:**

1. Aaron M. Tenenbaum, Yedidyah Langsam, Moshe J. Augenstein, “Data Structures Using C”, Pearson.
2. <https://en.wikipedia.org/wiki/Abstract_data_type>
3. <https://www.scaler.com/topics/abstract-data-type-in-data-structure/>

**Abstract**:-

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

In computer science, an Abstract Data Type (ADT) is a mathematical model for data types. It is defined by its behaviour from the point of view of the user, of the data, specifically in terms of possible values, possible operations on data of this type, and the behaviour of those operations. Formally, an ADT may be defined as a “class of objects whose logical behaviour is defined by a set of values and a set of operations”.

The ADT’s are important in data structures due to the following reasons:

1. ADT in data structure makes it very easy to use complex data structures and their complex functions. It follows an object oriented programming paradigm.
2. By using ADT’s, any data structure can be customized depending on how that data structure is planned to be used.
3. ADT in data structure follows the concept of reusability of a code. This means that a particular piece of code does not have to be repeatedly written. An ADT can be created instead and can be used by simply calling the functions present in it.

**Abstract Data Type for** Complex Number

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

*/\* value definition \*/*

abstract typedef <int, int>COMPLEX;

condition NONE;

/\* operator definition \*/

abstract add(a, b) /\* written a + b \*/

COMPLEX a, b;

postcondition add.r == a.r + b.r;

add.i == a.i + b.i;

abstract sub(a, b) /\* written a – b \*/

COMPLEX a, b;

postcondition sub.r == a.r – b.r;

sub.i == a.i – b.i;

abstract mul(a, b) /\* written a\*b \*/

COMPLEX a, b;

postcondition mul.r == a.r\*b.r – a.i\*b.i;

mul.i == a.i\*b.r + a.r\*b.i;

abstract eq(a, b)

COMPLEX a, b;

postcondition eq == ((a.r==b.r) && (a.i==b.i));

**Implementation Details:**

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

Ans. In order to create the ADT “Complex Number”, the following steps have been followed:

1. A structure called Complex was defined which has the variables r and i of integer data type.
2. Methods used are add, sub, mul and eq.
3. In the main method, the real and imaginary parts of the two complex numbers are accepted separately from the user.
4. The methods are called by passing the two complex numbers as parameters and storing the result sent by them (if any) in a variable.
5. Finally, the result is printed.

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

Ans. In this program, a structure named Complex has been defined which stores the two parts of a complex number, i.e., the real and the imaginary part. Then, a variable of type Complex (named complex) is defined. Then, the add() method accepting two complex numbers a and b and returning the sum of the two complex numbers is defined. The real part of result is equal to the sum of the real parts of a and b. Similarly, the imaginary part of result is equal to the sum of the imaginary parts of a and b. The sum is returned to the main() method, which is the calling method. Then, the sub() method accepting two complex numbers a and b and returning the difference of the two complex numbers is defined. The real part of result is equal to the difference of the real parts of a and b. Similarly, the imaginary part of result is equal to the difference of the imaginary parts of a and b. The difference is returned to the main() method, which is the calling method. Then, the mul() method accepting two complex numbers a and b and returning the product of the two complex numbers is defined. The real part of result is equal to the product of the real parts of a and b. Similarly, the imaginary part of result is equal to the product of the imaginary parts of a and b. The product is returned to the main() method, which is the calling method. Then, the eq() method accepting two complex numbers and returning no data is defined. If the real part of a and the real part of b as well as the imaginary part of a and the imaginary part of b are equal, then it prints the statement that the two complex numbers are equal, otherwise it prints that the two complex numbers are not equal. In the main method, c1, c2, sum, diff and prod variables of type complex are declared. The real and imaginary parts of c1 and c2 are separately input from the user. The add() method is called by passing c1 and c2 as arguments and storing the result sent by it in sum variable. The sub() method is called by passing c1 and c2 as arguments and storing the result sent by it in diff variable. The mul() method is called by passing c1 and c2 as arguments and storing the result sent by it in prod variable. The eq() method is called by passing c1 and c2 as arguments. Then, the sum, diff and prod variables are displayed to the user.

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

Ans. In this approach, the concept of Abstract Data Type is implemented. The user is unaware how the required action is performed and the desired result is obtained. The user only enters the real and imaginary parts of the two complex numbers separately and he gets their sum, difference, product and equality check without knowing how it is performed.

ADT’s are important for large-scale programming. They package data structures and operations on them, hiding internal details. In this case, the Complex Number ADT addition, subtraction, multiplication and equality check operations to users while keeping the underlying structure, whether an array, list, structure or binary tree, invisible.

**Program code and Output screenshots:**

Array as ADT:

#include<stdio.h>

int add(int a, int b)

{

printf("\nThe sum of %d and %d is: ", a, b);

return (a+b);

}

int sub(int a, int b)

{

printf("\nThe difference of %d and %d is: ", a, b);

return (a-b);

}

int mul(int a, int b)

{

printf("\nThe product of %d and %d is: ", a, b);

return (a\*b);

}

int d\_v(int a, int b)

{

printf("\nThe quotient of %d and %d is: ", a, b);

return ((float)(a/b));

}

int (\*fp[10])(int, int);

void main()

{

fp[1] = add;

fp[2] = sub;

fp[3] = mul;

fp[4] = d\_v;

int a, b, c;

float result;

do{

printf("\nEnter the first number: ");

scanf("%d", &a);

printf("\nEnter the second number: ");

scanf("%d", &b);

printf("\nEnter '1' to add, '2' to subtract,\n'3' to multiply, '4' to divide,\n'5' to exit. Enter your choice: ");

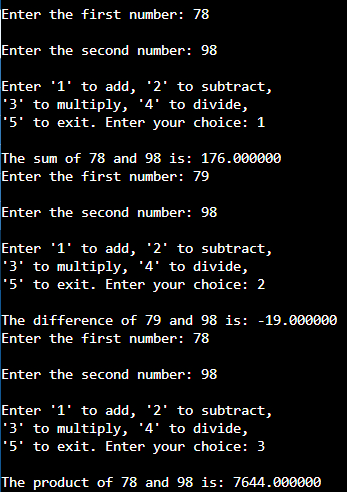
scanf("%d", &c);

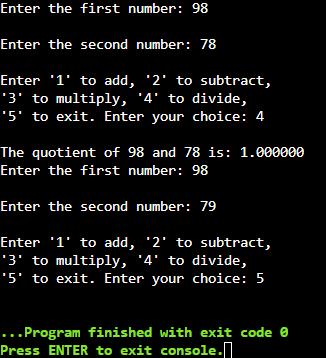
result = fp[c](a, b);

printf("%f", result);

}while(c!=5);

}





Complex Number as ADT:

#include<stdio.h>

typedef struct Complex

{

int r;

int i;

}complex;

complex add(complex a, complex b)

{

complex result;

result.r = a.r + b.r;

result.i = a.i + b.i;

return result;

}

complex sub(complex a, complex b)

{

complex result;

result.r = a.r - b.r;

result.i = a.i - b.i;

return result;

}

complex mul(complex a, complex b)

{

complex result;

result.r = a.r\*b.r - a.i\*b.i;

result.i = a.i\*b.r + a.r\*b.i;

return result;

}

void eq(complex a, complex b)

{

if((a.r==b.r)&&(a.i==b.i))

printf("\nThe complex numbers %d+i%d and %d+i%d are equal.", a.r, a.i, b.r, b.i);

else

printf("\nThe complex numbers %d+i%d and %d+i%d are not equal.", a.r, a.i, b.r, b.i);

}

void main()

{

complex c1, c2, sum, diff, prod;

printf("\nEnter the real part of the first complex number: ");

scanf("%d", &c1.r);

printf("\nEnter the imaginary part of the first complex number: ");

scanf("%d", &c1.i);

printf("\nEnter the real part of the second complex number: ");

scanf("%d", &c2.r);

printf("\nEnter the imaginary part of the second complex number: ");

scanf("%d", &c2.i);

sum = add(c1, c2);

diff = sub(c1, c2);

prod = mul(c1, c2);

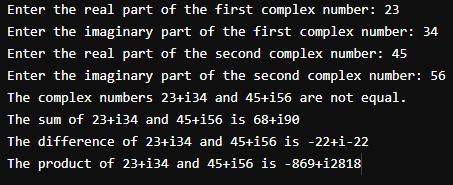
eq(c1, c2);

printf("\nThe sum of %d+i%d and %d+i%d is %d+i%d", c1.r, c1.i, c2.r, c2.i, sum.r, sum.i);

printf("\nThe difference of %d+i%d and %d+i%d is %d+i%d", c1.r, c1.i, c2.r, c2.i, diff.r, diff.i);

printf("\nThe product of %d+i%d and %d+i%d is %d+i%d", c1.r, c1.i, c2.r, c2.i, prod.r, prod.i);

}



**Conclusion:-**

Thus, in this experiment, the concept of ADT has been learnt and has been implemented in the from of Complex Numbers and Arrays.