**Input-Output Examples**

**Case 1: Inserting Data into the Hash Table (No Collision)**

**Input**:

mathematica

Copy code

Enter Number of Clients: 3

Enter Phone Numbers:

--> 12

--> 25

--> 30

Enter Hash Table Size: 5

If collision occurs, which collision resolution technique do you want to use?

1. Linear Probing

2. Double Hashing

3. Exit

Enter Your Choice: 1

**Output**:

lua

Copy code

Hash Value Key

2 ---> 12

0 ---> 25

1 ---> 30

**Case 2: Collision Resolved by Linear Probing**

**Input**:

mathematica

Copy code

Enter Number of Clients: 3

Enter Phone Numbers:

--> 15

--> 20

--> 25

Enter Hash Table Size: 5

If collision occurs, which collision resolution technique do you want to use?

1. Linear Probing

2. Double Hashing

3. Exit

Enter Your Choice: 1

**Output**:

lua

Copy code

Hash Value Key

0 ---> 15

1 ---> 20

2 ---> 25

**Case 3: Collision Resolved by Double Hashing**

**Input**:

mathematica

Copy code

Enter Number of Clients: 3

Enter Phone Numbers:

--> 15

--> 25

--> 35

Enter Hash Table Size: 7

If collision occurs, which collision resolution technique do you want to use?

1. Linear Probing

2. Double Hashing

3. Exit

Enter Your Choice: 2

**Output**:

lua

Copy code

Hash Value Key

1 ---> 15

2 ---> 25

4 ---> 35

**Case 4: Hash Table Full**

**Input**:

mathematica

Copy code

Enter Number of Clients: 4

Enter Phone Numbers:

--> 15

--> 25

--> 35

--> 45

Enter Hash Table Size: 3

If collision occurs, which collision resolution technique do you want to use?

1. Linear Probing

2. Double Hashing

3. Exit

Enter Your Choice: 1

**Output**:

sql

Copy code

Hash Value Key

0 ---> 15

1 ---> 25

2 ---> 35

Table is full, cannot insert 45.

**Case 5: Exit the Program**

**Input**:

mathematica

Copy code

Enter Number of Clients: 2

Enter Phone Numbers:

--> 10

--> 20

Enter Hash Table Size: 5

If collision occurs, which collision resolution technique do you want to use?

1. Linear Probing

2. Double Hashing

3. Exit

Enter Your Choice: 3

**Output**:

diff

Copy code

\*\*\* TERMINATED SUCCESSFULLY \*\*\*

These examples cover possible scenarios for **Hash Table Implementation** with different collision resolution techniques and outputs. Let me know if further clarifications are needed!

4o

**You said:**

go ahead

**ChatGPT said:**

ChatGPT

Let’s proceed with the input-output explanations for the next implementation in the uploaded file. Here’s the **Set Operations** code from **prac-2**:

**Set Operations: Input-Output Examples**

**Case 1: Insert Elements into Two Sets**

**Input**:

mathematica

Copy code

-----------DSAL PRACTICAL NO :- 02(A-4)----------

\*\*SET OPERATION

1:Insert

Enter your Choice: 1

Enter the number of elements in set1: 3

Enter the elements in set1: apple

Enter the elements in set1: orange

Enter the elements in set1: banana

Enter the number of elements in set2: 2

Enter the elements in set2: orange

Enter the elements in set2: mango

**Output**:

arduino

Copy code

Set1: {'apple', 'orange', 'banana'}

Set2: {'orange', 'mango'}

**Case 2: Get the Size of Sets**

**Input**:

ruby

Copy code

-----------DSAL PRACTICAL NO :- 02(A-4)----------

\*\*SET OPERATION

2:Size

Enter your Choice: 2

**Output**:

yaml

Copy code

Size of set1: 3

Size of set2: 2

**Case 3: Remove an Element from a Set**

**Input**:

mathematica

Copy code

-----------DSAL PRACTICAL NO :- 02(A-4)----------

\*\*SET OPERATION

3:Remove

Enter your Choice: 3

Remove element from set1 and set2 (1 for set1 & 2 for set2): 1

**Output**:

arduino

Copy code

{'orange', 'banana'}

**Case 4: Search for an Element in Sets**

**Input**:

sql

Copy code

-----------DSAL PRACTICAL NO :- 02(A-4)----------

\*\*SET OPERATION

4:Search

Enter your Choice: 4

Enter element you want to search: orange

**Output**:

sql

Copy code

set1 contains the element orange

set2 contains the element orange

**Case 5: Perform Union of Two Sets**

**Input**:

sql

Copy code

-----------DSAL PRACTICAL NO :- 02(A-4)----------

\*\*SET OPERATION

5:Union

Enter your Choice: 5

**Output**:

python

Copy code

Union is: {'apple', 'orange', 'banana', 'mango'}

**Case 6: Perform Intersection of Two Sets**

**Input**:

sql

Copy code

-----------DSAL PRACTICAL NO :- 02(A-4)----------

\*\*SET OPERATION

6:Intersection

Enter your Choice: 6

**Output**:

csharp

Copy code

Intersection is: {'orange'}

**Case 7: Check Subset Relationship**

**Input**:

sql

Copy code

-----------DSAL PRACTICAL NO :- 02(A-4)----------

\*\*SET OPERATION

7:Subset

Enter your Choice: 7

**Output**:

mathematica

Copy code

Subset is: False

**Case 8: Find the Difference Between Two Sets**

**Input**:

ruby

Copy code

-----------DSAL PRACTICAL NO :- 02(A-4)----------

\*\*SET OPERATION

8:Difference

Enter your Choice: 8

**Output**:

csharp

Copy code

Difference is: {'mango'}

**Case 9: Exit the Program**

**Input**:

mathematica

Copy code

-----------DSAL PRACTICAL NO :- 02(A-4)----------

\*\*SET OPERATION

9:Exit

Enter your Choice: 9

**Output**:

css

Copy code

----------------SUCCESSFULLY TERMINATED----------------

These input-output examples demonstrate how various operations like insertion, removal, union, intersection, and subset checking work on sets in this implementation. Let me know if you’d like to move on to the next section or need more details!

4o

**You said:**

give all of them

**ChatGPT said:**

ChatGPT

Let’s continue with input-output examples for all the remaining implementations in the uploaded file.

**3. General Tree (Book Hierarchy)**

**Case 1: Create and Display a Book Hierarchy**

**Input**:

mathematica

Copy code

Enter name of book: Data Structures

Enter no. of chapters in book: 2

Enter Chapter name: Arrays

Enter no. of sections in Chapter: Arrays: 2

Enter Section 1 name: Basics

Enter Section 2 name: Applications

Enter Chapter name: Linked Lists

Enter no. of sections in Chapter: Linked Lists: 1

Enter Section 1 name: Implementation

**Output**:

yaml

Copy code

-----Book Hierarchy---

Book title: Data Structures

Chapter 1: Arrays

Sections:

Basics

Applications

Chapter 2: Linked Lists

Sections:

Implementation

**4. Binary Search Tree (BST)**

**Case 1: Insert Nodes and Display Traversals**

**Input**:

mathematica

Copy code

Enter the data: 50

Do you want to insert more value: y

Enter the data: 30

Do you want to insert more value: y

Enter the data: 70

Do you want to insert more value: n

**Output**:

markdown

Copy code

The Total no. of nodes are: 2

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*INORDER\*\*\*\*\*\*\*\*\*\*\*\*\*\*

30 50 70

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*PREORDER\*\*\*\*\*\*\*\*\*\*\*\*\*\*

50 30 70

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*POSTORDER\*\*\*\*\*\*\*\*\*\*\*\*\*\*

30 70 50

**Case 2: Search for a Key**

**Input**:

mathematica

Copy code

Enter your key: 30

**Output**:

vbnet

Copy code

KEY FOUND

**Case 3: Get the Height of the Tree**

**Input**:

javascript

Copy code

Number of nodes in longest path:

**Output**:

Copy code

2

**5. Expression Tree**

**Case 1: Create an Expression Tree**

**Input**:

mathematica

Copy code

Enter prefix Expression: +a\*bc

**Output**:

yaml

Copy code

ab\*c+

Deleting node: +

Deleting node: \*

Deleting node: a

Deleting node: b

Deleting node: c

**6. Graph Implementation**

**Case 1: BFS and DFS**

**Input**:

yaml

Copy code

Enter no. of vertices: 4

Enter no. of edges: 4

Edges:

0 1

0 2

1 3

2 3

Enter initial vertex: 0

**Output**:

yaml

Copy code

The adjacency matrix of the graph is:

0 1 1 0

1 0 0 1

1 0 0 1

0 1 1 0

The BFS of the Graph:

0 1 2 3

The DFS of the Graph:

0 2 3 1

**7. Minimum Spanning Tree (Prim's Algorithm)**

**Case 1: Compute MST**

**Input**:

yaml

Copy code

Enter the no. of branches: 4

Enter the no. of connections: 5

Enter the end branches of connections:

1 2

Phone charges: 10

1 3

Phone charges: 15

2 4

Phone charges: 20

3 4

Phone charges: 25

2 3

Phone charges: 30

**Output**:

csharp

Copy code

Minimum cost connection is 1 -> 2 with charge: 10

Minimum cost connection is 1 -> 3 with charge: 15

Minimum cost connection is 2 -> 4 with charge: 20

The minimum total cost of connections: 45

**8. Optimal Binary Search Tree (OBST)**

**Case 1: Construct OBST**

**Input**:

perl

Copy code

Enter the no. of nodes: 3

Enter the probability for successful search:

p[1]: 0.2

p[2]: 0.5

p[3]: 0.3

Enter the probability for unsuccessful search:

q[0]: 0.1

q[1]: 0.1

q[2]: 0.1

q[3]: 0.1

**Output**:

less

Copy code

Optimal BST is:

w[0][3] :: 1.9

c[0][3] :: 2.1

r[0][3] :: 2

**9. AVL Tree**

**Case 1: Insert Nodes and Display Traversals**

**Input**:

mathematica

Copy code

Enter the keyword: data

Enter the meaning: information

Do you want to add another word? y

Enter the keyword: tree

Enter the meaning: structure

Do you want to add another word? y

Enter the keyword: algorithm

Enter the meaning: process

Do you want to add another word? n

**Output**:

yaml

Copy code

The keywords in ascending order are:

algorithm : process

data : information

tree : structure

The keywords in descending order are:

tree : structure

data : information

algorithm : process

**Case 2: Search for a Keyword**

**Input**:

css

Copy code

Enter the keyword you wish to search: data

**Output**:

swift

Copy code

The entered keyword is present in the AVL tree

**Case 3: Delete a Keyword**

**Input**:

css

Copy code

Enter the keyword to be deleted: data

**Output**:

mathematica

Copy code

Record deleted successfully.

**10. Priority Queue**

**Case 1: Insert and Display Priority Queue**

**Input**:

yaml

Copy code

Enter the number of patients: 3

Enter your name of the patient: John

Enter your Priority (3: serious, 2: non-serious, 1: general checkup): 3

Enter your name of the patient: Jane

Enter your Priority (3: serious, 2: non-serious, 1: general checkup): 2

Enter your name of the patient: Doe

Enter your Priority (3: serious, 2: non-serious, 1: general checkup): 1

**Output**:

rust

Copy code

Patient's Name - John Priority - 'Serious'

Patient's Name - Jane Priority - 'Non-serious'

Patient's Name - Doe Priority - 'Checkup'

**11. Sequential File Organization**

**Case 1: Add and Display Records**

**Input**:

mathematica

Copy code

Enter Roll Number: 1

Enter Name: Alice

Enter Division: A

Enter Address: NY

Enter Roll Number: 2

Enter Name: Bob

Enter Division: B

Enter Address: LA

**Output**:

css

Copy code

Roll Number Name Division Address

1 Alice A NY

2 Bob B LA

**Case 2: Search for a Record**

**Input**:

mathematica

Copy code

Enter Roll Number To Be Searched: 1

**Output**:

css

Copy code

Record Found

Roll Number Name Division Address

1 Alice A NY