Assignment 1 - Family Linked List

Creating a doubly linked list with my family members as elements, details : {Name, Age, Relation} are included as node values.

Creating the node class

In [1]:

```
# Making a class for node of doubly linked list.

class Node:

  def __init__(self, next=None, previous=None, name=None, relation=None, age=None):
    """Constructor to create a Node."""

    # Initializing the node with the given values.
    self.next = next                              # Pointer to next node.
    self.previous = previous                          # Pointer to previous node.

    # Data of the node.
    self.data1 = name                              # Name of the person.
    self.data2 = relation                          # Relation with the person.
    self.data3 = age                              # Age of the person.
  def __del__(self):                              # Destructor to delete the node.
    pass
```

Creating the class for Doubly Linked List(Family_DLL).

- defining functions to insert nodes into the list and display the list.

In [2]:

```
# Creating a class for doubly linked list.

class Family_DLL:

  def __init__(self) -> None:
    """Constructor to create a empty doubly linked list."""

    self.head = None                              # Head of the empty doubly linked list.

  def add_front(self, name, relation, age):
    """Function to add a node at the front of the doubly linked list."""

    # Creating a new node and put the data in it.
```

```
        new_node = Node(name=name, relation=relation, age=age)

        new_node.next = self.head                      # Making next of new node as head.

        if self.head is not None:
            self.head.prev = new_node                  # Changing previous of head node to new node.

        self.head = new_node                         # Moving the head to point to the new node.

    def add_after(self, prev_node, name, relation, age):
        """Function to add a node after a given node."""

        # Checking if the given previous node exists.
        if prev_node is None:
            print("The given previous node must be in DLL.")
            return

        # Creating a new node and put the data in it.
        new_node = Node(name=name, relation=relation, age=age)

        new_node.next = prev_node.next                    # Making next of new node as next of
previous node.
        prev_node.next = new_node                     # Making next of previous node as new node.
        new_node.previous = prev_node                   # Making previous of new node as previous
node.
        if new_node.next is not None:
            new_node.next.previous = new_node                # Changing previous of new node's next
node.

    def add(self, name, relation, age):
        """Function to add a node at the back of the doubly linked list."""

        # Creating a new node and put the data in it.
        new_node = Node(name=name, relation=relation, age=age)

        last = self.head                          # Initializing the last node as head.

        new_node.next = None                         # Making next of new node as None.

        if self.head is None:                        # If the DLL is empty, then make the new node as
head.
            new_node.previous = None
            self.head = new_node
            return

        while last.next is not None:                    # Else traverse till the last node.
            last = last.next
        last.next = new_node                          # Change the next of last node.
        new_node.previous = last                        # Make last node as previous of new node.

        return

    def printDLL(self, node):
        """Function to print the doubly linked list."""

        strng = "My Family: "                         # String to store the family members.

        # Traversing the doubly linked list.
        while node is not None:
            strng += f"<==> [Name: {node.data1}, Relation: {node.data2}, Age: {node.data3}] "   #
Adding the data of the node to the string.

            # Moving to the next node.
            last = node
            node = node.next

        print(strng)                              # Printing the string.

    def __del__(self):                            # Destructor to delete the doubly linked list.
        pass
```

Creating the Doubly Linked List

In [3]:

```
family = Family_DLL()                             # Creating a doubly linked list representing my
Family.

# Adding the parents of the family.

family.add_front("Prakash Chand Meena", "Father", 56)
family.add("Vimla Meena", "Mother", 52)

# Adding the siblings of the family.
family.add_after(family.head,"Manobal Singh Bagady", "Me", 18)
family.add_after(family.head.next,"Kushal Meena", "Sister", 28)

# Adding the spouses of the siblings of the family.
family.add_after(family.head.next,"Ashok Meena", "Brother-in-Law", 32)

# Adding the children of the siblings.
family.add_after(family.head.next.next,"Kuvika", "Niece", 2)
family.add_after(family.head.next.next.next,"Kunal", "Nephew", 2)
```

Printing the Family Doubly Linked List

In [4]:

```
family.printDLL(family.head)
```

```
My Family: <==> [Name: Prakash Chand Meena, Relation: Father, Age: 56] <==> [Name: Manobal Singh Bagady, Relation: Me, Age: 18] <==> [Name: Ashok Meena, Relation: Brother-in-Law, Age: 32] <==> [Name: Kuvika, Relation: Niece, Age: 2] <==> [Name: Kunal, Relation: Nephew, Age: 2] <==> [Name: Kushal Meena, Relation: Sister, Age: 28] <==> [Name: Vimla Meena, Relation: Mother, Age: 52]
```

Way to Link the family members' doubly-linked list based on their relationship:

- Parent(Male) <=> Children <==> Children of Children <==> Spouses of Children <==> Parent(Female)
- To Achieve this using Linked List is not possible because it is a linear data structure, but we can achieve this using tree/graph data structure.