```
#include <bits/stdc++.h>
#include <iostream>
#include <string>
using namespace std;

//Counter to count elements in queue
int ct=0;

//Tree Node Structure -> Has data members and a parametrized constructor to initialize values
struct Tnode
{
   string fname;
   string lname;
   int age;
   int year;
   Tnode *left;
   Tnode *right;

   //Parametrized constructor
   Tnode(string fnam, string lnam, int a, int yr)
   {
      fname = fnam;
      lname = lnam;
      age = a;
      year = yr;
      left = NULL;
      right = NULL;
   }
};
```

```
//Tree Data Structure
struct Tree
{
   //Root node
   Tnode *root;

   //Constructor sets root to NULL when tree is created
   Tree()
   {
      root = NULL;
   }

   //Checks if tree is empty
   bool isTreeEmpty()
   {
      //Empty if root is NULL
      if(root==NULL)
         return true;
      else
         return false;
   }

   //Inserts node to make a BST -> Based on age data of each node
   void insertNode(Tnode *new_node)
   {
      //If tree is empty then new node becomes the root node
      if(isTreeEmpty())
      {
         root = new_node;
         return;
```

```
}
else
{
   //Traverse tree till the end using temp node -> temp represents the current node
   Tnode *temp = root;
   while(temp!=NULL)
   {
      //If new node data is the same as temp node data then return -> no duplicates
      if(new_node->age == temp->age)
      {
         cout << "Value already exists, Insert another node" << endl;
         return;
      }

      //If new node is smaller then temp and temp is a leaf node
      else if((new_node->age < temp->age) && (temp->left == NULL))
      {
         //Append new node on the left of temp
         temp->left = new_node;
         break;
      }

      //If new node is smaller then temp and temp is not a leaf node
      else if(new_node->age < temp->age)
      {
         //Traverse to the next left node
         temp = temp->left;
      }

      //If new node is larger then temp and temp is a leaf node
```

```
            else if((new_node->age > temp->age) && (temp->right == NULL))
            {
               //Append new node on the right of temp
               temp->right = new_node;
               break;
            }

            //If new node is larger then temp and temp is a not leaf node
            else
            {
               //Traverse to the next right node
               temp = temp->right;
            }
         }
      }
   }

   //PreOrder DFS -> NODE LEFT RIGHT
   void printPreOrder(Tnode *temp)
   {
      //If n is NULL then return
      if(temp==NULL)
         return;
      //Print data out in order NLR using recursion
      cout << "(First Name: " << temp->fname << ", Last Name: " << temp->lname << ", Age: " << temp-
>age << ", Year: " << temp->year << ") ---> " << endl;
      printPreOrder(temp->left);
      printPreOrder(temp->right);
   }
```

```
   //InOrder DFS -> LEFT NODE RIGHT
   void printInOrder(Tnode *temp)
   {
      if(temp==NULL)
         return;
      //Print data out in order LNR using recursion
      printInOrder(temp->left);
      cout << "(First Name: " << temp->fname << ", Last Name: " << temp->lname << ", Age: " << temp->age << ", Year: " << temp->year << ") ---> " << endl;
      printInOrder(temp->right);
   }

   //PostOrder DFS -> LEFT RIGHT NODE
   void printPostOrder(Tnode *temp)
   {
      if(temp==NULL)
         return;
      //Print data out in order LRN using recursion
      printPostOrder(temp->left);
      printPostOrder(temp->right);
      cout << "(First Name: " << temp->fname << ", Last Name: " << temp->lname << ", Age: " << temp->age << ", Year: " << temp->year << ") ---> " << endl;
   }
};

//Node Structure - Has data members and a parametrized constructor to initialize values
//For Queue Stack & Linked List
struct node
{
   string fname;
```

```
    string lname;
    int age;
    int year;
    node *next;

    //Parametrized constructor
    node(string fnam, string lnam, int a, int yr)
    {
        fname = fnam;
        lname = lnam;
        age = a;
        year = yr;
        next = NULL;
    }
};

//Queue Data Structure
struct Queue
{
    //Front and Rear nodes of queue
    node *front, *rear;

    //Constructor sets values of front and rear to NULL as queue is empty when we create it
    Queue()
    {
        front = NULL;
        rear = NULL;
    }

    //Checks if queue is empty
```

```
bool isQueueEmpty()
{
   //Empty if front and rear are NULL
   if(front==NULL && rear==NULL)
      return true;
   else
      return false;
}

//Enqueues Node into queue
void enqueue(node *temp)
{
   //If the queue is empty then front and rear both point to the new node
   if(isQueueEmpty())
   {
      front = rear = temp;
      //Increment count
      ct++;
      return;
   }

   //Add new node at end of queue and change rear to point to new node
   rear->next = temp;
   rear = temp;
   //Increment count
   ct++;
}

//Dequeues Node from queue
node* dequeue()
```

```
{
    //If queue is empty print out UNDERFLOW
    if(isQueueEmpty())
    {
        cout << "UNDERFLOW" << endl;
        return NULL;
    }

    //Element to be dequeued is stored in temp and front pointer is moved to the next node
    node *temp = front;
    front = front->next;

    //If front is NULL after above operation it implies queue is empty and so rear is also changed to
NULL
    if(front==NULL)
    {
        rear=NULL;
    }

    //Decrement count
    ct--;

    //Return Dequeued Element
    return temp;
}

//Displays Queue Data
void displayQueue()
{
    //If queue empty
```

```
    if(isQueueEmpty())
    {
      cout << "Queue Empty!" << endl;
      return;
    }
    else
    {
      //Traverse Queue element by element and print the data members
      node *temp = front;
      while(temp!=NULL)
      {
        cout << "(First Name: " << temp->fname << ", Last Name: " << temp->lname << ", Age: " << temp->age << ", Year: " << temp->year << ") ---> " << endl;
        temp = temp->next;
      }
    }
  }

  void displayQueueNode(node *temp)
  {
    cout << "(First Name: " << temp->fname << ", Last Name: " << temp->lname << ", Age: " << temp->age << ", Year: " << temp->year << ") ---> " << endl;
  }

  void emptyQueue()
  {
    while(!isQueueEmpty())
    {
      node *temp = dequeue();
      free(temp);
```

```
        }
        front = NULL;
        rear = NULL;
        free(front);
        free(rear);
        return;
    }
};

//Stack Data Structure
struct Stack
{
    //Top node -> represents the topmost node in the stack
    node *top;

    //Constructor sets top node to NULL when stack is created
    Stack()
    {
        top = NULL;
    }

    //Checks if stack is empty
    bool isStackEmpty()
    {
        //Empty is top is NULL
        if(top==NULL)
        {
            return true;
        }
        else
```

```
    {
        return false;
    }
}

//Pushes Node into stack
void push(node* new_node)
{
    //Pushes node to the top of the stack and then makes the new node the new TOP node
    new_node->next = top;
    top = new_node;
    return;
}

//Pops top node from stack
node *pop()
{
    //If stack is empty -> UNDERFLOW
    if(isStackEmpty())
    {
        cout << "Underflow" << endl;
        return top;
    }

    else
    {
        //Store top node in temp and then move the top node to the next node
        node *temp = top;
        top = top->next;
        //Unlink the old top node from the stack
```

```
        temp->next = NULL;
        //Return popped node
        return temp;
    }
}

//Display Stack data
void displayStack()
{
    //If stack is empty
    if(isStackEmpty())
    {
        cout << "Stack Empty!" << endl;
        return;
    }
    else
    {
        //Traverse stack element by element and display node data
        node *temp = top;
        while(temp!=NULL)
        {
            cout << "(First Name: " << temp->fname << ", Last Name: " << temp->lname << ", Age: " << temp->age << ", Year: " << temp->year << ") ---> " << endl;
            temp = temp->next;
        }
    }
}

//Empty the stack
void emptyStack()
```

```
    {
        //Till stack is not empty
        while(!isStackEmpty())
        {
            //Pop each node and free memory for that node
            node *temp = pop();
            free(temp);
        }

        //Set top back to NULL and free top memory
        top=NULL;
        free(top);
        return;
    }
};

//Linked List Data Structure
struct LinkedList
{
    //Head node
    node *head;

    //Constructor sets head to NULL when LL is created
    LinkedList()
    {
        head = NULL;
    }

    //Checks if LL is Empty
    bool isLLEmpty()
```

```
{
   //Empty if head is NULL
   if(head==NULL)
      return true;
   else
      return false;
}

//Inserts Node into LL
void insertNode(node *new_node)
{
   //If LL is empty
   if(isLLEmpty())
   {
      //Insert new node as the head node
      head = new_node;
      return;
   }

   else
   {
      //Traverse till the end of the LL
      node *temp = head;
      while(temp->next!=NULL)
      {
         temp = temp->next;
      }
      //Append new node at the end of the LL
      temp->next = new_node;
      return;
```

```
    }
}

//Gets the tail node of the LL -> The last node
node *getTail(node *currptr)
{
    //Traverse till the end
    while(currptr!=NULL && currptr->next!=NULL)
        currptr = currptr->next;

    //Return last node(tail)
    return currptr;
}

node *Partition(node *head, node *end, node **NewHead, node **NewEnd)
{
    node *pivot = end;
    node *prev = NULL, *cur = head, *tail = pivot;

    while(cur!=pivot)
    {
        if(cur->age < pivot->age)
        {
            if((*NewHead) == NULL)
                (*NewHead) = cur;

            prev = cur;
            cur = cur->next;
        }
        else
```

```
        {
            if(prev)
            {
                prev->next = cur->next;
            }
            node *temp = cur->next;
            cur->next = NULL;
            tail->next = cur;
            tail = cur;
            cur = temp;
        }
    }
    if((*NewHead) == NULL)
        (*NewHead) = pivot;

    (*NewEnd) = tail;
    return pivot;
}

node *quickSortRecur(node *head, node *end)
{
    if(!head || head==end)
        return head;

    node *newHead = NULL, *newEnd = NULL;

    node *pivot = Partition(head, end, &newHead, &newEnd);

    if(newHead != pivot)
    {
```

```
        node *temp = newHead;
        while(temp->next!=pivot)
            temp = temp->next;
        temp->next = NULL;

        newHead = quickSortRecur(newHead, temp);

        temp = getTail(newHead);
        temp->next = pivot;
    }

    pivot->next = quickSortRecur(pivot->next, newEnd);

    return newHead;
}

//QuickSort Function Call
void quickSort(node **head)
{
    //Sorts LL from head to end using quick sort
    *head = quickSortRecur(*head, getTail(*head));
    return;
}

//Display LL data
void displayLinkedList()
{
    //If LL is empty
    if(isLLEmpty())
    {
```

```
        cout << "Linked List Empty!" << endl;
        return;
      }
      else
      {
        //Traverse LL element by element and output node data for each node
        node *temp = head;
        while(temp!=NULL)
        {
          cout << "(First Name: " << temp->fname << ", Last Name: " << temp->lname << ", Age: " << temp->age << ", Year: " << temp->year << ") ---> " << endl;
          temp = temp->next;
        }
      }
   }
};

int main()
{
   Queue q1, q2, q3;
   Stack s1;
   LinkedList l1;
   Tree t1;

   node *node1 = new node(“Rahul", "Goel", 19, 2001);
   node *node2 = new node(Devansh ", "Agarwal", 20, 2000);
   node *node3 = new node("Yajat", "Raisinghani", 18, 2002);
   node *node4 = new node(“Ishvak", "Taneja", 34, 2004);
   node *node5 = new node("Aryan", "Jain", 12, 1999);
   node *node6 = new node("Eshaan", "kapoor", 10, 1989);
```

```
node *node7 = new node("Taru", "Gupta", 64, 1987);
node *node8 = new node("Anand", "Modi", 8, 1996);

q1.enqueue(node1);
q1.enqueue(node2);
q1.enqueue(node3);
q1.enqueue(node4);
q1.enqueue(node5);
q1.enqueue(node6);
q1.enqueue(node7);
q1.enqueue(node8);

q1.displayQueue();
cout << endl;

int cnt = ct;
while(cnt--)
{
   node *temp = q1.dequeue();
   q1.displayQueueNode(temp);
   q2.enqueue(temp);
}

cout << endl;
q2.displayQueue();

while(!q2.isQueueEmpty())
{
   s1.push(q2.dequeue());
}
```

```
cout << endl;
s1.displayStack();

while(!s1.isStackEmpty())
{
   q1.enqueue(s1.pop());
}

cout << endl;
q1.displayQueue();
cout << endl;

while(!q1.isQueueEmpty())
{
   node *temp = q1.dequeue();
   q1.displayQueueNode(temp);
   q3.enqueue(temp);
}

cout << endl;
q3.displayQueue();
cout << endl;

int count = 0;
while(q3.isQueueEmpty())
{
   node *temp = q3.dequeue();
   l1.insertNode(temp);
   Tnode *temp1 = new Tnode(temp->fname, temp->lname, temp->age, temp->year);
```

```
    t1.insertNode(temp1);
    cnt++;

    if(cnt==8)
    {
        t1.printPreOrder(temp1);
        cout << endl;
        t1.printPostOrder(temp1);
        cout << endl;
        t1.printInOrder(temp1);
        cout << endl;
    }
}

l1.displayLinkedList();
cout << endl;
l1.quickSort(&node8);
l1.displayLinkedList();
cout << endl;

int option, age, year;
string fname, lname;

do
{
    cout << "Enter Option: 0 to exit: " << endl;
    cout << "1. To insert a node" << endl;
    cin >> option;

    switch(option)
```

```
        {
        case 0:
            cout << "Quitting" << endl;
            break;

        case 1:
            cout << "Enter First and Last name and age and year of birth: " << endl;
            cin >> fname >> lname >> age >> year;
            node *new_node = new node(fname, lname, age, year);
            l1.insertNode(new_node);
            l1.quickSort(&node8);
            l1.displayLinkedList();
            break;
        }
    }while(option!=0);

    return 0;
}
```