DSA ASSIGNMENT - 3

Done by
UNDELA PAVAN KUMAR (RA1911026010032)
Thontepu Manikanta (RA1911026010033)
Ryalli Ajay(RA1911026010034)
Chintala Aditya(RA1911026010050)

INTRODUCTION :

Language Used : C++
Compiler Used : g++ version 5.4.0
Abstract Data Types Used in the code is

1)Arrays
2)Linked List
3)Stack
4)Queue
5)Tree

Some of the used functions in the code are String and file functions.

REPORT:

For easier explanation, we will begin from the main function. Initially, we set up a Queue 'q'. Then we open the file to read data from. Here, the file is "input.txt". The while loop runs until end of file is reached. Within the while loop, we read one line at a time and divide it with respect to comma and space. This line is enqueued in queue 'q'. The enqueued line is dequeued, printed and enqueued again. After this we wait for user input to press any key to go next step.

Next, we declare a stack 'st'. We dequeue each element from qB and push it in the stack. Then we pop it and enqueue in another queue qC. Since stack follows FILO order, we now have a reversed queue.

This queue is dequeued at each step and re-enqueued after displaying Each element.We again wait for the user input that is press any key to move to next step.

Next we are declaring a Binary tree bt and input values in it from the queue. We are displaying the elements in it using preorder and postorder traversal. After each print operation, we are awaiting user input. Using inorder traversal we are transferring elements from the binary tree to a linked list and sorting it using quick sort. The final function of this program is taking user input and adding it the the linked list. We will print the list after the user is done entering the details. Finally we will wait for the user input and then terminate the program.

As far as each ADT is concerned we have the following functions:

1) LinkedList :

```
class LinkedList{

private:
    Node *head;
    int size = 0;

public:
    // constructor for LinkedList class
    LinkedList(){…
    // To change the head of linked list
    void changeHead(Node *val){…

    // to assign the data to the node
    void setData(string fst, string lst, int a, string db, Node *newNode){…
    // to create the first node of the linkedlist
    void firstNode(string fst, string lst, int a, string db){…
    // To push a node at the starting of the linkedlist
    void prepend(string fst, string lst, int a, string db){…
    // To push a node at the end of the linkedlist
    void append(string fst, string lst, int a, string db){…
    // removing a element fron the ending of the queue
    Node *pop(){…

    //removing element from the starting of the queue
    Node *popFirst(){…

    // To know the current size of the queue(linkedlist implementation)
    int sizeVal(){…

    //To know the starting address of the queue(linkedList implementation)
    Node *headVal(){…
};
```

LinkedList():

A constructor that initializes the head to NULL.

• changeHead():

Used to change the head to a given pointer.

• setData():

It will save new data(name, age, dob) in a new node.

• firstNode():

Used for creating the first node.

• Prepend():

Adds a new node in the front of the list.

• Append():

Adds a new node in the end of the list.

• Pop():

Removes an element from the list.

• popFirst():

Used to pop the head from the list.

• sizeVal():

Returns the size of the list.

• HeadVal():

Returns a pointer to the head of the list.

2) STACK :

```
class Stack{
private:
    LinkedList ll;
    int size = 0;

public:

    void push(string fst, string lst, int a, string db){ ···

    // To push a node into a Stack(Linkedlist implementation)
    void push(Node *node){ ···
    // To remove the node in a Stack
    Node *pop(){ ···

    // To know the size of the Stack;
    int sizeVal(){ ···
    // T0 return the address of the starting node
    Node *headVal(){ ···
};
```

• Push():

Pushes a new element in the stack. There are two push functions, one will take the data as argument and add it to a new node, another will take a node as an argument and at it to the stack.

• Pop():

Pops the topmost element in the stack.

• sizeVal():

Returns the size of the stack.

• HeadVal():

Returns a pointer to the head of the stack.

3) Queue :

```
class Queue{
private:
    LinkedList ll;
    int size = 0;

public:
    //Add the element to queue(using linkedlist implementation)
    void enqueue(string fst, string lst, int a, string dob){ ...

    void enqueue(Node *node){ ...

    Node *dequeue(){ ...

    int sizeVal(){ ...

    Node *headVal(){ ...

    void disp(){ ...
};
```

- Enqueue():

Used to add a new element in the queue. There are two enqueue functions, one will take the data as argument and add it to a new node, another will take a node as an argument and at it to the queue.

- dequeue():

Used to remove a queue element.

- sizeVal():

Returns the size of the queue.

- HeadVal():

Returns a pointer to the head of the queue.

- Disp():

Prints the elements in a queue.

4) Tree :

In tree in this problem we used only the binary tree.
Binary Tree :

```
class BinaryTree{

public:
    Node *newNode(string fst, string lst, int a, string db){ ...

    void traversePreOrder(Node *temp){ ...

    void traverseInOrder(Node *temp){ ...

    void traversePostOrder(Node *temp){ ...

    Node *InsertNode(Node *root, string fst, string lst, int a, string db){ ...
};
```

• newNode():

Creates a new tree node.

• traversePreOrder():

Recursive function for preorder traversal. (In this traversal method, the root node is visited first, then the left subtree and finally the right subtree.)

• traverseInOrder():

Recursive function for inorder traversal.

(In this traversal method, the left subtree is visited first, then the root and later the right sub-tree.)

• traversePostOrder():

Recursive function for postorder traversal. (In this traversal method, the root node is visited last, hence the name. First we traverse the left subtree, then the right subtree and finally the root node.)

• InsertNode(): Inserts a new node.

Now Other functions Used in this code are:

For the Sorting:

```
> struct Node *getTail(struct Node *cur){ ...

> struct Node *partition(struct Node *head, struct Node *end, struct Node **newHead, struct Node **newEnd){ ...

> struct Node *quickSortRecur(struct Node *head, struct Node *end){ ...

> void quickSort(struct Node **headRef) ...
```

• Partition():

Partitions the list taking the last element as the pivot.

• quickSortRecur():

Quick sort function using recursion.

• quickSort():

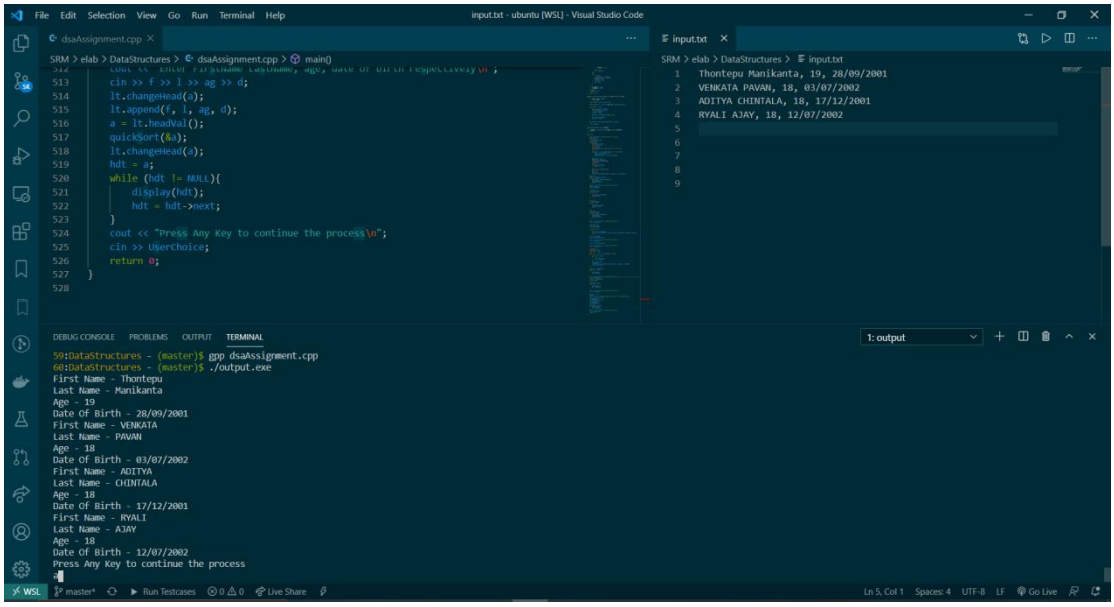
The main function for quicksort which calls the quickSortRecur function. This is a wrapper over the recursive function.

HOW TO USE THE PROGRAM :

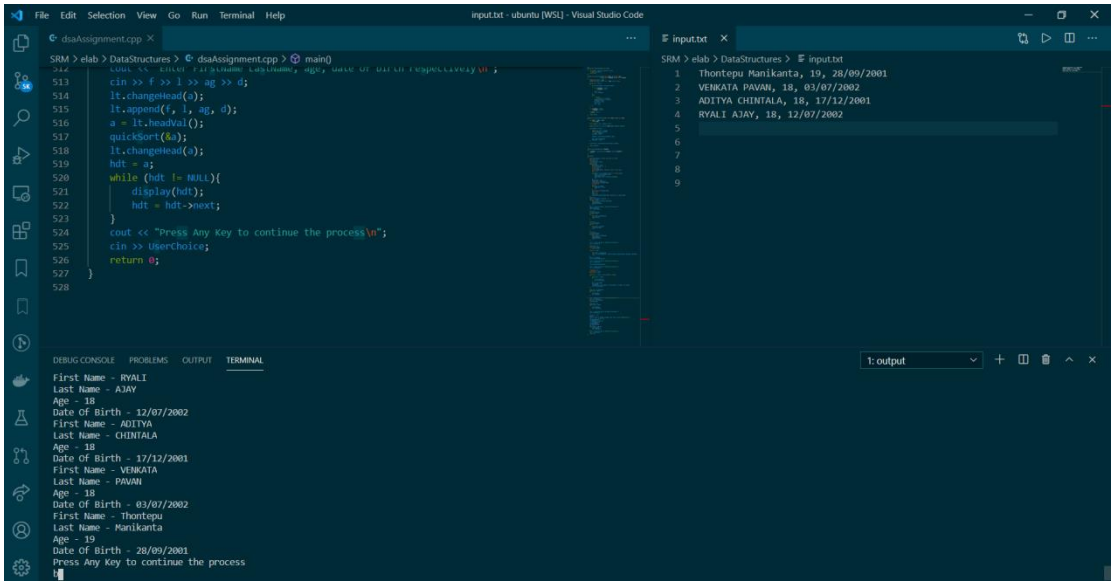
The name of the input TXT file should be input.txt, if it is of any other name it should be updated in the code. In the end of the code provide another name, age and date when prompted. No further input will be required from the keyboard.

The input should in a proper format. As in the question it should have comma seperated details.
Eg : FirstName LastName, Age, DateOfBirth

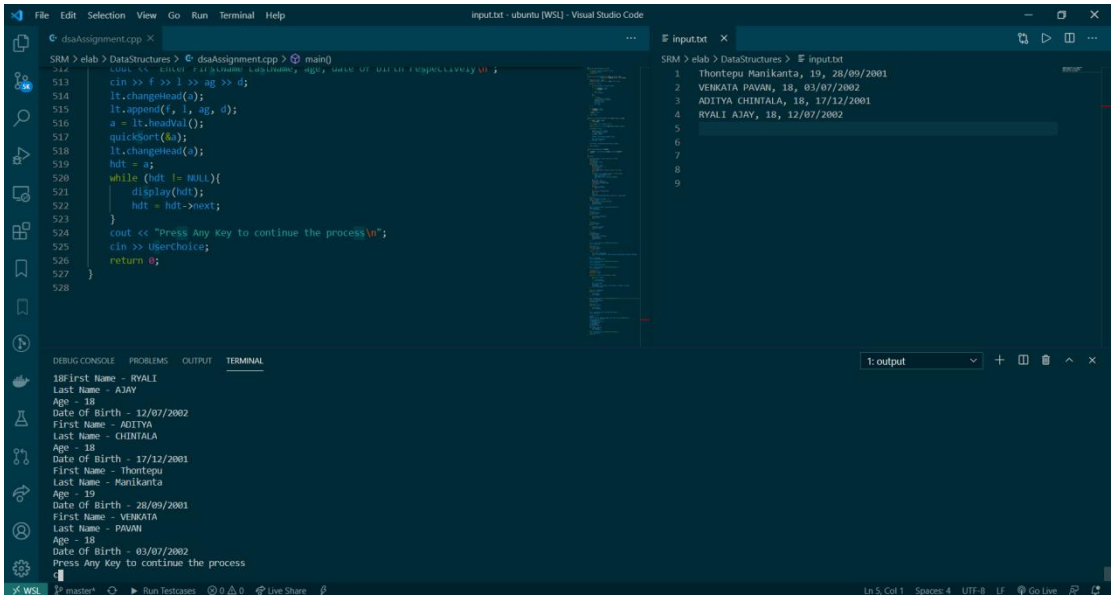
SAMPLE INPUT AND OUTPUT :



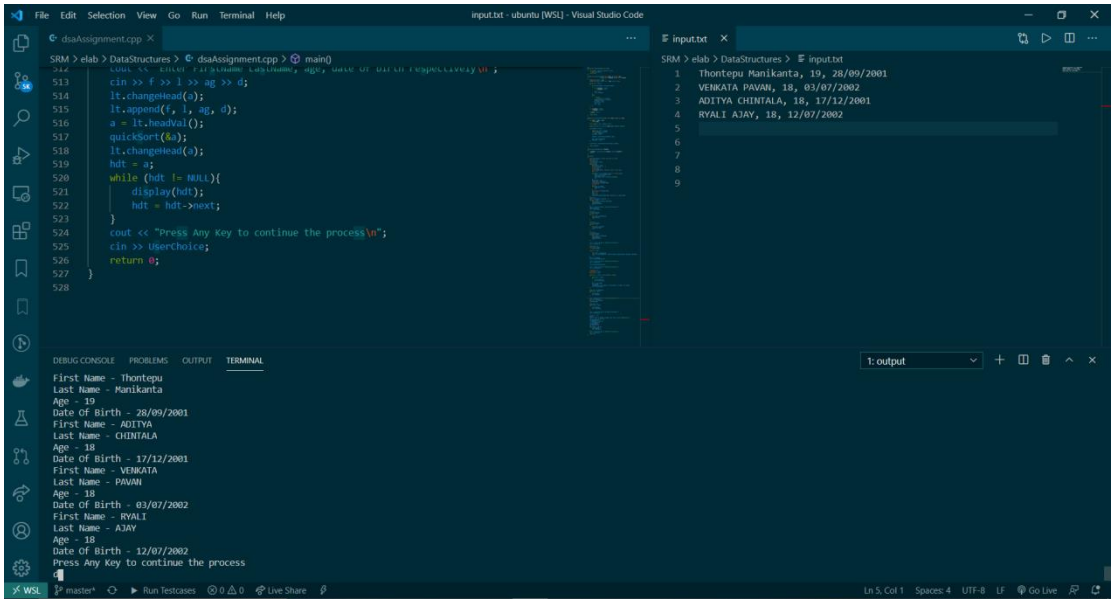
Output for Taking input from the file.Displaying And Storing in the queue.



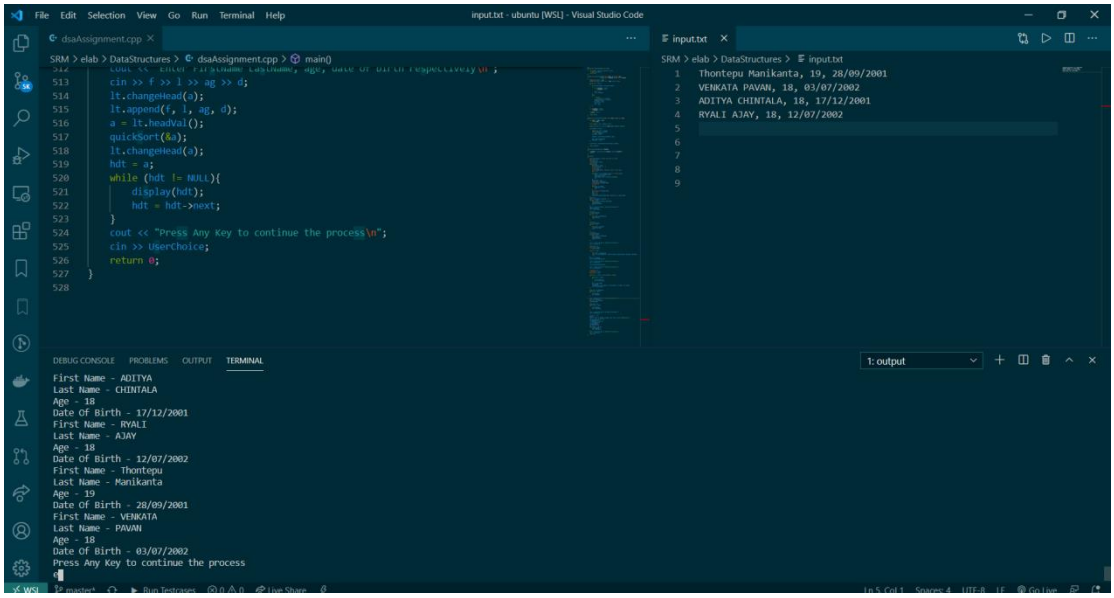
Output [conti]: reversing the queue by using stack.And displaying result



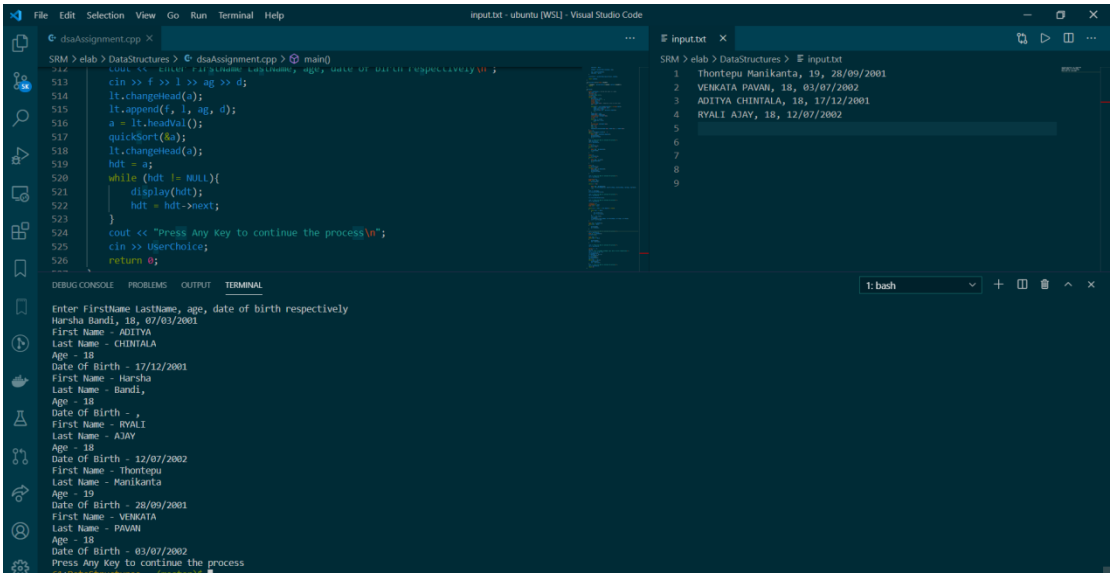
Output for the binary tree after preorder Traversal
Pre pos in



Output For the binary tree after postorder Traversal



Output For the binary tree after Inorder Traversal.



Final Output After sorting all the elements based on the first name.

Contribution:

Pavan :
By Writing the class of Linked List and implementing it from the main function.

Ajay:
By Writing the class of Stack by using linked list implementation and doing the appropriate operations from the main function.

Manikanta :
By Writing the class for the binary tress and implementing preorder,postorder,inorder traversals.

Aditya :
Writing class for the queues and doing operations on the queue.Like taking input from user and storing the data.

All together done the report,Sorting technique ,checking the output for various inputs, making the README file,compilation.