STOCK MARKET TRADING APPLICATION

STATEMENT:

To create a stock trading application using various data structure techniques in C++.

FEASIBILITY:

The project can be implemented using data structures concepts. It can be made user friendly and presentable using appropriate front end user interface.

BENEFITS:

- The project proves to be beneficial for both the companies selling their stock and the people who want to buy stock as it simplifies the processes involved in trading to a great extent.
- The people who want to buy stocks can do so with ease as they are provided with an user friendly interface to check the stock values of various companies and to buy and sell multiple stocks.
- The companies are also benefited as they can find buyers for their stocks with ease.
- Easily monitor current stock rates and details.
- Users can easily manage and keep track of their recent activities.

ABSTRACT:

The main motive of the project is to make the process of trading easier for the general public. The project should protect the credentials and personal details of the users to make it trustworthy. It should also ensure that the users don't face any issues while buying and selling stocks. The user should be able to get timely updates and follow the stock trend with ease so that he/she will be able to buy the desired stocks. The companies should also be provided with the provision to release the stocks that they want to sell.

INPUT DATA:

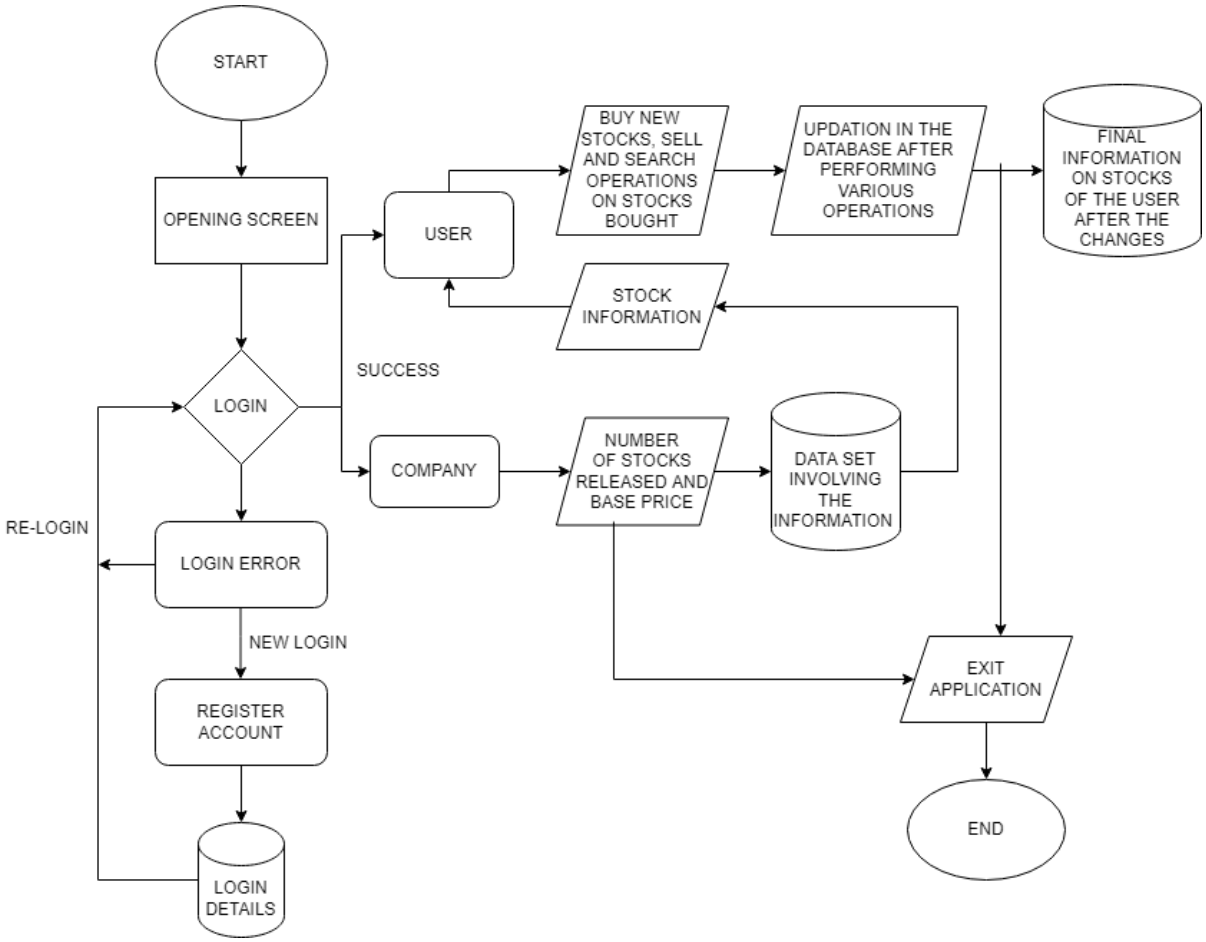
The input for the project is as follows:

• Personal data including bank account details and login credentials of the user.

• Number of stocks released by the companies for sale.

• Buying and selling of stocks by the user.

WORK FLOW:



IDENTIFICATION OF MODULES:

- Doubly linked list.
- AVL trees.
- Structure for implementing DLL and AVL.
- Class corresponding to User and Admin.

SOURCE CODE:

```
#include <iostream>
#include<cstdlib>
using namespace std;
struct Node
{
    string name[25];
    int data[25];
    int user_id;
    int password;
    int i=0;
    int sbought=0;
    int ssold=0;
    Node* next;
    Node* prev;
};
class User
{
    private:
        Node* head;
    public:
        User(){
            head=NULL;
        }
        //inserting new stock/user
        void push(Node** head_ref,string new_name,int new_data)
        {
            int x,y,z;
            cout<<"\n1. New user\n2. Existing user\nEnter your
choice: ";
            cin>>x;
            if(x==1){
                Node* new_node = (Node*)malloc(sizeof(struct Node));
                new_node->user_id=rand()%10000;
                new_node->password=rand()%10000;
                cout<<"Your user ID is "<<new_node->user_id<<" and
password is "<<new_node->password<<endl;
                new_node->name[new_node->i]= new_name;
                new_node->data[new_node->i]= new_data;
                new_node->sbought+=new_data;
                new_node->prev = NULL;
                new_node->next = (*head_ref);
                if ((*head_ref) != NULL) {
                    (*head_ref)->prev = new_node;
                }
                (*head_ref) = new_node;
                cout<<"\nStock(s) bought successfully!";
                new_node->i++;
            }
            else{
                Node* node;
                cout<<"Enter user ID:";
                cin>>y;
                cout<<"Enter password: ";
                cin>>z;
```

```
        node=*head_ref;
        while (node != NULL)
        {
            if(y==node->user_id && z==node->password){
                node->name[node->i]=new_name;
                node->data[node->i]=new_data;
                node->sbought+=new_data;
                node->i++;
                node=NULL;
            }
            else{
                node = node->next;
            }
            cout<<"\nStock(s) bought successfully!";
        }
    }
}
//displaying
void disp(Node* node)
{
    while (node != NULL)
    {
        for(int k=0;k<node->i;k++){
            cout<<"\nUser ID: "<<node->user_id;
            cout<<"\nName of the company: "<<node->name[k];
            cout<<"\nNumber of stocks: "<<node->data[k];
        }
        node =node->next;
    }
}
//selling stock
void sell(Node ** node, int x,int y,string name,int no)
{
    Node * h=*node;
    while(h!=NULL){
        if(h->user_id==x && h->password==y){
            for(int k=0;k<h->i;k++){
                if(name==h->name[k]){
                    h->ssold+=no;
                    h->data[k]=h->data[k]-no;
                }
            }
            h=NULL;
        }
        else
            h=h->next;
    }
}
//searching user details
void search(Node** head_ref,int x)
{
    Node* temp = *head_ref;
    int pos = 0;
    while (temp->user_id != x && temp->next != NULL) {
        pos++;
        temp = temp->next;
```

```
                }
                if (temp->user_id != x){
                    cout<<"\nUser not found! ";return;
                }
                cout<<"\nUser ID: "<<x;
                for(int k=0;k<temp->i;k++){
                    cout<<"\nName of company: "<<temp->name[k];
                    cout<<"\nStocks bought: "<<temp->data[k]<<endl;
                }
                cout<<"Total number of stocks bought by the user:
"<<temp->sbought;
                cout<<"\nTotal number of stocks sold by user:
"<<temp->ssold;
        }
        //delete user
        void deluser(Node ** node,int x,int y){
            Node* h =*node;
            while(h!=NULL){
                if(h->user_id==x && h->password==y && h->sbought==h-
>ssold){
                    if (h->next != NULL)
                        h->next->prev = h->prev;
                    if (h->prev != NULL)
                        h->prev->next = h->next;
                    free(h);
                    cout<<"\nUser deleted succesfully! ";
                    h=NULL;
                }
                else
                    h=h->next;
            }

        }
};
struct Node1{
   public:
      int num;
      float price;
      string name;
      Node1 *left;
      Node1 *right;
      int height;
};
class Admin
{
    public:
        Node1 *root = NULL;
        int height(Node1 *N) {
          if (N == NULL)
            return 0;
          return N->height;
        }
        int max(int a, int b) {
          return (a > b) ? a : b;
        }
        Node1 *newNode(int num,float price,string name) {
```

```
  Node1 *node = new Node1();
  node->num=num;
  node->price=price;
  node->name=name;
  node->left = NULL;
  node->right = NULL;
  node->height = 1;
  return (node);
}
Node1 *rightRotate(Node1 *y) {
  Node1 *x = y->left;
  Node1 *T2 = x->right;
  x->right = y;
  y->left = T2;
  y->height = max(height(y->left),height(y->right)) + 1;
  x->height = max(height(x->left), height(x->right)) + 1;
  return x;
}
Node1 *leftRotate(Node1 *x) {
  Node1 *y = x->right;
  Node1 *T2 = y->left;
  y->left = x;
  x->right = T2;
  x->height = max(height(x->left),height(x->right)) + 1;
  y->height = max(height(y->left),height(y->right)) + 1;
  return y;
}
int getBalanceFactor(Node1 *N) {
  if (N == NULL)
    return 0;
  return height(N->left) -
       height(N->right);
}
Node1 *insertNode(Node1 *node,int num,float price,string
name) {
  if (node == NULL)
    return (newNode(num,price,name));
  if (name < node->name)
    node->left = insertNode(node->left,num,price,name);
  else if (name > node->name)
    node->right = insertNode(node->right,num,price,name);
  else
    return node;
  node->height = 1 + max(height(node->left),
                height(node->right));
  int balanceFactor = getBalanceFactor(node);
  if (balanceFactor > 1) {
    if (name < node->left->name) {
      return rightRotate(node);
    } else if (name > node->left->name) {
      node->left = leftRotate(node->left);
      return rightRotate(node);
    }
  }
  if (balanceFactor < -1) {
    if (name > node->right->name) {
```

```
            return leftRotate(node);
          } else if (name < node->right->name) {
            node->right = rightRotate(node->right);
            return leftRotate(node);
          }
        }
        return node;
      }
      Node1 *nodeWithMimumValue(Node1 *node) {
        Node1 *current = node;
        while (current->left != NULL)
          current = current->left;
        return current;
      }
      Node1 *deleteNode(Node1 *root, string name) {
        if (root == NULL)
          return root;
        if (name < root->name)
          root->left = deleteNode(root->left, name);
        else if (name > root->name)
          root->right = deleteNode(root->right, name);
        else {
          if ((root->left == NULL) ||
            (root->right == NULL)) {
            Node1 *temp = root->left ? root->left : root->right;
            if (temp == NULL) {
              temp = root;
              root = NULL;
            } else
              *root = *temp;
            free(temp);
          } else {
            Node1 *temp = nodeWithMimumValue(root->right);
            root->name = temp->name;
            root->right = deleteNode(root->right,temp->name);
          }
        }
        if (root == NULL)
          return root;
        root->height = 1 + max(height(root->left),height(root-
>right));
        int balanceFactor = getBalanceFactor(root);
        if (balanceFactor > 1) {
          if (getBalanceFactor(root->left) >= 0) {
            return rightRotate(root);
          } else {
            root->left = leftRotate(root->left);
            return rightRotate(root);
          }
        }
        if (balanceFactor < -1) {
          if (getBalanceFactor(root->right) <= 0) {
            return leftRotate(root);
          } else {
            root->right = rightRotate(root->right);
            return leftRotate(root);
```

```
            }
          }
          return root;
        }
        void inc(Node1* root,string name,int n)
        {
            Node1 *current, *pre;
            if (root == NULL)
                return;
            current = root;
            while (current != NULL) {
                if (current->left == NULL) {
                    if(current->name==name)
                    {
                       current->num=current->num+n;
                    }
                    current = current->right;
                }
                else {
                    pre = current->left;
                    while (pre->right != NULL && pre->right !=
current)
                        pre = pre->right;
                    if (pre->right == NULL) {
                        pre->right = current;
                        current = current->left;
                    }
                    else {
                        pre->right = NULL;
                        if(current->name==name)
                        {
                           current->num=current->num+n;
                        }
                        current = current->right;
                    }
                }
            }
        }
        void dec(Node1* root,string name,int n)
        {
            Node1 *current, *pre;
            if (root == NULL)
                return;
            current = root;
            while (current != NULL) {
                if (current->left == NULL) {
                    if(current->name==name && current->num>=n)
                    {
                       current->num=current->num-n;
                    }
                    current = current->right;
                }
                else {
                    pre = current->left;
                    while (pre->right != NULL && pre->right !=
current)
```

```
                    pre = pre->right;
                if (pre->right == NULL) {
                    pre->right = current;
                    current = current->left;
                }
                else {
                    pre->right = NULL;
                    if(current->name==name && current->num>=n)
                    {
                       current->num=current->num-n;
                    }
                    current = current->right;
                }
            }
        }
        void printTree(Node1 *root)
        {
            if (root != NULL) {
                printTree(root->left);
                cout <<"\nName of company: "<<root->name<<"\nNumber
of stocks: "<<root->num<<"\nPrice of each stock: "<<root->price<<
endl;
                printTree(root->right);
            }
        }
        void printStock(Node1 *root,string name)
        {
            if (root != NULL and root->name==name) {
                printStock(root->left,name);
                cout <<"Name of company: "<<root->name<<"\nNumber of
stocks: "<<root->num<<"\nPrice of each stock: "<<root->price<< endl;
                printStock(root->right,name);
            }
        }
};
int main() {
  Admin A;
  Node1 *root = NULL;
  int a,b,c,choice;
  float p;
  string s;
  User U;
  Node* head=NULL;
  int x,y;
  int value;
  string name,s1;
  while(choice!=3)
  {
      cout<<"\n\nMAIN MENU\n1. User\n2. Admin\n3. Exit\nEnter your
choice: ";
      cin>>choice;
      if(choice==1)
      {
            while(c != 6)
            {
```

```
                int c;
                cout<<"\n\nUSER MENU\n1. View available stocks\n2.
Buy stocks\n3. Sell stocks\n4. Search for user details\n5. Delete
user\n6. Exit\n";
                cout<<"Enter your choice: ";
                cin>>c;
                if(c==1){
                    A.printTree(root);
                }
                else if(c==2){
                    string new_name;
                    int new_data;
                    cout<<"Enter the name of the company: ";
                    cin>>new_name;
                    cout<<"Enter number of stocks you want to buy:
";
                    cin>>new_data;
                    U.push(&head,new_name,new_data);
                    A.dec(root,new_name,new_data);

                }
                else if(c==3){
                    cout<<"\nEnter user ID: ";
                    cin>>x;
                    cout<<"Enter password: ";
                    cin>>y;
                    string name1;
                    int no1;
                    cout<<"Enter the name of company: ";
                    cin>>name1;
                    cout<<"Number of stocks to be sold: ";
                    cin>>no1;
                    U.sell(&head,x,y,name1,no1);
                    A.inc(root,name1,no1);
                }
                else if(c==4){
                    cout<<"\nEnter user ID: ";
                    cin>>x;
                    U.search(&head,x);
                }
                else if(c==5){
                    cout<<"\nEnter user ID: ";
                    cin>>x;
                    cout<<"Enter password: ";
                    cin>>y;
                    U.deluser(&head,x,y);
                }
                else if(c==6)
                    break;
                else
                    cout<<"Please enter a valid number.";
            }
        }
        else if(choice==2)
        {
            string pass;
```

```
          cout<<"Enter password: ";
          cin>>pass;
          if(pass=="1234")
          {
              while(b!=5)
              {
                  cout<<"\n\nADMIN MENU\n1. Insert stock\n2. Delete
stock\n3. Update stock\n4. View stock details\n5. View stocks
bought\n6. Exit\nEnter your choice: ";
                  cin>>b;
                  if(b==1)
                  {
                      cout<<"\nEnter name of the company: ";
                      cin>>s;
                      cout<<"Enter number of stocks to be released:
";
                      cin>>a;
                      cout<<"Enter price of each stock: ";
                      cin>>p;
                      root=A.insertNode(root,a,p,s);
                      cout<<"\nStock added successfully!";
                  }
                  else if(b==2)
                  {
                      cout<<"\nEnter the name of the company: ";
                      cin>>s;
                      root=A.deleteNode(root,s);
                      cout<<"\nStock deleted successfully!";
                  }
                  else if(b==3)
                  {
                      int y,d;
                      cout<<"\nEnter the name of the company: ";
                      cin>>s;
                      cout<<"\n1. Increase stock\n2. Decrease
stock\nEnter your choice: ";
                      cin>>y;
                      if(y==1)
                      {
                        cout<<"\nEnter the number of stocks: ";
                        cin>>d;
                        A.inc(root,s,d);
                        cout<<"\nStock updated successfully!";
                      }
                      else if(y==2)
                      {
                        cout<<"\nEnter the number of stocks: ";
                        cin>>d;
                        A.dec(root,s,d);
                        cout<<"\nStock updated successfully!";
                      }
                  }
                  else if(b==4)
                  {
                      cout<<"\nEnter the name of the company: ";
                      cin>>s;
```

```
                    cout<<"\nSTOCK DETAILS"<<endl;
                    A.printStock(root,s);
                }
                else if(b==5)
                {
                    U.disp(head);
                }
                else if(b==6)
                  break;
                else if(b!=6)
                  cout<<"\nEnter a valid number";
            }
          }
          else
              cout<<"Incorrect password";
        }
      else if(choice==3)
      {
          break;
      }
      else
      {
          cout<<"Enter a valid number.\n\n";
      }
  }
}
```

SAMPLE INPUT/OUTPUT:

Main Menu:

```
MAIN MENU
1. User
2. Admin
3. Exit
Enter your choice:
```

Admin Menu:

```
MAIN MENU
1. User
2. Admin
3. Exit
Enter your choice: 2
Enter password: 1234


ADMIN MENU
1. Insert stock
2. Delete stock
3. Update stock
4. View stock details
5. View stocks bought
6. Exit
Enter your choice:
```

Inserting a stock:

```
ADMIN MENU
1. Insert stock
2. Delete stock
3. Update stock
4. View stock details
5. View stocks bought
6. Exit
Enter your choice: 1

Enter name of the company: L&T
Enter number of stocks to be released: 55
Enter price of each stock: 1204

Stock added successfully!
```

Deleting stock:

```
ADMIN MENU
1. Insert stock
2. Delete stock
3. Update stock
4. View stock details
5. View stocks bought
6. Exit
Enter your choice: 2

Enter the name of the company: Cognizant

Stock deleted successfully!
```

Updating Stock:

```
ADMIN MENU
1. Insert stock
2. Delete stock
3. Update stock
4. View stock details
5. View stocks bought
6. Exit
Enter your choice: 3

Enter the name of the company: L&T

1. Increase stock
2. Decrease stock
Enter your choice: 1

Enter the number of stocks: 5

Stock updated successfully!

ADMIN MENU
1. Insert stock
2. Delete stock
3. Update stock
4. View stock details
5. View stocks bought
6. Exit
Enter your choice: 3

Enter the name of the company: L&T

1. Increase stock
2. Decrease stock
Enter your choice: 2

Enter the number of stocks: 2

Stock updated successfully!
```

View stock details:

```
ADMIN MENU
1. Insert stock
2. Delete stock
3. Update stock
4. View stock details
5. View stocks bought
6. Exit
Enter your choice: 4

Enter the name of the company: L&T

STOCK DETAILS
Name of company: L&T
Number of stocks: 58
Price of each stock: 1204
```

View stocks bought:

```
ADMIN MENU
1. Insert stock
2. Delete stock
3. Update stock
4. View stock details
5. View stocks bought
6. Exit
Enter your choice: 5

User ID: 9383
Name of the company: L&T
Number of stocks: 21
```

Exit to return to main menu:

```
ADMIN MENU
1. Insert stock
2. Delete stock
3. Update stock
4. View stock details
5. View stocks bought
6. Exit
Enter your choice: 6


MAIN MENU
1. User
2. Admin
3. Exit
Enter your choice:
```

User Menu:

```
USER MENU
1. View available stocks
2. Buy stocks
3. Sell stocks
4. Search for user details
5. Delete user
6. Exit
Enter your choice:
```

Viewing available stocks in the User Menu:

```
USER MENU
1. View available stocks
2. Buy stocks
3. Sell stocks
4. Search for user details
5. Delete user
6. Exit
Enter your choice: 1

Name of company: L&T
Number of stocks: 58
Price of each stock: 1204
```

Buying stocks(New User):

```
USER MENU
1. View available stocks
2. Buy stocks
3. Sell stocks
4. Search for user details
5. Delete user
6. Exit
Enter your choice: 2
Enter the name of the company: L&T
Enter number of stocks you want to buy: 24

1. New user
2. Existing user
Enter your choice: 1
Your user ID is 9383 and password is 886

Stock(s) bought successfully!
```

Buying stocks(Existing User):

```
USER MENU
1. View available stocks
2. Buy stocks
3. Sell stocks
4. Search for user details
5. Delete user
6. Exit
Enter your choice: 2
Enter the name of the company: L&T
Enter number of stocks you want to buy: 2

1. New user
2. Existing user
Enter your choice: 2
Enter user ID:9383
Enter password: 886

Stock(s) bought successfully!
```

Sell stocks:

```
USER MENU
1. View available stocks
2. Buy stocks
3. Sell stocks
4. Search for user details
5. Delete user
6. Exit
Enter your choice: 3

Enter user ID: 9383
Enter password: 886
Enter the name of company: L&T
Number of stocks to be sold: 5
```

Searching for user details:

```
USER MENU
1. View available stocks
2. Buy stocks
3. Sell stocks
4. Search for user details
5. Delete user
6. Exit
Enter your choice: 4

Enter user ID: 9383

User ID: 9383
Name of company: L&T
Stocks bought: 21
Total number of stocks bought by the user: 26
Total number of stocks sold by user: 5
```

Delete a user:

```
USER MENU
1. View available stocks
2. Buy stocks
3. Sell stocks
4. Search for user details
5. Delete user
6. Exit
Enter your choice: 5

Enter user ID: 9383
Enter password: 886

User deleted succesfully!
```

Exit to return to Main Menu:

```
USER MENU
1. View available stocks
2. Buy stocks
3. Sell stocks
4. Search for user details
5. Delete user
6. Exit
Enter your choice: 6


MAIN MENU
1. User
2. Admin
3. Exit
Enter your choice:
```

Exit Main Menu to terminate the program:

```
MAIN MENU
1. User
2. Admin
3. Exit
Enter your choice: 3


...Program finished with exit code 0
Press ENTER to exit console.
```

JUSTIFICATION FOR THE DATA STURCTURES CHOSEN:

ADMIN SIDE:

For the admin side, AVL TREE data structure has been used to implement the storing and keeping track of various companies with stocks, their price and the number of stocks.

Why AVL trees?

- AVL trees are most used for implementations with a lot of frequent insert and delete operations. This hold true for our project as well, as the only operations on the stock details is adding more stocks/ deleting them. These operations are more frequent when compared to searching of stocks.
- AVL trees are height balanced and the height never goes beyond Log(N) where N is the number of nodes, in this case the number of different types of stocks.
- AVL trees have self-balancing capabilities.
- AVL trees give better search when compared to binary search trees.

Time and space complexity of processes of AVL trees:

AVL tree	Space	O(n)	O(n)
	Insert	O(log n)	O(log n)
	Search	O(log n)	O(log n)
	Delete	O(log n)	O(log n)

USER SIDE:

For the user side, doubly linked list data structure has been implemented to keep track of various users buying/ selling stocks including the details of the user and the stocks as well.

Why DLL?

In our project, traversal of the user details including the details of the stocks that has been purchased / sold from both ends(recent to oldest and oldest to recent) is very common and hence DLLs have been chosen.

- The singly linked list allows for direct access from a list node only to the next node in the list. A doubly linked list allows convenient access from a list node to the next node and also to the preceding node on the list
- Reversing the doubly linked list is very easy.
- It can allocate or reallocate memory easily during its execution. This helps to update stock count and when a user wants to sell all their stocks/ delete the user account on the whole, memory reallocation will not be a problem.
- The traversal of this doubly linked list is bidirectional which is not possible in a singly linked list. Traversal from both the start and end is made possible hence the recently

bought as well as the old stocks can be accessed. It is possible to find out the order of the users in which they purchased stocks.

- Deletion of nodes is easy as compared to a Singly Linked List. A singly linked list deletion requires a pointer to the node and previous node to be deleted but in the doubly linked list, it only required the pointer which is to be deleted.

Time Complexity for Linked Operations

Time Complexity for the Linked Data Structure		
Operation	**Cost**	
	Singly Linked	**Doubly Linked**
read (anywhere in the linked list)	O(n)	O(n)
add/remove (at the head)	O(1)	O(1)
add/remove (at the tail)	O(n)	O(1)
add/remove (in the interior of the structure)	O(n)	O(n)
resize	N/A	N/A
find by position	O(n)	O(n)
find by target	O(n)	O(n)

GAP ANALYSIS:

User interface

GUI or front end can be done to make it more accessible for the users.

Extending the app

The project can be extended to incorporate fluctuating rates of the stock market by linking the project to the database available online about.

User side payment

Paytm/GPAY/Net banking can be linked for easy transaction for buying/selling stocks from the user side.

Data structure implemented- User side

Doubly linked lists have been used to store user details. Compared to this we realise that using a database management system for the user side will be more efficient.

Data structure implemented – Admin side

AVL trees have been used to store stock and company details. B+ trees can be implemented to organize stocks based on their rates in a better way.

B+ trees are m way search trees whereas AVL trees are binary search trees.

Why B+ Tree is better than AVL Tree?

- An AVL tree is a self-balancing binary search tree, balanced to maintain O(log n) height. A B+ tree is a balanced tree, but it is not a binary tree. Nodes have more children, which increases per-node search time but decreases the number of nodes the search needs to visit. This makes them good for disk-based trees.

- The principal advantage of B+ trees over B trees is they allow you to pack in more pointers to other nodes by removing pointers to data, thus increasing the fanout and potentially decreasing the depth of the tree.