3. Abstract Data Types

Stack

- Insert
- Delete

```
class stack{
    int[] sta;
    int basepointer=0;
    int toppointer=-1;//初始化为-1 代表stack为空的情况
    int item;
    int size;//最大容量
    public stack(int si){
        this.sta= new int[si];
        this.size=si;
    }
    public int pop(){
        if(toppointer==-1){//也可以写成 if(toppointer==basepointer-1)
            System.out.println("stack is empty, cannot pop");
            return -1;
        }
        else{
            item=sta[toppointer];
            toppointer--;
            return item;
        }
    }
    public void push(int item){//添加item进入stack
        if(toppointer==size-1){
            System.out.println("stack is full, cannot push");
        }
        else{
            sta[++toppointer]=item;
            //也可以写成: toppointer=toppointer+1;
            //             stack[toppointer]=item;
        }
    }
    public boolean isEmpty(){//大概率不会考
        if(toppointer==-1){
            return true;
        }
        else{
            return false;
        }
        //return (toppointer==-1)? true: false;
    }
    public int peek(){
        if(this.isEmpty()){
            System.out.println("stack is empty");
            return -1;
        }
        else{
            return sta[toppointer];
        }
    }
}
```

Queue

- Insert
- Delete

```
class queue{
    int[] que= new int[10];
    int front=0;
    int rear=-1;
    int queueFull=10;//store the maximum size of the queue
    int queueLength=0;// store the current size of the queue
    public void enqueue(int item){
        if(queueLength<queueFull){
            if(rear<queueFull-1){
                rear++;
            }
            else{
                rear=0;
            }
            queueLength=queueLength+1;
            que[rear]=item;
        }
        else{
            System.out.println("queue is full, cannot enqueue");
        }
    }
    public int dequeue(){
        if(queueLength==0){
            System.out.println("Queue is empty, cannot dequeue");
            return -1;
        }
        else{
            int item=que[front];
            if(front==queueFull-1){
                front=0;
            }
            else{
                front=front+1;
            }
            queueLength--;
            return item;
        }
    }
}
```

Linked List

- Insert
- Delete
- Find

Initialization

```
CONSTANT nullPointer=-1
TYPE node
        DECLARE data:STRING
        DECLARE nextPointer:INTEGER
ENDTYPE
DECLARE startPointer,freePointer:INTEGER
DECLARE list : ARRAY[0:N] OF node
PROCEDURE Initialization
        startPointer <- -1
        freePointer <- 0
        FOR i <- 0 TO N-1
                list[i].nextPointer<-i+1
        NEXT i
        list[N].nextPointer<-nullPointer
ENDPROCEDURE
```

Insert

```
static void insert(String value) {//根据大小插入 这里对比的是String的字典序
    if (freepointer != nullPointer) {
        int newNodePtr = freepointer;
        list[newNodePtr].data = value;
        freepointer = list[freepointer].pointer;
        int previousPtr = nullPointer;
        int thisPtr = startpointer;
        while((thisPtr != nullPointer) && (value.compareTo(list[thisPtr].data) > 0)){//如果value比thisptr的data大
            previousPtr = thisPtr;
            thisPtr = list[thisPtr].pointer;
        }
        //最终 value会比previousptr的大，比thisptr的小
        if (previousPtr == nullPointer) { // insert at the start of the node
            list[newNodePtr].pointer = startpointer;
            startpointer = newNodePtr;
        } else {
            list[newNodePtr].pointer = list[previousPtr].pointer;
            list[previousPtr].pointer = newNodePtr;
        }
    } else {
        System.out.println("no space");
    }
}
static void insertAtend(String value){
    if(freepointer!=nullPointer){
        int newNodeptr=freepointer;
        list[newNodeptr].data=value;
        freepointer=list[freepointer].pointer;
        int prevPointer=nullPointer;
        int thisptr=startpointer;
        while(thisptr!=nullPointer){
            prevPointer=thisptr;
            thisptr=list[thisptr].pointer;
        }
        if(prevPointer==nullPointer){
            startpointer=newNodeptr;
        }
        else{
            list[prevPointer].pointer=newNodeptr;
        }
        list[newNodeptr].pointer=nullPointer;
    }
}
```

```
PROCEDURE insert(item:STRING)
        DECLARE newPointer,prevPointer,thisPointer:INTEGER
        IF freePointer<>nullPointer
                THEN
                        newPointer<-freePointer
                        freePointer<-list[freePointer].nextPointer
                        list[newPointer].data<-item
                        prevPointer<-nullPointer
                        thisPointer<-startPointer
                        WHILE thisPointer<>nullPointer
                                prevPointer<-thisPointer
                                thisPointer<-
list[thisPointer].nextPointer
                        ENDWHILE
                        IF prevPointer=nullPointer
                                THEN
                                        startPointer<-newPointer
                                ELSE
                                        list[prevPointer].nextPointer<-
newPointer
                        ENDIF
                        list[newPointer].nextPointer<-nullPointer
        ENDIF
ENDPROCEDURE
```

Delete

```
static void deleteNode(String data){
    int thisPtr = startpointer;
    int previousPtr = nullPointer;
    while ((thisPtr != nullPointer) &&
(!data.equalsIgnoreCase(list[thisPtr].data))) {
        previousPtr = thisPtr;
        thisPtr = list[thisPtr].pointer;
    }
    if (thisPtr == nullPointer)
        System.out.println("data is not present");
    else {
        if (thisPtr == startpointer)//删除startPointer存的元素
            startpointer = list[startpointer].pointer;//把startPointer设置为下一个linked的元素
        else
            list[previousPtr].pointer = list[thisPtr].pointer;//若是删除链表中的元素，把previousPtr的元素link的要删除元素的下一个元素上
        list[thisPtr].pointer = freepointer;//把删除的元素设置为free
        freepointer = thisPtr;
    }
}
```

```
PROCEDURE delete(item: STRING)
        DECLARE prevPointer, thisPointer:INTEGER
        prevPointer<-nullPointer
        thisPointer<-startPointer
        WHILE list[thisPointer]<>null && list[thisPointer].data<>item
                prevPointer<-thisPointer
                thisPointer<-list[thisPointer].nextPointer
        ENDWHILE
        IF thisPointer<>nullPointer
                THEN
                        IF thisPointer=startPointer
                                THEN
                                        startPointer<-list[startPointer].nextPointer
                                ELSE
                                        list[prevPointer].nextPointer<-list[thisPointer].nextPointer
                        ENDIF
                        list[thisPointer].nextPointer<-freePointer
                        freePointer<-thisPointer
                ELSE
                        OUTPUT "not found the item to be deleted"
        ENDIF
ENDPROCEDURE
```

Find

```
static int findNode(String data) {
    int currentNode = startpointer;
    while (currentNode != nullPointer) {
        if (data.equalsIgnoreCase(list[currentNode].data)) {
            return currentNode;
        }
        currentNode = list[currentNode].pointer;
    }
    return currentNode;
}
```

完整代码

```
public class Main {
    static class ListNode {
        String data;
        int pointer;
        public ListNode() {
            data = "";
            pointer = nullPointer;
        }
    }
    static final int nullPointer = -1;
    static int startpointer, freepointer;
    static Scanner input = new Scanner(System.in);
    static ListNode[] list = new ListNode[8];
    static void initializeList() {
        startpointer = nullPointer;
        freepointer = 0;
        for (int i = 0; i < 7; i++) {
            list[i] = new ListNode(); // set all nodes as free nodes;
            list[i].pointer = i + 1;
        }
        list[7] = new ListNode();
        list[7].pointer = -1; // last node is intilized as -1;
    }
    static int menu() {
        System.out.println("1. insert an item");
        System.out.println("2. delete an item");
        System.out.println("3. search an item");
        System.out.println("4. display an item");
        System.out.println("Enter your choice 1/2/3/4");
        int ch = input.nextInt();
        return ch;
    }
    static void insert(String value) {//根据大小插入 这里对比的是String的
字典序
        if (freepointer != nullPointer) {
            int newNodePtr = freepointer;
            list[newNodePtr].data = value;
            freepointer = list[freepointer].pointer;
            int previousPtr = nullPointer;
            int thisPtr = startpointer;
            while((thisPtr != nullPointer) &&
(value.compareTo(list[thisPtr].data) > 0)){//如果value比thisptr的data大
                previousPtr = thisPtr;
                thisPtr = list[thisPtr].pointer;
            }
            //最终 value会比previousptr的大，比thisptr的小
            if (previousPtr == nullPointer) { // insert at the start of
the node
                list[newNodePtr].pointer = startpointer;
                startpointer = newNodePtr;
            } else {
                list[newNodePtr].pointer = list[previousPtr].pointer;
                list[previousPtr].pointer = newNodePtr;
            }
        } else {
            System.out.println("no space");
        }
    }
    static void insertAtend(String value){
        if(freepointer!=nullPointer){
            int newNodeptr=freepointer;
            list[newNodeptr].data=value;
            freepointer=list[freepointer].pointer;
            int prevPointer=nullPointer;
            int thisptr=startpointer;
            while(thisptr!=nullPointer){
                prevPointer=thisptr;
                thisptr=list[thisptr].pointer;
            }
            if(prevPointer==nullPointer){
                startpointer=newNodeptr;
            }
            else{
                list[prevPointer].pointer=newNodeptr;
            }
            list[newNodeptr].pointer=nullPointer;
        }
    }
    static void display(){
        int currentNode = startpointer;
        if (startpointer == nullPointer)
            System.out.println("no data in list");
        while (currentNode != nullPointer) {
            System.out.print(list[currentNode].data + "  ");
            currentNode = list[currentNode].pointer;
        }
    }
    static int findNode(String data) {
        int currentNode = startpointer;
        while (currentNode != nullPointer) {
            if (data.equalsIgnoreCase(list[currentNode].data)) {
                return currentNode;
            }
            currentNode = list[currentNode].pointer;
        }
        return currentNode;
    }
    static void deleteNode(String data){
        int thisPtr = startpointer;
        int previousPtr = nullPointer;
        while ((thisPtr != nullPointer) &&
(!data.equalsIgnoreCase(list[thisPtr].data))) {
            previousPtr = thisPtr;
            thisPtr = list[thisPtr].pointer;
        }
        if (thisPtr == nullPointer)
            System.out.println("data is not present");
        else {
            if (thisPtr == startpointer)//删除startPointer存的元素
                startpointer = list[startpointer].pointer;//把
startPointer设置为下一个linked的元素
            else
                list[previousPtr].pointer = list[thisPtr].pointer;//若是
删除链表中的元素，把previousPtr的元素link的要删除元素的下一个元素上
        }
        list[thisPtr].pointer = freepointer;//把删除的元素设置为free
        freepointer = thisPtr;
    }
    public static void main(String[] args) {
        initializeList();
        String Data, Choice = "";
        do {
            System.out.println("enter your choice");
            int ch = menu();
            switch (ch) {
                case 1:
                    System.out.println("Enter the value");
                    Data = input.next();
                    insert(Data);
                    display();
                    break;
                case 2:
                    System.out.println("Enter the value");
                    Data = input.next();
                    deleteNode(Data);
                    display();
                    break;
                case 3:
                    System.out.println("Enter the value");
                    Data = input.next();
                    int find = findNode(Data);
                    System.out.println(" item found " + find);
                    if (find == nullPointer)
                        System.out.println("data not found");
                    display();
                    break;
                case 4:
                    System.out.println("Print complete list");
                    display();
            }
            System.out.println("do you want to continue enter Y /y ");
            Choice = input.next();
        } while (Choice.equalsIgnoreCase("Y"));
    }
}
```

Binary tree

State the purpose of the free pointer:

- To point to the start/ first of the empty node/nodes

Initialization

```
TYPE node
        DECLARE leftPointer: INTEGER
        DECLARE rightPointer : INTEGER
        DECLARE data : STRING
ENDTYPE
DECLARE N : INTEGER
DECLARE freePointer,rootPointer, nullPointer : INTEGER
DECLARE tree : ARRAY [0:N] OF node
PROCEDURE initialization
        freePointer <- 0
        nullPointer <- -1
        rootPointer <- nullPointer
        FOR i <- 0 TO N-1
                tree[i].leftPointer <- i+1
        NEXT i
        tree[N].leftPointer <- -1
ENDPROCEDURE
```

Insert

```
public static void AddNode(){
    Scanner sca = new Scanner(System.in);
    System.out.println("input the data to be Added");
    int data = sca.nextInt();
    if(FreeNode<=19){//如果Binary tree没用满
        int newPtr=FreeNode;
        ArrayNodes[newPtr]=new node(-1,data,-1);
        if(RootPointer==-1){//如果当前为空
            RootPointer=newPtr;
        }
        else{
            Boolean placed=false;
            int CurrentNode=RootPointer;
            while(placed==false){
                if(data<ArrayNodes[CurrentNode].data){//如果插入数值比当前数值小，则看左边
                    if(ArrayNodes[CurrentNode].leftPointer==-1){//如果左节点为空 则放到左节点
```

```
                        ArrayNodes[CurrentNode].leftPointer=newPtr;
                        placed=true;
                    }
                    else{//如果不为空 则继续往下搜空的地方插入
                        CurrentNode=ArrayNodes[CurrentNode].leftPointer;
                    }
                }
                else{//如果插入数值比当前数值大，则看右边
                    if(ArrayNodes[CurrentNode].rightPointer==-1){
                        ArrayNodes[CurrentNode].rightPointer=newPtr;
                        placed=true;
                    }
                    else{

CurrentNode=ArrayNodes[CurrentNode].rightPointer;
                    }
                }
            }
        }
        FreeNode=FreeNode+1;
    }
    else{
        System.out.println("Tree is full");
    }
}
```

```
PROCEDURE insert
        DECLARE newPointer,thisPointer,prevPointer:INTEGER
        DECLARE turnedLeft : BOOLEAN
        IF freePointer<>-1
                THEN
                        newPointer<-freePointer
                        freePointer<-tree[freePointer].leftPointer
                        INPUT data
                        tree[newPointer].data <- data
                        tree[newPointer].leftPointer<- nullPointer
                        tree[newPointer].rightPointer <- nullPointer
                        IF rootPointer = nullPointer
                                THEN
                                        rootPointer<- newPointer
                                ELSE
                                        thisPointer<-rootPointer
                                        prevPointer<-nullPointer
                                        WHILE thisPointer<>-1
                                                prevPointer<-thisPointer
                                                IF
tree[thisPointer].data < data
                                                        THEN

turnedLeft<-FALSE

thisPointer<-tree[thisPointer].rightPointer
                                                        ELSE

turnedLeft<-TRUE

thisPointer<-tree[thisPointer].leftPointer
                                                ENDIF
                                        ENDWHILE
                                        IF turnedLeft
                                                THEN

tree[prevPointer].leftPointer<-newPointer
                                                ELSE

tree[prevPointer].rightPointer<-newPointer
                                        ENDIF
                        ENDIF
        ENDIF
ENDPROCEDURE
```

Search 🔍

```
public static void search(int element){
    int nownode=RootPointer;
    while(nownode!=-1){
        if(ArrayNodes[nownode].data==element){
            System.out.println("found, the position is:"+ nownode);
            return;
        }
        else if(ArrayNodes[nownode].data>element){
            nownode=ArrayNodes[nownode].leftPointer;
        }
        else{
            nownode=ArrayNodes[nownode].rightPointer;
        }
    }
    System.out.println("not found");
}
```

```
FUNCTION search(item : STRING) RETURNS INTEGER
        DECLARE thisPointer : INTEGER
        thisPointer<-rootPointer
        WHILE thisPointer<>nullPointer
                IF tree[thisPointer].data=item
                        THEN
                                RETURN thisPointer
                        ELSE
                                IF tree[thisPointer].data>item
                                        THEN
                                                thisPointer<-
tree[thisPointer].leftPointer
                                        ELSE
                                                thisPointer<-
tree[thisPointer].rightPointer
                                ENDIF
                ENDIF
        ENDWHILE
ENDFUNCTION
```

Traversal

```
public static void inorderTraverse(int ptr){
    if(ptr==-1) return;
    inorderTraverse(ArrayNodes[ptr].leftPointer);
    System.out.println(ArrayNodes[ptr].data);
    inorderTraverse(ArrayNodes[ptr].rightPointer);
}
```

完整代码

```
class Main{
    static node[] ArrayNodes = new node[20];
    static int RootPointer=-1;
    static int FreeNode=0;
    public static void main(String[] args){
        //Adding: 23 5 8 100 9 88
        System.out.println("Adding Nodes");
        AddNode();
        AddNode();
        AddNode();
        AddNode();
        AddNode();
        AddNode();
        System.out.println("Printing Tree Content");
        printContent();
        System.out.println("Inorder Traverse");
        inorderTraverse(0);
        search(5);
    }
    public static void AddNode(){
        Scanner sca = new Scanner(System.in);
        System.out.println("input the data to be Added");
        int data = sca.nextInt();
        if(FreeNode<=19){//如果
            int newPtr=FreeNode;
            ArrayNodes[newPtr]=new node(-1,data,-1);
            if(RootPointer==-1){//如果当前为空
                RootPointer=newPtr;
            }
            else{
                Boolean placed=false;
                int CurrentNode=RootPointer;
                while(placed==false){
                    if(dataelement){
                nownode=ArrayNodes[nownode].leftPointer;
            }
            else{
                nownode=ArrayNodes[nownode].rightPointer;
            }
        }
        System.out.println("not found");
    }
    static class node{
        int leftPointer;
        int data;
        int rightPointer;
        public node(int lp,int da, int rp){
            this.leftPointer=lp;
            this.rightPointer=rp;
            this.data=da;
```

```
                }
            }
        }
```