Institute Of Technology, Nirma university

BRANCH :- **Computer Science Engineering**

PRACTICAL SUBMISSION

|*|*STUDENT INFO*|*|

Name :- **Pratik Kansara**

Roll No. :- **20BCE510**

Division :- **E4**

|*|*SUBJECT INFO*|*|

Subject :- **Advanced Data Structures**

Practical No. :- **7**

Practical – 7

AIM :- Implement Heap using link list so that every time element with high priority should be fetched from heap.

Code Using Heap Trees:

```
#include<bits/stdc++.h>

#include <queue>

using namespace std;


class HeapNode{

    public:

        int data;

        HeapNode* left;

        HeapNode* right;

        HeapNode* parent;


        HeapNode(int val) {

            data = val;

            left = NULL;

            right = NULL;

            parent = NULL;

        }

};



class Heap {

    HeapNode* root;

    public:

        Heap() {

            root = NULL;
```

```
}

HeapNode* lastparent() {
    queue<HeapNode*> q;
    q.push(root);
    HeapNode* temp;

    while(!q.empty()) {
        temp = q.front();
        q.pop();

        if (temp->left && temp->right) {
            q.push(temp->left);
            q.push(temp->right);
        } else {
            break;
        }
    }

    return temp;
}

void upHeapify(HeapNode *temp) {
    if (temp->parent == NULL) {
        return;
    }
    if(temp->parent->data < temp->data) {
        swap(temp->parent->data, temp->data);
        upHeapify(temp->parent);
    }
}
```

```
void add(int data) {
    HeapNode* newnode = new HeapNode(data);
    cout << "Inserting : " << data<<"\n";

    if (root == NULL) {
        root = newnode;
        return;
    }

    HeapNode *lpar = lastparent();

    if(lpar->left == NULL) {
        lpar->left = newnode;
        newnode->parent = lpar;
    } else {
        lpar->right = newnode;
        newnode->parent = lpar;
    }

    upHeapify(newnode);
}


bool isEmpty() {
    if (root == NULL) {
        return true;
    } else {
        return false;
    }
}
```

```
HeapNode* lastNode() {
    queue<HeapNode*> q;
    q.push(root);
    HeapNode* last;

    while(!q.empty()) {
        last = q.front();
        q.pop();
        if (last->left) {
            q.push(last->left);
        }
        if (last->right) {
            q.push(last->right);
        }
    }
    return last;
}



void downHeapify(HeapNode* he) {
    HeapNode* largest = he;

    if (he->left && he->left->data > largest->data) {
        largest = he->left;
    }
    if (he->right && he->right->data > largest->data) {
        largest = he->right;
    }

    if (largest != he) {
```

```
            swap(largest->data, he->data);
            downHeapify(largest);
        }
    }

    void remove() {
        if (isEmpty()) {
            cout <<"Heap is Empty!!"<<endl;
            return;
        }
        cout << "Deleting : "<<root->data<<endl;

        HeapNode *last = lastNode();
        if (last == root)
        {
            delete last;
            root = NULL;
            return;

        }
        swap(root->data, last->data);
        HeapNode* pr = last->parent;

        if (pr->left == last) {
            delete last;
            pr->left = NULL;
        } else {
            delete last;
            pr->right = NULL;
        }
```

```
            downHeapify(root);

        }

        void inorder(HeapNode* he) {
            if (he != NULL) {
                inorder(he->left);
                cout<<he->data<<" ";
                inorder(he->right);
            }
        }

        void print() {
            cout << "Printing data of the Heap : ";
            HeapNode* temp = root;
            inorder(temp);
            cout<<endl;
        }
};


int main() {
    Heap h;
    while(1) {
        cout << "1. For insert" << endl;
        cout << "2. For Extract Max" << endl;
        cout << "3. For Display" << endl;
        cout << "4. Exit" << endl;
        int choice;
        cin >> choice;
```

```
        switch(choice) {
            case 1:
                int data;
                cout << "Enter Key : ";
                cin >> data;
                h.add(data);
                break;

            case 2:
                h.remove();

                break;

            case 3:
                h.print();
                break;

            case 4:
                exit(0);

        }
    }

}
```

OUTPUT

```
1. For insert
2. For Extract Max
3. For Display
4. Exit
1
Enter Key : 3
Inserting : 3
1. For insert
2. For Extract Max
3. For Display
4. Exit
1
Enter Key : 2
Inserting : 2
1. For insert
2. For Extract Max
3. For Display
4. Exit
1
Enter Key : 1
Inserting : 1
1. For insert
2. For Extract Max
3. For Display
4. Exit
1
Enter Key : 4
Inserting : 4
1. For insert
2. For Extract Max
3. For Display
4. Exit
1
Enter Key : 8
Inserting : 8
1. For insert
2. For Extract Max
3. For Display
4. Exit
1
Enter Key : 5
Inserting : 5
1. For insert
2. For Extract Max
3. For Display
4. Exit
1
Enter Key : 6
Inserting : 6
```

```
1. For insert
2. For Extract Max
3. For Display
4. Exit
1
Enter Key : 7
Inserting : 7
1. For insert
2. For Extract Max
3. For Display
4. Exit
3
Printing data of the Heap : 2 4 7 3 8 1 6 5
1. For insert
2. For Extract Max
3. For Display
4. Exit
2
Deleting : 8
1. For insert
2. For Extract Max
3. For Display
4. Exit
2
Deleting : 7
1. For insert
2. For Extract Max
3. For Display
4. Exit
2
Deleting : 6
1. For insert
2. For Extract Max
3. For Display
4. Exit
2
Deleting : 5
1. For insert
2. For Extract Max
3. For Display
4. Exit
2
Deleting : 4
```

```
1. For insert
2. For Extract Max
3. For Display
4. Exit
2
Deleting : 3
1. For insert
2. For Extract Max
3. For Display
4. Exit
2
Deleting : 2
1. For insert
2. For Extract Max
3. For Display
4. Exit
2
Deleting : 1
1. For insert
2. For Extract Max
3. For Display
4. Exit
2
Heap is Empty!!
```

Code Using Ordered Link list:

HNode.java

```
public class HNode {
    public int key, priority;
    public HNode next;

    public HNode(int key, int priority) {
        this.key = key;
        this.priority = priority;
    }

}
```

Heap.java

```
public class Heap {
    HNode head, tail;
    int type;

    public Heap(int type) {
        this.type = type;
    }

    public void push(int key, int priority) {
        if (head == null) {
            head = new HNode(key, priority);
            tail = head;
            return;
        }

        if (type == 0) {
            if (priority < head.priority) {
                pushtAtBeg(key, priority);
            } else if (priority > tail.priority) {
                pushtAtEnd(key, priority);
            } else {
                pushAtMid(key, priority);
            }
        } else {
            if (priority > head.priority) {
                pushtAtBeg(key, priority);
            } else if (priority < tail.priority) {
                pushtAtEnd(key, priority);
            } else {
                pushAtMid(key, priority);
            }
        }

    }

    private void pushtAtBeg(int key, int priority) {
        HNode temp = new HNode(key, priority);
```

```
        temp.next = head;
        head = temp;
    }

    private void pushtAtEnd(int key, int priority) {
        HNode temp = new HNode(key, priority);
        tail.next = temp;
        tail = temp;
    }

    private void pushAtMid(int key, int priority) {
        HNode curr = head;
        HNode prev = curr;

        if (type == 0) {
            while (curr != null && curr.priority < priority) {
                prev = curr;
                curr = curr.next;
            }

            HNode temp = new HNode(key, priority);
            prev.next = temp;
            temp.next = curr;
        } else {
            while (curr != null && curr.priority > priority) {
                prev = curr;
                curr = curr.next;
            }

            HNode temp = new HNode(key, priority);
            prev.next = temp;
            temp.next = curr;
        }
    }

    public int peek() {
        if (head == null) {
            return -1;
        }
        return head.key;
    }

    public HNode pop() {
        if (head == null) {
            return null;
        }
        if (head == tail) {
            HNode temp = head;
            head = tail = null;
            return temp;
        }
        HNode temp = head;
        head = head.next;

        return temp;
```

```
    }
}
```

HeapRunner.java

```
import java.util.Scanner;

public class HeapRunner {
    public static void main(String[] args) throws Exception {
        Heap h1;
        String heaptype = "";

        Scanner sc = new Scanner(System.in);

        System.out.print("Press 1 for MaxHeap 0 for MinHeap : ");
        int heapchoice = sc.nextInt();

        heaptype = (heapchoice == 1) ? "Maxheap" : "Minheap";
        h1 = new Heap(heapchoice);

        while (true) {
            System.out.println("\n1. For Push Element into " + heaptype + "
heap");
            System.out.println("2. For Pop Element from " + heaptype + " heap");
            System.out.println("3. For Fetch Element from " + heaptype + "
heap");
            System.out.println("4. For Exit");
            int choice = sc.nextInt();

            switch (choice) {
                case 1:
                    System.out.print("Enter Element : ");
                    int ele = sc.nextInt();
                    System.out.print("\nEnter Priority : ");
                    int prio = sc.nextInt();
                    h1.push(ele, prio);
                    break;
                case 2:
                    HNode temp = h1.pop();
                    if (temp == null) {
                        System.out.println("Heap is Empty!!!");
                        System.exit(0);
                    } else {
                        System.out.println(temp.key + " is popped from heap");
                    }
                    break;
                case 3:
                    System.out.println("Element : " + h1.peek());
                    break;
                case 4:
                    System.exit(0);
            }
        }
```

```
    }
}
```

OUTPUT

```
Press 1 for MaxHeap 0 for MinHeap : 1

1. For Push Element into Maxheap heap
2. For Pop Element from Maxheap heap
3. For Fetch Element from Maxheap heap
4. For Exit
1
Enter Element : 1

Enter Priority : 1

1. For Push Element into Maxheap heap
2. For Pop Element from Maxheap heap
3. For Fetch Element from Maxheap heap
4. For Exit
1
Enter Element : 2

Enter Priority : 2

1. For Push Element into Maxheap heap
2. For Pop Element from Maxheap heap
3. For Fetch Element from Maxheap heap
4. For Exit
1
Enter Element : 3

Enter Priority : 3
```

```
1. For Push Element into Maxheap heap
2. For Pop Element from Maxheap heap
3. For Fetch Element from Maxheap heap
4. For Exit
1
Enter Element : 4

Enter Priority : 4

1. For Push Element into Maxheap heap
2. For Pop Element from Maxheap heap
3. For Fetch Element from Maxheap heap
4. For Exit
1
Enter Element : 5

Enter Priority : 5

1. For Push Element into Maxheap heap
2. For Pop Element from Maxheap heap
3. For Fetch Element from Maxheap heap
4. For Exit
2
5 is popped from heap
4. For Exit
2
4 is popped from heap

1. For Push Element into Maxheap heap
2. For Pop Element from Maxheap heap
3. For Fetch Element from Maxheap heap
4. For Exit
2
3 is popped from heap
```

```
1. For Push Element into Maxheap heap
2. For Pop Element from Maxheap heap
3. For Fetch Element from Maxheap heap
4. For Exit
2
2 is popped from heap

1. For Push Element into Maxheap heap
2. For Pop Element from Maxheap heap
3. For Fetch Element from Maxheap heap
4. For Exit
2
1 is popped from heap

1. For Push Element into Maxheap heap
2. For Pop Element from Maxheap heap
3. For Fetch Element from Maxheap heap
4. For Exit
2
Heap is Empty!!!
```