**TASK MANAGEMENT SYSTEM**

**TYPES OF LINKED LISTS:**

**- Singly Linked List:**

Each node contains data and a reference to the next node.

Direction: One-way traversal.

**- Doubly Linked List:**

Each node contains data, a reference to the next, and a reference to the previous node.

Direction: Two-way traversal.

**ADVANTAGES OF LINKED LISTS:**

- Dynamic size (no need to define size ahead).

- Efficient insertions/deletions at any position (especially start).

- Better memory usage when frequently modifying data**.**

**PROGRAM:**

import java.util.\*;

public class ParentTask {

class Task {

int taskid;

String taskname, status;

Task nexttask;

public Task(int taskid, String taskname, String status) {

this.taskid = taskid;

this.taskname = taskname;

this.status = status;

this.nexttask = null;

}

}

Task head;

void addTask(int taskid, String taskname, String status) {

Task newTask = new Task(taskid, taskname, status);

if (head == null) {

head = newTask;

} else {

Task temp = head;

while (temp.nexttask != null) {

temp = temp.nexttask;

}

temp.nexttask = newTask;

}

System.out.println("Task Added..\n");

}

void searchTask(int taskid) {

if (head == null) {

System.out.println("NO TASK FOUND");

} else {

Task temp = head;

boolean found = false;

while (temp != null) {

if (temp.taskid == taskid) {

found = true;

System.out.println("TASK FOUND.\nTASKID = " + temp.taskid + "\nTASKNAME = " + temp.taskname + "\nSTATUS = " + temp.status);

break;

}

temp = temp.nexttask;

}

if (!found) {

System.out.println("TASK NOT FOUND IN LIST.");

}

}

}

void traverseTask() {

Task temp = head;

if (head == null) {

System.out.println("NO TASK FOUND");

} else {

int i = 1;

System.out.println("LIST OF TASKS:");

while (temp != null) {

System.out.println("TASK " + i + "\nTASKID: " + temp.taskid + "\nTASKNAME: " + temp.taskname + "\nSTATUS: " + temp.status);

temp = temp.nexttask;

i++;

}

}

}

void deleteTask(int taskid) {

if (head == null) {

System.out.println("NO TASK FOUND");

return;

}

if (head.taskid == taskid) {

head = head.nexttask;

System.out.println("Task deleted...\n");

return;

}

Task temp = head;

boolean deleted = false;

while (temp.nexttask != null) {

if (temp.nexttask.taskid == taskid) {

temp.nexttask = temp.nexttask.nexttask;

System.out.println("Task deleted...\n");

deleted = true;

break;

}

temp = temp.nexttask;

}

if (!deleted) {

System.out.println("CANNOT DELETE TASK!");

}

}

public static void main(String[] args) {

int ti;

String tn, st;

Scanner sc = new Scanner(System.in);

ParentTask pt = new ParentTask();

System.out.println("Enter\n1. ADD TASK\n2. SEARCH TASK\n3. TRAVERSE TASK\n4. DELETE TASK\n5. QUIT");

while (true) {

System.out.print("Enter choice: ");

int choice = sc.nextInt();

System.out.println();

if (choice == 1) {

System.out.println("Enter taskid, taskname, status:");

ti = sc.nextInt();

tn = sc.next();

st = sc.next();

pt.addTask(ti, tn, st);

} else if (choice == 2) {

System.out.println("Enter taskid to search:");

ti = sc.nextInt();

pt.searchTask(ti);

} else if (choice == 3) {

pt.traverseTask();

} else if (choice == 4) {

System.out.println("Enter taskid to delete:");

ti = sc.nextInt();

pt.deleteTask(ti);

} else{

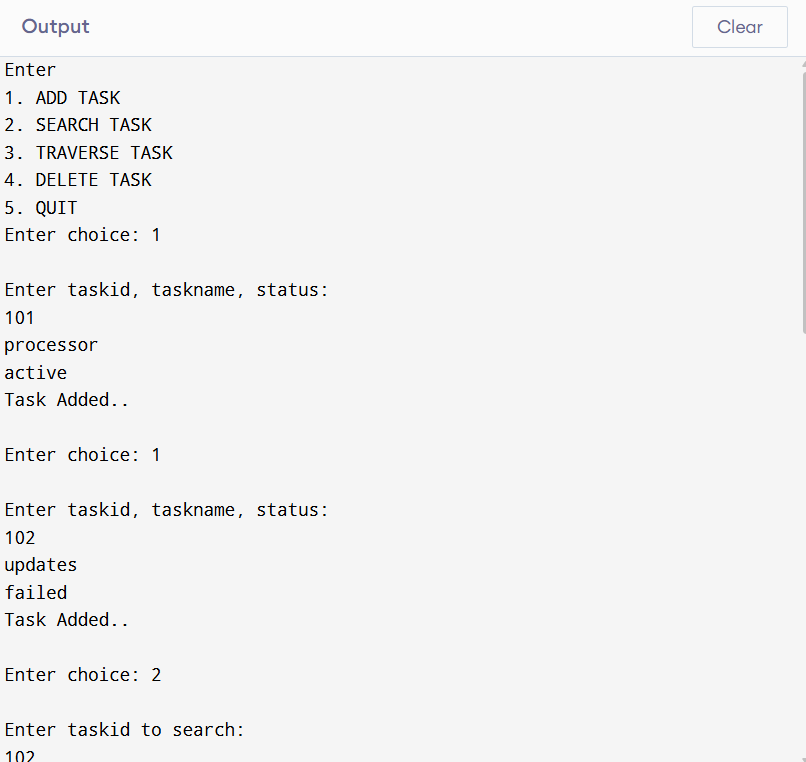
break;

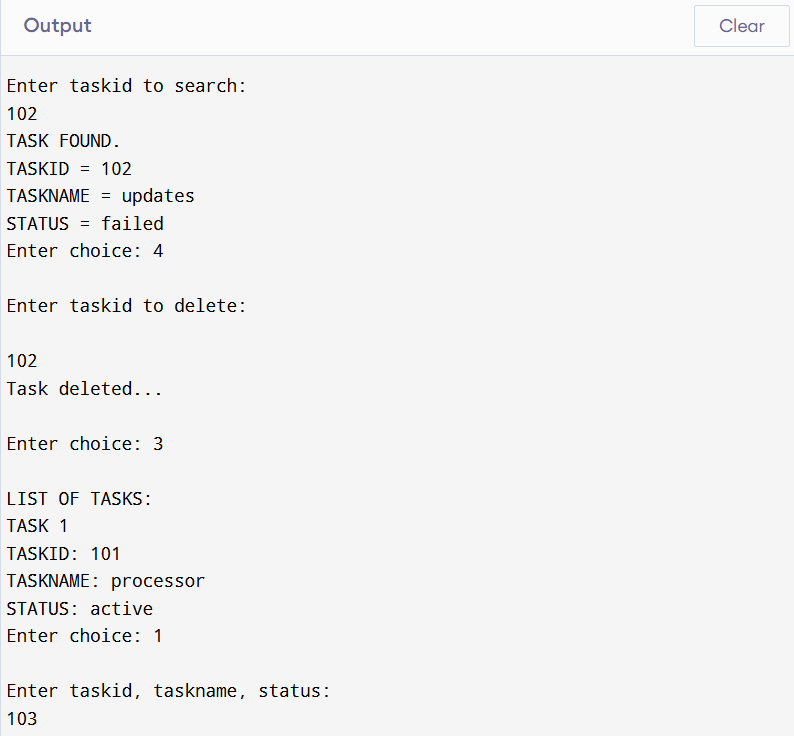
}

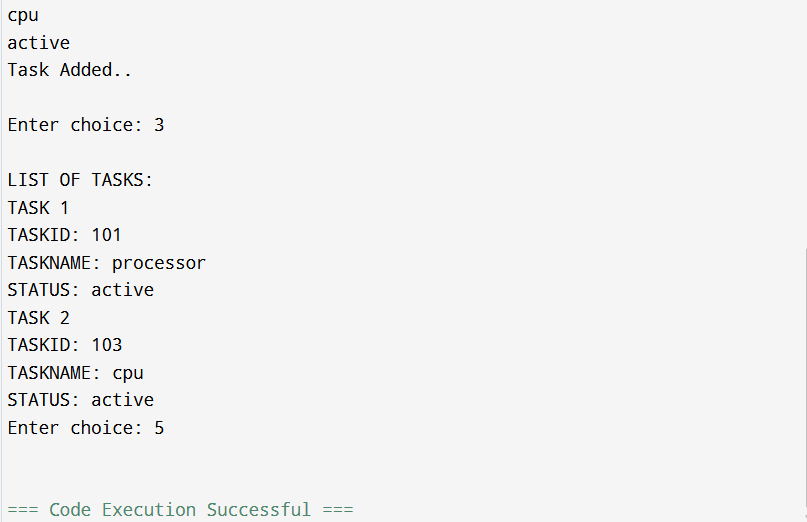
}

}

}







**TIME COMPLEXITY:**

- Add (at end): O(n)

- Search: O(n)

- Traverse: O(n)

- Delete: O(n)

**ADVANTAGES OF LINKED LISTS OVER ARRAYS:**

- Dynamic resizing: Easily add/remove elements without resizing.

- Efficient insertions/deletions at head or middle (no shifting like in arrays).

- Ideal for systems where size frequently changes.

**LIMITATIONS:**

- Slower access (no index-based access like arrays).

- Slightly higher memory usage (due to pointer storage).

- Traversal is linear and cannot go backwards in singly linked lists.