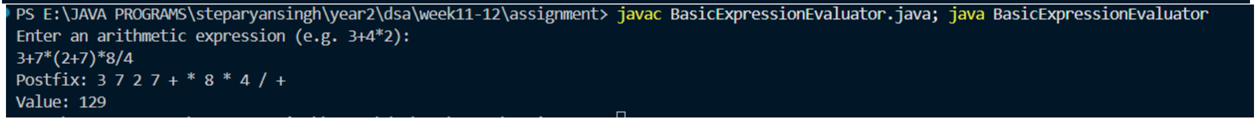
```java
import java.util.Scanner;

class StackString {
    String[] arr = new String[200];
    int top = -1;
    void push(String x) { arr[++top] = x; }
    String pop() { return arr[top--]; }
    String peek() { return arr[top]; }
    boolean empty() { return top == -1; }
}

class StackInt {
    int[] arr = new int[200];
    int top = -1;
    void push(int x) { arr[++top] = x; }
    int pop() { return arr[top--]; }
    boolean empty() { return top == -1; }
}

public class BasicExpressionEvaluator {
    static int prec(char op) {
        if (op == '+' || op == '-') return 1;
        if (op == '*' || op == '/') return 2;
        return 0;
    }

    static String infixToPostfix(String expr) {
        StackString stack = new StackString();
        StringBuilder output = new StringBuilder();
        for (int i = 0; i < expr.length(); i++) {
            char c = expr.charAt(i);
            if (Character.isWhitespace(c)) continue;
            if (Character.isDigit(c)) {
                String num = "";
                while (i < expr.length() &&
Character.isDigit(expr.charAt(i))) {
                    num += expr.charAt(i);
                    i++;
                }
                i--;
                output.append(num).append(" ");
            } else if (c == '(') {
                stack.push("(");
            } else if (c == ')') {
                while (!stack.empty() && !stack.peek().equals("(")) {
                    output.append(stack.pop()).append(" ");
                }
                if (!stack.empty()) stack.pop();
            } else if (c == '+' || c == '-' || c == '*' || c == '/') {
                while (!stack.empty() && prec(stack.peek().charAt(0)) >=
prec(c)) {
                    output.append(stack.pop()).append(" ");
                }
                stack.push(String.valueOf(c));
            }
        }
        while (!stack.empty()) {
            output.append(stack.pop()).append(" ");
        }
        return output.toString().trim();
    }

    static int evaluatePostfix(String postfix) {
        StackInt stack = new StackInt();
        String[] tokens = postfix.split("\\s+");
        for (String token : tokens) {
            if (token.equals("+") || token.equals("-") ||
token.equals("*") || token.equals("/")) {
                int b = stack.pop();
                int a = stack.pop();
                int res = 0;
                if (token.equals("+")) res = a + b;
                else if (token.equals("-")) res = a - b;
                else if (token.equals("*")) res = a * b;
                else if (token.equals("/")) res = a / b;
                stack.push(res);
            } else {
                stack.push(Integer.parseInt(token));
            }
        }
```

```java
        return stack.pop();
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter an arithmetic expression (e.g. 3+4*2):
");
        String expr = sc.nextLine();
        String postfix = infixToPostfix(expr);
        int value = evaluatePostfix(postfix);
        System.out.println("Postfix: " + postfix);
        System.out.println("Value: " + value);
    }
}
```

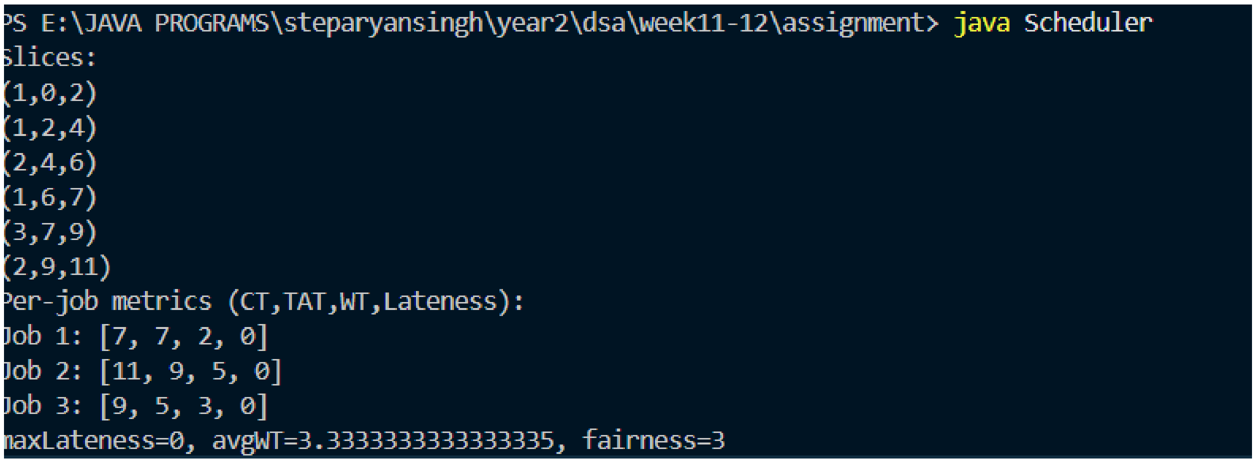


```
PS E:\JAVA PROGRAMS\steparyansingh\year2\dsa\week11-12\assignment> javac BasicExpressionEvaluator.java; java BasicExpressionEvaluator
Enter an arithmetic expression (e.g. 3+4*2):
3+7*(2+7)*8/4
Postfix: 3 7 2 7 + * 8 * 4 / +
Value: 129
```

```java
import java.util.*;

class Slice {
    int id;
    long start, end;
    Slice(int id, long s, long e) { this.id = id; start = s; end = e; }
    public String toString() { return "(" + id + "," + start + "," + end +
")"; }
}
```

```
class SchedResult {
    List<Slice> slices;
    long[][] metrics;
    long maxLateness;
    double avgWT;
    long fairness;
    SchedResult(List<Slice> slices, long[][] metrics, long maxLateness,
double avgWT, long fairness) {
        this.slices = slices; this.metrics = metrics; this.maxLateness =
maxLateness; this.avgWT = avgWT; this.fairness = fairness;
    }
}

public class Scheduler {

    static long enqCounter = 0;
    static long agingSeq = 0;

    static class Job {
        int id;
        long arrival;
        long remaining;
        int basePri;
        long deadline;
        boolean finished = false;
        long completionTime = -1;
        int currentPri;
        long waitingStart;
        long nextAgingTime;
        int promotionsDone;
        long lastEnqId = -1;
    }

    static class QEntry {
        int jobIdx;
        long enqId;
        QEntry(int jobIdx, long enqId) { this.jobIdx = jobIdx; this.enqId
= enqId; }
    }
    static class AgingEntry implements Comparable<AgingEntry> {
        long time;
        int jobIdx;
        long seq;
        AgingEntry(long time, int jobIdx, long seq) { this.time = time;
this.jobIdx = jobIdx; this.seq = seq; }
        public int compareTo(AgingEntry o) {
            if (this.time < o.time) return -1;
            if (this.time > o.time) return 1;
            return Long.compare(this.seq, o.seq);
        }
    }

    public static SchedResult schedule(int[] ids, long[] arrival, long[]
duration, int[] basePri,
                                       long[] deadline, long q, long A,
int K) {

        int n = ids.length;
        Job[] jobs = new Job[n];
        for (int i = 0; i < n; i++) {
            Job j = new Job();
            j.id = ids[i];
            j.arrival = arrival[i];
            j.remaining = duration[i];
            j.basePri = basePri[i];
            j.deadline = deadline[i];
            j.currentPri = basePri[i];
            j.promotionsDone = 0;
            j.waitingStart = -1;
            j.nextAgingTime = Long.MAX_VALUE;
            jobs[i] = j;
        }

        ArrayDeque<QEntry>[] queues = new ArrayDeque[K+1];
        for (int p = 1; p <= K; p++) queues[p] = new ArrayDeque<>();
        PriorityQueue<AgingEntry> agingHeap = new PriorityQueue<>();
        List<Slice> slices = new ArrayList<>();
        int nextArrivalIdx = 0;
        long time = 0;
        class EnqueueHelper {
            void enqueueWaiting(int jobIdx, long now) {
                Job J = jobs[jobIdx];
                J.waitingStart = now;
                if (A > 0 && J.currentPri < K) {
                    J.nextAgingTime = J.waitingStart + A *
(J.promotionsDone + 1);
                    agingHeap.add(new AgingEntry(J.nextAgingTime, jobIdx,
agingSeq++));
                } else J.nextAgingTime = Long.MAX_VALUE;
                J.lastEnqId = ++enqCounter;
                queues[J.currentPri].addLast(new QEntry(jobIdx,
J.lastEnqId));
            }
        }

        EnqueueHelper enq = new EnqueueHelper();

        if (n > 0 && arrival[0] > 0) time = 0;
        int finishedCount = 0;

        while (finishedCount < n) {
            while (nextArrivalIdx < n && arrival[nextArrivalIdx] <= time)
{
                Job J = jobs[nextArrivalIdx];
                J.currentPri = J.basePri;
                J.promotionsDone = 0;
                enq.enqueueWaiting(nextArrivalIdx, Math.max(time,
J.arrival));
                nextArrivalIdx++;
            }

            boolean anyReady = false;
            for (int p = K; p >= 1; p--) {
                if (!queues[p].isEmpty()) { anyReady = true; break; }
            }
            if (!anyReady) {
                if (nextArrivalIdx < n) {
                    time = Math.max(time, arrival[nextArrivalIdx]);
                    continue;
                } else break;
            }

            while (!agingHeap.isEmpty() && agingHeap.peek().time <= time)
{
                AgingEntry ae = agingHeap.poll();
                int ji = ae.jobIdx;
                Job J = jobs[ji];
                if (J.finished) continue;
                if (J.nextAgingTime != ae.time) continue;
                if (J.currentPri < K) {
                    J.promotionsDone++;
                    J.currentPri++;
                    J.lastEnqId = ++enqCounter;
                    queues[J.currentPri].addLast(new QEntry(ji,
J.lastEnqId));
                    if (A > 0 && J.currentPri < K) {
                        J.nextAgingTime = J.waitingStart + A *
(J.promotionsDone + 1);
                        agingHeap.add(new AgingEntry(J.nextAgingTime, ji,
agingSeq++));
                    } else J.nextAgingTime = Long.MAX_VALUE;
                }
            }

            int pickPri = -1;
            int pickedJobIdx = -1;
            for (int p = K; p >= 1; p--) {
                ArrayDeque<QEntry> dq = queues[p];
                while (!dq.isEmpty()) {
                    QEntry qent = dq.peekFirst();
                    Job cand = jobs[qent.jobIdx];
                    if (cand.finished || cand.lastEnqId != qent.enqId ||
cand.currentPri != p) {
                        dq.removeFirst();
                        continue;
                    }
                    dq.removeFirst();
                    pickedJobIdx = qent.jobIdx;
                    pickPri = p;
                    break;
                }
                if (pickPri != -1) break;
            }

            if (pickPri == -1) {
                if (nextArrivalIdx < n) {
                    time = Math.max(time, arrival[nextArrivalIdx]);
                    continue;
                } else break;
            }

            Job cur = jobs[pickedJobIdx];
            long run = Math.min(q, cur.remaining);
            long start = time;
            long end = time + run;
            slices.add(new Slice(cur.id, start, end));
            time = end;
            cur.remaining -= run;

            if (cur.remaining == 0) {
                cur.finished = true;
                cur.completionTime = time;
                finishedCount++;
            } else enq.enqueueWaiting(pickedJobIdx, time);
        }

        long[][] metrics = new long[n][4];
        long sumWT = 0;
        long maxLateness = 0;
        long maxWT = Long.MIN_VALUE, minWT = Long.MAX_VALUE;
        for (int i = 0; i < n; i++) {
            Job J = jobs[i];
            long CT = J.completionTime;
            long TAT = CT - J.arrival;
            long WT = TAT - (duration[i]);
            long Lateness = Math.max(0L, CT - J.deadline);
            metrics[i][0] = CT;
            metrics[i][1] = TAT;
```

```java
            metrics[i][2] = WT;
            metrics[i][3] = Lateness;
            sumWT += WT;
            maxLateness = Math.max(maxLateness, Lateness);
            maxWT = Math.max(maxWT, WT);
            minWT = Math.min(minWT, WT);
        }
        double avgWT = n > 0 ? ((double) sumWT) / n : 0.0;
        long fairness = (n > 0) ? (maxWT - minWT) : 0L;
        return new SchedResult(slices, metrics, maxLateness, avgWT,
fairness);
    }

    public static void main(String[] args) {
        int[] ids = {1,2,3};
        long[] arrival = {0,2,4};
        long[] dur = {5,4,2};
        int[] basePri = {1,1,1};
        long[] deadline = {10,12,11};
        long q = 2;
        long A = 3;
        int K = 3;
        SchedResult res = schedule(ids, arrival, dur, basePri, deadline,
q, A, K);
        System.out.println("Slices:");
        for (Slice s : res.slices) System.out.println(s);
        System.out.println("Per-job metrics (CT,TAT,WT,Lateness):");
        for (int i = 0; i < ids.length; i++) {
            System.out.println("Job " + ids[i] + ": " +
Arrays.toString(res.metrics[i]));
        }
        System.out.println("maxLateness=" + res.maxLateness + ", avgWT=" +
res.avgWT + ", fairness=" + res.fairness);
    }
}
```



```
PS E:\JAVA PROGRAMS\stepаryansingh\year2\dsa\week11-12\assignment> java Scheduler
Slices:
(1,0,2)
(1,2,4)
(2,4,6)
(1,6,7)
(3,7,9)
(2,9,11)
Per-job metrics (CT,TAT,WT,Lateness):
Job 1: [7, 7, 2, 0]
Job 2: [11, 9, 5, 0]
Job 3: [9, 5, 3, 0]
maxLateness=0, avgWT=3.3333333333333335, fairness=3
```