**Input Source Program (source):**

```
START 200

READ A

LOOP MOVER AREG,A

SUB AREG,='1'

BC GT,LOOP

STOP

LTORG

A DS 1

END
```

**Intermediate Code File (ic.txt)**

```
(AD,01) (C,200)

(IS,09) (S,01)

(IS,04) (1)(S,01)

(IS,02) (1)(L,01)

(IS,07) (4)(S,02)

(IS,00)

(AD,05)

(DL,02) (C,1)

(AD,02)
```

**Pass1.java**

```
import java.io.*;

import java.util.HashMap;


class symbol {

   String sym;

   int addr;

}
```

```
class littab {
    String lit;
    int addr;
}

public class Pass1 {
    HashMap<String, Integer> OPTAB = new HashMap<String, Integer>();
    HashMap<String, Integer> REGTAB = new HashMap<String, Integer>();
    HashMap<String, Integer> CONDTAB = new HashMap<String, Integer>();
    HashMap<String, Integer> ADTAB = new HashMap<String, Integer>();

    int MAX = 20;
    symbol SYMTAB[] = new symbol[MAX];
    littab LITTAB[] = new littab[MAX];

    String buffer;
    int lc, litcnt = 0, poolcnt = 0, proc_lit = 0, symcount = 0;

    Pass1() {
        initialize_OPTAB();
        initialize_REGTAB();
        initialize_CONDTAB();
        initialize_ADTAB();

        for (int i = 0; i < MAX; i++) {
            SYMTAB[i] = new symbol();
            LITTAB[i] = new littab();
        }
    }
```

```
public void initialize_OPTAB() {
    OPTAB.put("STOP", 0);
    OPTAB.put("ADD", 1);
    OPTAB.put("SUB", 2);
    OPTAB.put("MULT", 3);
    OPTAB.put("MOVER", 4);
    OPTAB.put("MOVEM", 5);
    OPTAB.put("COMP", 6);
    OPTAB.put("BC", 7);
    OPTAB.put("DIV", 8);
    OPTAB.put("READ", 9);
    OPTAB.put("PRINT", 10);
}

public void initialize_REGTAB() {
    REGTAB.put("AREG", 1);
    REGTAB.put("BREG", 2);
    REGTAB.put("CREG", 3);
    REGTAB.put("DREG", 4);
}

public void initialize_CONDTAB() {
    CONDTAB.put("LT", 1);
    CONDTAB.put("LE", 2);
    CONDTAB.put("EQ", 3);
    CONDTAB.put("GT", 4);
    CONDTAB.put("GE", 5);
    CONDTAB.put("ANY", 6);
}

public void initialize_ADTAB() {
    ADTAB.put("START", 1);
    ADTAB.put("END", 2);
    ADTAB.put("ORIGIN", 3);
    ADTAB.put("EQU", 4);
    ADTAB.put("LTORG", 5);
}

public int search_OPTAB(String str) { return OPTAB.getOrDefault(str, -1); }
public int search_REGTAB(String str) { return REGTAB.getOrDefault(str, -1); }
public int search_CONDTAB(String str) { return CONDTAB.getOrDefault(str, -1); }
public int search_ADTAB(String str) { return ADTAB.getOrDefault(str, -1); }

public int search_symbol(String str) {
    for (int i = 0; i < symcount; i++) {
        if (str.equals(SYMTAB[i].sym)) return i;
    }
    return -1;
}

void passone() throws IOException {
    int n, i = 0, j = 0, p, k;

    FileReader source_file = new FileReader("source");
    BufferedReader fs = new BufferedReader(source_file);

    FileWriter ic_file = new FileWriter("ic.txt");
    BufferedWriter ft = new BufferedWriter(ic_file);

    while ((buffer = fs.readLine()) != null) {
        String[] tokens = buffer.split(" |\\,");
        n = tokens.length;
```

```
switch (n) {
    case 1:
        i = search_OPTAB(tokens[0]);
        if (i == 0) {
            ft.write("(IS," + String.format("%02d)", i));
            lc++;
            break;
        }
        i = search_ADTAB(tokens[0]);
        if (i == 2 || i == 5) {
            for (j = proc_lit; j < litcnt; j++) {
                LITTAB[j].addr = lc++;
            }
            proc_lit = litcnt;
            ft.write("(AD," + String.format("%02d)", i));
        }
        break;

    case 2:
        i = search_ADTAB(tokens[0]);
        if (i == 1 || i == 3) {
            lc = Integer.parseInt(tokens[1]);
            ft.write("(AD," + String.format("%02d)", i) + " (C," + tokens[1] + ")");
            break;
        }
        i = search_OPTAB(tokens[0]);
        if (i == 9 || i == 10) {
            p = search_symbol(tokens[1]);
            if (p == -1) {
                SYMTAB[symcount].sym = tokens[1];
```

```
                        symcount++;
                        ft.write("(IS," + String.format("%02d)", i) + " (S," + String.format("%02d)", symcount));
                    } else {
                        ft.write("(IS," + String.format("%02d)", i) + " (S," + String.format("%02d)", p));
                    }
                    lc++;
                    break;
                }
                break;

            case 3:
                i = search_OPTAB(tokens[0]);
                if (i >= 1 && i <= 8) {
                    lc++;
                    if (i == 7)
                        k = search_CONDTAB(tokens[1]);
                    else
                        k = search_REGTAB(tokens[1]);

                    if (tokens[2].charAt(0) == '=') {
                        String teemp = tokens[2].substring(2, 3);
                        LITTAB[litcnt].lit = teemp;
                        litcnt++;
                        ft.write("(IS," + String.format("%02d) (", i) + k + ")(L," + String.format("%02d)", litcnt));
                    } else {
                        p = search_symbol(tokens[2]);
                        if (p == -1) {
                            SYMTAB[symcount].sym = tokens[2];
                            symcount++;
                            ft.write("(IS," + String.format("%02d) (", i) + k + ")(S," + String.format("%02d)",
symcount));
```

```
                    } else {
                        ft.write("(IS," + String.format("%02d) (", i) + k + ")(S," + String.format("%02d)",
symcount));
                    }
                }
                break;
            }
            if (tokens[1].equals("DS")) {
                p = search_symbol(tokens[0]);
                if (p == -1) {
                    SYMTAB[symcount].sym = tokens[0];
                    SYMTAB[symcount].addr = lc;
                    symcount++;
                    ft.write("(DL,02) (C," + tokens[2] + ")");
                } else {
                    SYMTAB[p].addr = lc;
                    ft.write("(DL,02) (C," + tokens[2] + ")");
                }
                lc = lc + Integer.parseInt(tokens[2]);
                break;
            }
            if (tokens[1].equals("DC")) {
                p = search_symbol(tokens[0]);
                if (p == -1) {
                    SYMTAB[symcount].sym = tokens[0];
                    SYMTAB[symcount].addr = lc;
                    symcount++;
                    ft.write("(DL,01) (C," + tokens[2]);
                } else {
                    SYMTAB[p].addr = lc;
                    ft.write("(DL,01) (C," + tokens[2]);
                }
                break;
            }
            break;
        }
        ft.write("\n");
    }
    ft.close();
}

void print_littab() {
    for (int i = 0; i < litcnt; i++) {
        System.out.println(LITTAB[i].lit + "\t" + LITTAB[i].addr);
    }
}

void print_symtab() {
    for (int i = 0; i < symcount; i++) {
        System.out.println(SYMTAB[i].sym + "\t" + SYMTAB[i].addr);
    }
}

void print_srcfile() throws IOException {
    FileReader source_file = new FileReader("source");
    BufferedReader fs = new BufferedReader(source_file);
    String buffer;
    while ((buffer = fs.readLine()) != null) {
        System.out.println(buffer);
    }
    fs.close();
}
```

```
void print_icfile() throws IOException {
    FileReader source_file = new FileReader("ic.txt");
    BufferedReader fs = new BufferedReader(source_file);
    String buffer;
    while ((buffer = fs.readLine()) != null) {
        System.out.println(buffer);
    }
    fs.close();
}

public static void main(String[] args) throws IOException {
    Pass1 obj = new Pass1();
    obj.passone();

    System.out.println("SOURCE CODE\n");
    obj.print_srcfile();

    System.out.println("\n\n***********************************************");
    System.out.println("\n\nINTERMEDIATE CODE\n");
    obj.print_icfile();

    System.out.println("\n\n***********************************************");
    System.out.println("\n\nSYMBOL TABLE");
    System.out.println("================");
    System.out.println("Symbol\tAddress");
    System.out.println("================");
    obj.print_symtab();

    System.out.println("\n\n***********************************************");
    System.out.println("\n\nLITERAL TABLE");
```

```
        System.out.println("=================");
        System.out.println("Literal\tAddress");
        System.out.println("=================");
        obj.print_littab();
    }
}
```

**Output :**

**SOURCE CODE**

```
START 200
READ A
LOOP MOVER AREG,A
SUB AREG,='1'
BC GT,LOOP
STOP
LTORG
A DS 1
END
```

```
*************************************************
```

**INTERMEDIATE CODE**

```
(AD,01) (C,200)
(IS,09) (S,01)
(IS,04) (1)(S,01)
(IS,02) (1)(L,01)
(IS,07) (4)(S,02)
```

```
(IS,00)

(AD,05)

(DL,02) (C,1)

(AD,02)



**************************************************


SYMBOL TABLE

================

Symbol   Address

================

A       205

LOOP     201



**************************************************


LITERAL TABLE

================

Literal  Address

================

1       203
```

**File: intermediate.txt**

```
(AD,01)(C,200)
(IS,04)(1)(L,1)
(IS,05)(1)(S,1)
(IS,04)(1)(S,1)
(IS,04)(3)(S,3)
(IS,01)(3)(L,2)
(IS,07)(6)(S,4)
(DL,01)(C,5)
(DL,01)(C,1)
(IS,02)(1)(L,3)
(IS,07)(1)(S,5)
(IS,00)
(AD,03)(S,2)+2
(IS,03)(3)(S,3)
(AD,03)(S,6)+1
(DL,02)(C,1)
(DL,02)(C,1)
(AD,02)
(DL,01)(C,1)
```

**File: littab.txt**

```
5    206
1    207
1    213
```

**File: symtab.txt**

```
A      211   1
LOOP   202   1
```

```
B     212   1

NEXT   208   1

BACK   202   1

LAST   210   1
```

**File: Pass2.java**

```java
import java.io.BufferedReader;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.util.HashMap;


public class Pass2 {

    public static void main(String[] Args) throws IOException {

        BufferedReader b1 = new BufferedReader(new FileReader("intermediate.txt"));

        BufferedReader b2 = new BufferedReader(new FileReader("symtab.txt"));

        BufferedReader b3 = new BufferedReader(new FileReader("littab.txt"));

        FileWriter f1 = new FileWriter("Pass2.txt");


        HashMap<Integer, String> symSymbol = new HashMap<Integer, String>();

        HashMap<Integer, String> litSymbol = new HashMap<Integer, String>();

        HashMap<Integer, String> litAddr = new HashMap<Integer, String>();


        String s;

        int symtabPointer = 1, littabPointer = 1, offset;


        while ((s = b2.readLine()) != null) {

            String word[] = s.split("\t\t\t");

            symSymbol.put(symtabPointer++, word[1]);

        }
```

```
while ((s = b3.readLine()) != null) {
    String word[] = s.split("\t\t");
    litSymbol.put(littabPointer, word[0]);
    litAddr.put(littabPointer++, word[1]);
}

while ((s = b1.readLine()) != null) {
    if (s.substring(1, 6).compareToIgnoreCase("IS,00") == 0) {
        f1.write("+ 00 0 000\n");
    } else if (s.substring(1, 3).compareToIgnoreCase("IS") == 0) {
        f1.write("+ " + s.substring(4, 6) + " ");
        if (s.charAt(9) == ')') {
            f1.write(s.charAt(8) + " ");
            offset = 3;
        } else {
            f1.write("0 ");
            offset = 0;
        }

        if (s.charAt(8 + offset) == 'S')
            f1.write(symSymbol.get(Integer.parseInt(s.substring(10 + offset, s.length() - 1))) + "\n");
        else
            f1.write(litAddr.get(Integer.parseInt(s.substring(10 + offset, s.length() - 1))) + "\n");
    } else if (s.substring(1, 6).compareToIgnoreCase("DL,01") == 0) {
        String s1 = s.substring(10, s.length() - 1), s2 = "";
        for (int i = 0; i < 3 - s1.length(); i++)
            s2 += "0";
        s2 += s1;
        f1.write("+ 00 0 " + s2 + "\n");
    } else {
```

```
                f1.write("\n");
            }
        }

        f1.close();
        b1.close();
        b2.close();
        b3.close();
    }
}
```

**File: Pass2.txt (Output)**

```
 + 04 1 206
 + 05 1 211
 + 04 1 211
 + 04 3 212
 + 01 3 207
 + 07 6 208
 + 00 0 005
 + 00 0 001
 + 02 1 213
 + 07 1 202
 + 00 0 000

 + 03 3 212

+ 00 0 001
```

```
import java.util.*;

/**
 * CPU Scheduling Algorithms Demo
 * ----------------------------------
 * This program demonstrates 4 scheduling algorithms:
 * 1) FCFS (First Come First Serve)
 * 2) SRTF (Shortest Remaining Time First - Preemptive SJF)
 * 3) Priority Scheduling (Non-preemptive)
 * 4) Round Robin
 *
 * Author: ChatGPT (for teaching purposes)
 */
class Main {

    // ---------- Process Class (common to all algorithms) ----------
    static class Process {
        int pid;   // Process ID
        int at;    // Arrival Time
        int bt;    // Burst Time
        int pr;    // Priority (for priority scheduling)
        int ct;    // Completion Time
        int tat;   // Turnaround Time
        int wt;    // Waiting Time
        int rem;   // Remaining Time (for SRTF & RR)

        Process(int pid, int at, int bt) {
            this.pid = pid;
            this.at = at;
        this.bt = bt;
        this.rem = bt;  // initially rem = bt
    }

    Process(int pid, int at, int bt, int pr) {
        this(pid, at, bt);
        this.pr = pr;
    }
}

// ---------- 1) FCFS ----------
static void fcfs(List<Process> list) {
    System.out.println("\n--- FCFS Scheduling ---");

    // Sort by arrival time
    Collections.sort(list, (a, b) -> a.at - b.at);

    int time = 0;
    List<String> gantt = new ArrayList<>();

    for (Process p : list) {
        if (time < p.at) time = p.at; // CPU idle if no process yet
        int start = time;
        time += p.bt;
        p.ct = time;
        p.tat = p.ct - p.at;
        p.wt = p.tat - p.bt;
        gantt.add("P" + p.pid + "(" + start + "-" + time + ")");
    }

    // Print results
    printResults(list, gantt);
}

// ---------- 2) SRTF ----------
static void srtf(List<Process> list) {
    System.out.println("\n--- SRTF Scheduling ---");

    int n = list.size();
    int completed = 0, time = 0;
    List<String> gantt = new ArrayList<>();
    int lastPid = -1, segStart = 0;

    // start at earliest arrival
    int earliest = list.stream().mapToInt(p -> p.at).min().getAsInt();
    time = earliest;

    while (completed < n) {
        int idx = -1, minRem = Integer.MAX_VALUE;
        for (int i = 0; i < n; i++) {
            Process p = list.get(i);
            if (p.at <= time && p.rem > 0) {
                if (p.rem < minRem) {
                    minRem = p.rem;
                    idx = i;
                }
            }
        }

        if (idx == -1) { time++; continue; } // idle

        Process cur = list.get(idx);
        if (lastPid != cur.pid) {
            if (lastPid != -1) gantt.add("P" + lastPid + "(" + segStart + "-" + time + ")");
            lastPid = cur.pid;
            segStart = time;
        }

        cur.rem--; // run for 1 unit
        time++;
        if (cur.rem == 0) {
            cur.ct = time;
            cur.tat = cur.ct - cur.at;
            cur.wt = cur.tat - cur.bt;
            completed++;
        }
    }
    if (lastPid != -1) gantt.add("P" + lastPid + "(" + segStart + "-" + time + ")");

    printResults(list, gantt);
}

// ---------- 3) Priority (Non-preemptive) ----------
static void priorityNonPreemptive(List<Process> list) {
    System.out.println("\n--- Priority Scheduling (Non-preemptive) ---");

    int n = list.size();
    int completed = 0, time = 0;
    List<String> gantt = new ArrayList<>();
    boolean[] done = new boolean[n];

    // start at earliest arrival
    int earliest = list.stream().mapToInt(p -> p.at).min().getAsInt();
    time = earliest;

    while (completed < n) {
        int idx = -1, bestPr = Integer.MAX_VALUE;
        for (int i = 0; i < n; i++) {
            Process p = list.get(i);
            if (!done[i] && p.at <= time) {
                if (p.pr < bestPr) {
                    bestPr = p.pr;
                    idx = i;
                }
            }
        }

        if (idx == -1) { time++; continue; }

        Process cur = list.get(idx);
        int start = time;
        time += cur.bt;
        cur.ct = time;
        cur.tat = cur.ct - cur.at;
        cur.wt = cur.tat - cur.bt;
        done[idx] = true;
        completed++;
        gantt.add("P" + cur.pid + "(" + start + "-" + time + ")");
    }

    printResults(list, gantt);
}

// ---------- 4) Round Robin ----------
```

```
static void roundRobin(List<Process> list, int q) {
   System.out.println("\n--- Round Robin Scheduling (q = " + q + ") ---");

   Queue<Process> ready = new LinkedList<>();
   int n = list.size(), completed = 0, time = 0, idx = 0;
   List<String> gantt = new ArrayList<>();

   // sort by arrival
   Collections.sort(list, (a, b) -> a.at - b.at);
   int earliest = list.stream().mapToInt(p -> p.at).min().getAsInt();
   time = earliest;

   while (completed < n) {
      while (idx < n && list.get(idx).at <= time) {
         ready.add(list.get(idx));
         idx++;
      }
      if (ready.isEmpty()) { time++; continue; }

      Process cur = ready.poll();
      int start = time;
      int run = Math.min(q, cur.rem);
      cur.rem -= run;
      time += run;
      gantt.add("P" + cur.pid + "(" + start + "-" + time + ")");

      while (idx < n && list.get(idx).at <= time) {
         ready.add(list.get(idx));
         idx++;
      }
```

```
        if (cur.rem > 0) ready.add(cur);
        else {
            cur.ct = time;
            cur.tat = cur.ct - cur.at;
            cur.wt = cur.tat - cur.bt;
            completed++;
        }
    }

    printResults(list, gantt);
}

// ---------- Utility: Print Gantt + Table ----------
static void printResults(List<Process> list, List<String> gantt) {
    Collections.sort(list, (a, b) -> a.pid - b.pid);

    System.out.println("\nGantt Chart: " + gantt);
    System.out.println("PID\tAT\tBT\tCT\tTAT\tWT");
    double sumWT = 0, sumTAT = 0;
    for (Process p : list) {
        System.out.printf("P%d\t%d\t%d\t%d\t%d\t%d\n",
            p.pid, p.at, p.bt, p.ct, p.tat, p.wt);
        sumWT += p.wt; sumTAT += p.tat;
    }
    System.out.printf("\nAverage WT = %.2f\n", sumWT / list.size());
    System.out.printf("Average TAT = %.2f\n", sumTAT / list.size());
}

// ---------- MAIN ----------
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
```

```
System.out.println("CPU Scheduling Algorithms");
System.out.println("1. FCFS");
System.out.println("2. SRTF");
System.out.println("3. Priority (Non-preemptive)");
System.out.println("4. Round Robin");
System.out.print("Choose algorithm: ");
int choice = sc.nextInt();

System.out.print("Enter number of processes: ");
int n = sc.nextInt();

List<Process> list = new ArrayList<>();
for (int i = 1; i <= n; i++) {
  System.out.printf("P%d Arrival Time: ", i); int at = sc.nextInt();
  System.out.printf("P%d Burst Time: ", i); int bt = sc.nextInt();
  if (choice == 3) {
    System.out.printf("P%d Priority: ", i); int pr = sc.nextInt();
    list.add(new Process(i, at, bt, pr));
  } else {
    list.add(new Process(i, at, bt));
  }
}

switch (choice) {
  case 1: fcfs(list); break;
  case 2: srtf(list); break;
  case 3: priorityNonPreemptive(list); break;
  case 4:
    System.out.print("Enter Time Quantum: ");
    int q = sc.nextInt();
```

```
                roundRobin(list, q);
                break;
            default: System.out.println("Invalid choice.");
        }

        sc.close();
    }
}
```

**OUTPUT(all cases)**

**1) FCFS (First Come First Serve)**

**Gantt chart:**

P1(0-7)  P2(7-11)  P3(11-12)  P4(12-16)

**Table (sorted by PID):**

| PID | AT | BT | CT | TAT = CT-AT | WT = TAT-BT |
|---|---|---|---|---|---|
| P1 | 0 | 7 | 7 | 7 | 0 |
| P2 | 2 | 4 | 11 | 9 | 5 |
| P3 | 4 | 1 | 12 | 8 | 7 |
| P4 | 5 | 4 | 16 | 11 | 7 |

**Averages:**

- Average Waiting Time = (0 + 5 + 7 + 7) / 4 = **4.75**
- Average Turnaround Time = (7 + 9 + 8 + 11) / 4 = **8.75**

---

**2) SRTF (Shortest Remaining Time First — Preemptive SJF)**

Simulated stepwise (time unit by unit). Key segments:

- P1 runs 0–2, preempted by P2
- P2 runs 2–4, preempted by arriving P3
- P3 runs 4–5 and completes
- P2 continues 5–7 and completes
- P4 runs 7–11 and completes

- P1 finishes 11–16

**Gantt chart:**

P1(0-2) P2(2-4) P3(4-5) P2(5-7) P4(7-11) P1(11-16)

**Table (sorted by PID):**

| PID | AT | BT | CT | TAT = CT-AT | WT = TAT-BT |
|---|---|---|---|---|---|
| P1 | 0 | 7 | 16 | 16 | 9 |
| P2 | 2 | 4 | 7 | 5 | 1 |
| P3 | 4 | 1 | 5 | 1 | 0 |
| P4 | 5 | 4 | 11 | 6 | 2 |

**Averages:**

- Average Waiting Time = (9 + 1 + 0 + 2) / 4 = **3.00**
- Average Turnaround Time = (16 + 5 + 1 + 6) / 4 = **7.00**

SRTF gives the best average waiting & turnaround here because short jobs (P3, P2) get priority when they arrive.

---

**3) Priority Scheduling (Non-preemptive)**

(Here lower priority number = higher priority)

At each scheduling decision (when CPU free) pick arrived process with smallest pr.

**Execution order chosen:**

- Start at t=0 → P1 runs 0–7 (non-preemptive)
- t=7 choose among arrived {P2(pr1),P3(pr3),P4(pr2)} → P2 (pr1) runs 7–11
- then choose P4 (pr2) runs 11–15
- finally P3 (pr3) runs 15–16

**Gantt chart:**

P1(0-7) P2(7-11) P4(11-15) P3(15-16)

**Table (sorted by PID):**

| PID | AT | BT | CT | TAT = CT-AT | WT = TAT-BT |
|---|---|---|---|---|---|
| P1 | 0 | 7 | 7 | 7 | 0 |
| P2 | 2 | 4 | 11 | 9 | 5 |
| P3 | 4 | 1 | 16 | 12 | 11 |

| **PID** | **AT** | **BT** | **CT** | **TAT = CT-AT** | **WT = TAT-BT** |
|---|---|---|---|---|---|
| P4 | 5 | 4 | 15 | 10 | 6 |

**Averages:**

- Average Waiting Time = (0 + 5 + 11 + 6) / 4 = **5.50**
- Average Turnaround Time = (7 + 9 + 12 + 10) / 4 = **9.50**

Priority (non-preemptive) penalized P3 here (arrived earlier than P4 but lower priority), causing larger WT/TAT for P3.

---

**4) Round Robin (Quantum = 2)**

Simulated with a ready queue and arrivals enqueued when arrival_time <= current_time. Time quantum = 2.

Sequence of time slices produced:

**Gantt chart:**

P1(0-2) P2(2-4) P1(4-6) P3(6-7) P2(7-9) P4(9-11) P1(11-13) P4(13-15) P1(15-16)

(You can read this as: P1 ran 0–2, P2 2–4, P1 4–6, P3 6–7 (only 1 unit), P2 7–9, P4 9–11, P1 11–13, P4 13–15, P1 15–16.)

**Table (sorted by PID):**

| **PID** | **AT** | **BT** | **CT** | **TAT = CT-AT** | **WT = TAT-BT** |
|---|---|---|---|---|---|
| P1 | 0 | 7 | 16 | 16 | 9 |
| P2 | 2 | 4 | 9 | 7 | 3 |
| P3 | 4 | 1 | 7 | 3 | 2 |
| P4 | 5 | 4 | 15 | 10 | 6 |

**Averages:**

- Average Waiting Time = (9 + 3 + 2 + 6) / 4 = **5.00**
- Average Turnaround Time = (16 + 7 + 3 + 10) / 4 = **9.00**

```
import java.util.Scanner;

public class Main {
    static void firstFit(int blockSize[], int m, int processSize[], int n) {
        int allocation[] = new int[n];

        for (int i = 0; i < n; i++)
            allocation[i] = -1;

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                if (blockSize[j] >= processSize[i]) {
                    allocation[i] = j;
                    blockSize[j] -= processSize[i];
                    break;
                }
            }
        }

        System.out.println("\nProcess No.\tProcess Size\tBlock no.");
        for (int i = 0; i < n; i++) {
            System.out.print(" " + (i + 1) + "\t\t" +
                             processSize[i] + "\t\t");
            if (allocation[i] != -1)
                System.out.println(allocation[i] + 1);
            else
                System.out.println("Not Allocated");
        }
    }
```

```
static void bestFit(int blockSize[], int m, int processSize[], int n) {
  int allocation[] = new int[n];

  for (int i = 0; i < n; i++)
    allocation[i] = -1;

  for (int i = 0; i < n; i++) {
    int bestIdx = -1;
    for (int j = 0; j < m; j++) {
      if (blockSize[j] >= processSize[i]) {
        if (bestIdx == -1 || blockSize[j] < blockSize[bestIdx])
          bestIdx = j;
      }
    }

    if (bestIdx != -1) {
      allocation[i] = bestIdx;
      blockSize[bestIdx] -= processSize[i];
    }
  }

  System.out.println("\nProcess No.\tProcess Size\tBlock no.");
  for (int i = 0; i < n; i++) {
    System.out.print(" " + (i + 1) + "\t\t" +
            processSize[i] + "\t\t");
    if (allocation[i] != -1)
      System.out.println(allocation[i] + 1);
    else
      System.out.println("Not Allocated");
  }
```

```
}

static void worstFit(int blockSize[], int m, int processSize[], int n) {
    int allocation[] = new int[n];

    for (int i = 0; i < n; i++)
        allocation[i] = -1;

    for (int i = 0; i < n; i++) {
        int wstIdx = -1;
        for (int j = 0; j < m; j++) {
            if (blockSize[j] >= processSize[i]) {
                if (wstIdx == -1 || blockSize[j] > blockSize[wstIdx])
                    wstIdx = j;
            }
        }

        if (wstIdx != -1) {
            allocation[i] = wstIdx;
            blockSize[wstIdx] -= processSize[i];
        }
    }

    System.out.println("\nProcess No.\tProcess Size\tBlock no.");
    for (int i = 0; i < n; i++) {
        System.out.print(" " + (i + 1) + "\t\t" +
                         processSize[i] + "\t\t");
        if (allocation[i] != -1)
            System.out.println(allocation[i] + 1);
        else
            System.out.println("Not Allocated");
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter number of blocks: ");
        int m = sc.nextInt();
        int blockSize[] = new int[m];
        System.out.println("Enter block sizes:");
        for (int i = 0; i < m; i++) blockSize[i] = sc.nextInt();

        System.out.print("Enter number of processes: ");
        int n = sc.nextInt();
        int processSize[] = new int[n];
        System.out.println("Enter process sizes:");
        for (int i = 0; i < n; i++) processSize[i] = sc.nextInt();

        int choice;
        do {
            System.out.println("\n--- Memory Allocation Strategies ---");
            System.out.println("1. First Fit");
            System.out.println("2. Best Fit");
            System.out.println("3. Worst Fit");
            System.out.println("4. Exit");
            System.out.print("Enter your choice: ");
            choice = sc.nextInt();

            int tempBlock[] = blockSize.clone();

            switch (choice) {
                case 1: firstFit(tempBlock, m, processSize, n); break;
                case 2: bestFit(tempBlock, m, processSize, n); break;
                case 3: worstFit(tempBlock, m, processSize, n); break;
                case 4: System.out.println("Exiting..."); break;
                default: System.out.println("Invalid choice!");
            }
        } while (choice != 4);
    }
}
```

**FIRST FIT — Output**

```
Process No.     Process Size     Block no.
1               212              2
2               417              5
3               112              2
4               426              Not Allocated
```

**BEST FIT — Output**

```
Process No.     Process Size     Block no.
1               212              4
2               417              5
3               112              5
4               426              Not Allocated
```

**WORST FIT — Output**

```
Process No.     Process Size     Block no.
1               212              5
2               417              2
3               112              5
4               426              Not Allocated
```

```
import java.util.*;

public class Main {

  // FIFO Page Replacement
  static void fifo(int pages[], int n, int capacity) {
    Set<Integer> s = new HashSet<>(capacity);
    Queue<Integer> q = new LinkedList<>();
    int pageFaults = 0;

    for (int i = 0; i < n; i++) {
      if (!s.contains(pages[i])) {
        if (s.size() == capacity) {
          int removed = q.poll();
          s.remove(removed);
        }
        s.add(pages[i]);
        q.add(pages[i]);
        pageFaults++;
      }
      System.out.println("Step " + (i + 1) + " -> Frames: " + q);
    }
    System.out.println("Total Page Faults (FIFO): " + pageFaults);
  }

  // LRU Page Replacement
  static void lru(int pages[], int n, int capacity) {
    Set<Integer> s = new HashSet<>(capacity);
    Map<Integer, Integer> indexes = new HashMap<>();
    int pageFaults = 0;

    for (int i = 0; i < n; i++) {
        if (s.size() < capacity) {
            if (!s.contains(pages[i])) {
                s.add(pages[i]);
                pageFaults++;
            }
            indexes.put(pages[i], i);
        } else {
            if (!s.contains(pages[i])) {
                int lru = Integer.MAX_VALUE, val = -1;
                for (int x : s) {
                    if (indexes.get(x) < lru) {
                        lru = indexes.get(x);
                        val = x;
                    }
                }
                s.remove(val);
                s.add(pages[i]);
                pageFaults++;
            }
            indexes.put(pages[i], i);
        }
        System.out.println("Step " + (i + 1) + " -> Frames: " + s);
    }
    System.out.println("Total Page Faults (LRU): " + pageFaults);
}

// Optimal Page Replacement
static void optimal(int pages[], int n, int capacity) {
```

```
Set<Integer> s = new HashSet<>(capacity);
int pageFaults = 0;

for (int i = 0; i < n; i++) {
    if (s.size() < capacity) {
        if (!s.contains(pages[i])) {
            s.add(pages[i]);
            pageFaults++;
        }
    } else {
        if (!s.contains(pages[i])) {
            int farthest = i + 1, val = -1;
            for (int x : s) {
                int j;
                for (j = i + 1; j < n; j++) {
                    if (pages[j] == x) break;
                }
                if (j == n) {
                    val = x;
                    break;
                }
                if (j > farthest) {
                    farthest = j;
                    val = x;
                }
            }
            s.remove(val);
            s.add(pages[i]);
            pageFaults++;
        }
    }
```

```java
            System.out.println("Step " + (i + 1) + " -> Frames: " + s);
        }
        System.out.println("Total Page Faults (Optimal): " + pageFaults);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter number of pages: ");
        int n = sc.nextInt();
        int pages[] = new int[n];
        System.out.println("Enter page reference string:");
        for (int i = 0; i < n; i++) pages[i] = sc.nextInt();

        System.out.print("Enter frame capacity: ");
        int capacity = sc.nextInt();

        int choice;
        do {
            System.out.println("\n--- Page Replacement Algorithms ---");
            System.out.println("1. FIFO");
            System.out.println("2. LRU");
            System.out.println("3. Optimal");
            System.out.println("4. Exit");
            System.out.print("Enter your choice: ");
            choice = sc.nextInt();

            switch (choice) {
                case 1: fifo(pages, n, capacity); break;
                case 2: lru(pages, n, capacity); break;
                case 3: optimal(pages, n, capacity); break;
                case 4: System.out.println("Exiting..."); break;
                default: System.out.println("Invalid choice!");
            }
        } while (choice != 4);
    }
}
```

**OUTPUT (all cases):**

**FIFO Page Replacement Output**

```
Step 1 -> Frames: [7]
Step 2 -> Frames: [7, 0]
Step 3 -> Frames: [7, 0, 1]
Step 4 -> Frames: [0, 1, 2]
Step 5 -> Frames: [0, 1, 2]
Step 6 -> Frames: [1, 2, 3]
Step 7 -> Frames: [2, 3, 0]
Step 8 -> Frames: [3, 0, 4]
Step 9 -> Frames: [0, 4, 2]
Step 10 -> Frames: [4, 2, 3]
Step 11 -> Frames: [2, 3, 0]
Step 12 -> Frames: [2, 3, 0]

Total Page Faults (FIFO): 10
```

**LRU Page Replacement Output**

```
Step 1 -> Frames: [7]
Step 2 -> Frames: [0, 7]
Step 3 -> Frames: [0, 1, 7]
Step 4 -> Frames: [0, 1, 2]
Step 5 -> Frames: [0, 1, 2]
Step 6 -> Frames: [0, 2, 3]
Step 7 -> Frames: [0, 2, 3]
Step 8 -> Frames: [0, 3, 4]
Step 9 -> Frames: [0, 2, 4]
Step 10 -> Frames: [2, 3, 4]
Step 11 -> Frames: [0, 3, 4]
Step 12 -> Frames: [0, 3, 4]

Total Page Faults (LRU): 9
```

**Optimal Page Replacement Output**

```
Step 1 -> Frames: [7]
Step 2 -> Frames: [0, 7]
Step 3 -> Frames: [0, 1, 7]
Step 4 -> Frames: [0, 1, 2]
Step 5 -> Frames: [0, 1, 2]
Step 6 -> Frames: [0, 2, 3]
Step 7 -> Frames: [0, 2, 3]
Step 8 -> Frames: [0, 4, 3]
Step 9 -> Frames: [0, 2, 4]
Step 10 -> Frames: [3, 2, 4]
Step 11 -> Frames: [3, 2, 0]
Step 12 -> Frames: [3, 2, 0]

Total Page Faults (Optimal): 7
```

Tokens.c

```
%{
#include <stdio.h>
int kw=0, iden=0, num=0, w=0;
%}

DIGIT [0-9]
NUMBER {DIGIT}+
REAL {DIGIT}*"."{DIGIT}+
TEXT [a-zA-Z]+
TEXT_NUMBERS [a-zA-Z0-9]+

CONDITIONALS "if"|"else"|"else if"|"switch"|"case"
KEYWORD "break"|"class"|"args"|"nextInt"|"continue"|"goto"|"print"|"Scanner"|"System"|"out"|"println"|"new"|"try"|"catch"|"Exception"|"public"|"close"|"return"|"int"|"float"|"char"|"unsigned"|"signed"|"String"|"long"|"double"|"static"|"void"|"main"
ITERATORS "for"|"while"|"do"
PREPROCESSOR "import"|"java.util.Scanner"|"package"
DELIMITER [;:\t\n(){}]
IDENTIFIER [a-zA-Z][a-zA-Z0-9_]*

NON_IDENTIFIER {NUMBER}[A-Za-z]+
COMMENT "/*"[a-zA-Z0-9 \t\n;.~!@#$%^&*()_+=<>?:"{}]*"*/"

OPERATOR "+"|"-"|"*"|"/"|"="
```

```
UNARY "++"|"--"

LOPERATOR "&&"|"||"|">="|"<="|"=="|"&"|"|"|"~"|">"|"<"


%%


{CONDITIONALS} { printf("%s is a conditional\n", yytext); kw++; }

{ITERATORS} { printf("%s is an iterator\n", yytext); kw++; }

{REAL} { printf("%s is a real number\n", yytext); num++; }

{NUMBER} { printf("%s is a number\n", yytext); num++; }

{PREPROCESSOR} { printf("%s is a preprocessor directive\n",yytext); kw++; }

{KEYWORD} { printf("%s is a keyword\n", yytext); kw++; }

{COMMENT} { printf("%s is a comment\n", yytext); }

{IDENTIFIER} { printf("%s is an identifier\n", yytext); w++; }

{OPERATOR} { printf("%s is a mathematical operator\n", yytext); }

{LOPERATOR} { printf("%s is a logical operator\n", yytext); }

{UNARY} { printf("%s is a unary operator\n", yytext); }

{DELIMITER} { /*ignore*/ }

. { /*ignore unknown*/ }


%%


int main() {

   yyin = fopen("add.java","r");

   yylex();

   printf("\nNumber of identifiers: %d\nNumber of keywords: %d\nNumber of numbers: 
%d\n", w, kw, num);
```

```
    return 0;
}

int yywrap(){ return 1; }
```

add.java

```
package lex_example;
import java.util.Scanner;
public class add {
    public static void main(String[] args){
        Scanner in = new Scanner(System.in);
        int a,b;
        System.out.print("Enter two numbers: ");
        a = in.nextInt();
        b = in.nextInt();
        int c = a + b;
        System.out.println("Addition of "+a+" and "+b+" is: "+c);
        a = 5;
        b = 10;
        c = a + b;
        System.out.println("Addition of "+a+" and "+b+" is: "+c);
    }
}
```

Output :

```
package is a preprocessor directive

import is a preprocessor directive

public is a keyword

class is a keyword

add is an identifier

int is a keyword

a is an identifier

= is a mathematical operator

5 is a number

b is an identifier

= is a mathematical operator

10 is a number

…

Number of identifiers: 37

Number of keywords: 27

Number of numbers: 2
```