**Project brief:**

A pharmaceutical company manufactures 4 product lines from its factory in Dublin. The production line is sophisticated and audited to the highest quality standards. Team members need to pass a certification exam in order to work on each production line. Each team member is certified to work on 1 or more lines and there are 4 production teams in total.

Each production line is used for the manufacture of multiple products in a multi-stage production process which ends when the product is packed for dispatch.

QA is important to the company. Each product is manufactured in a single identified batch and faults or issues identified at every stage are logged and tracked.

Working in the company's IT department, you have been given access to the quality logs of each line. You have been asked to assist the management team in preparing information on the manufacturing process. The quality logs contain the following data and are newly created for each month:-

* 1. Line Code – Numeric
  2. Batch code - Numeric
  3. Batch date & time – numbered for day of month, hour of day, minute of hour.
  4. Product Id - numeric
  5. Issue Code & description - numeric + text
  6. Resolution code & description – numeric + text
  7. Reporting employee id – numeric

**Pseudocode:**

Structure LogEntry:

Integer product\_id

Integer line\_id

Integer issue\_code

Integer day

Integer month

Integer year

Integer hour

Integer minute

Array of Strings employee[MAX\_EMPLOYEES]

String issue\_description[50]

String resolution\_description[50]

Function compare(a, b):

Compare dates and times of log entries a and b

Function sortLogs(logs, num\_logs):

Sort logs by date & time using compare function

Function reportIssues(logs, num\_logs):

Call sortLogs

Initialize issue\_map[MAX\_PRODUCTS][MAX\_LINES]

For each log in logs:

Populate issue\_map with issue codes by product ID and line ID

For each product\_id in range(MAX\_PRODUCTS):

For each line\_id in range(MAX\_LINES):

If issue\_map[product\_id][line\_id] != 0:

Print "Product ID: product\_id + 1, Line ID: line\_id + 1, Issue Code: issue\_map[product\_id][line\_id]"

Function findEarliestIssue(logs, num\_logs, product\_id):

Initialize earliest\_time to -1

For each log in logs:

If log.product\_id == product\_id:

If earliest\_time == -1 or compare(log, logs[earliest\_time]) < 0:

Set earliest\_time to current index

Return earliest\_time

Function summarizeIssues(logs, num\_logs):

Call sortLogs

Initialize issue\_count[MAX\_PRODUCTS]

For each log in logs:

Increment issue\_count for corresponding product ID

Print "Issue Summary by Product ID:"

For each product\_id in range(MAX\_PRODUCTS):

Print "Product ID: product\_id + 1, Number of Issues: issue\_count[product\_id]"

Print "Detailed Issue Summary:"

For each log in logs:

Print log details

Function main():

Define logs as array of LogEntry with sample data

Call reportIssues, findEarliestIssue, and summarizeIssues functions accordingly

**Full C code:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_PRODUCTS 10

#define MAX\_LINES 4

#define MAX\_EMPLOYEES 10

// Define a structure to represent log entry

typedef struct {

    int product\_id;

    int line\_id;

    int issue\_code;

    int day;

    int month;

    int year;

    int hour;

    int minute;

    char employee[MAX\_EMPLOYEES][20];

    char issue\_description[50];

    char resolution\_description[50];

} LogEntry;

// Compare function for merge sort

int compare(const void \*a, const void \*b) {

    const LogEntry \*entry1 = (const LogEntry \*)a;

    const LogEntry \*entry2 = (const LogEntry \*)b;

    // Compare dates

    if (entry1->year != entry2->year) return entry1->year - entry2->year;

    if (entry1->month != entry2->month) return entry1->month - entry2->month;

    if (entry1->day != entry2->day) return entry1->day - entry2->day;

    // Compare times

    if (entry1->hour != entry2->hour) return entry1->hour - entry2->hour;

    return entry1->minute - entry2->minute;

}

// Function to sort logs by date & time

void sortLogs(LogEntry \*logs, int num\_logs) {

    qsort(logs, num\_logs, sizeof(LogEntry), compare);

}

// Function to report issue codes by product ID and line ID, sorted by line and product ID

void reportIssues(LogEntry \*logs, int num\_logs) {

    // Sort logs by date & time

    sortLogs(logs, num\_logs);

    // Print the report header

    printf("Issue Reports by Product ID and Line ID:\n");

    // Initialize hash map for product\_id and line\_id

    int issue\_map[MAX\_PRODUCTS][MAX\_LINES] = {0};

    // Iterate through logs to populate the map

    for (int i = 0; i < num\_logs; i++) {

        issue\_map[logs[i].product\_id][logs[i].line\_id] = logs[i].issue\_code;

    }

    // Print the report

    for (int i = 0; i < MAX\_PRODUCTS; i++) {

        for (int j = 0; j < MAX\_LINES; j++) {

            if (issue\_map[i][j] != 0) {

                printf("Product ID: %d, Line ID: %d, Issue Code: %d\n",

                       i + 1, j + 1, issue\_map[i][j]);

            }

        }

    }

    printf("\n");

}

// Function to find the earliest occurrence of an issue code for a given product ID across all lines

int findEarliestIssue(LogEntry \*logs, int num\_logs, int product\_id) {

    int earliest\_time = -1;

    // Iterate through logs to find the earliest occurrence

    for (int i = 0; i < num\_logs; i++) {

        if (logs[i].product\_id == product\_id && (earliest\_time == -1 || compare(&logs[i], &logs[earliest\_time]) < 0)) {

            earliest\_time = i;

        }

    }

    return earliest\_time;

}

// Function to summarize the number of issues reported for each product across all lines

void summarizeIssues(LogEntry \*logs, int num\_logs) {

    // Sort logs by date & time

    sortLogs(logs, num\_logs);

    // Initialize hash map for product\_id and count of issues

    int issue\_count[MAX\_PRODUCTS] = {0};

    // Iterate through logs to count the occurrences of issues

    for (int i = 0; i < num\_logs; i++) {

        issue\_count[logs[i].product\_id]++;

    }

    // Print the summary header

    printf("Issue Summary by Product ID:\n");

    // Print the summary

    for (int i = 0; i < MAX\_PRODUCTS; i++) {

        printf("Product ID: %d, Number of Issues: %d\n", i + 1, issue\_count[i]);

    }

    printf("\n");

    // Print the detailed summary

    printf("Detailed Issue Summary:\n");

    for (int i = 0; i < num\_logs; i++) {

        printf("Product ID: %d, Issue Code: %d, Date: %02d/%02d/%02d, Time: %02d:%02d, Employees: ",

               logs[i].product\_id, logs[i].issue\_code, logs[i].day, logs[i].month, logs[i].year, logs[i].hour, logs[i].minute);

        for (int j = 0; j < MAX\_EMPLOYEES; j++) {

            if (strlen(logs[i].employee[j]) > 0) {

                printf("%s, ", logs[i].employee[j]);

            }

        }

        printf(", Issue Description: %s, Resolution Description: %s\n",

               logs[i].issue\_description, logs[i].resolution\_description);

    }

}

int main() {

    // Sample logs data

    LogEntry logs[] = {

        {1, 1, 101, 1, 4, 24, 12, 30, {"Max"}, "Leak", "Cap replaced"},  // product\_id, line\_id, issue\_code, day, month, year, hour, minute, employee, issue\_description, resolution\_description

        {2, 2, 102, 2, 4, 24, 10, 15, {"Ethan"}, "Faulty circuit", "Circuit fixed"},

        {3, 3, 103, 3, 4, 24, 8, 45, {"Max"}, "Broken seal", "Seal fixed"},

        {1, 2, 101, 4, 4, 24, 9, 0, {"Ethan"}, "Leak", "Cap tightened"},

        // More log entries can be added as data exapnds

    };

    int num\_logs = sizeof(logs) / sizeof(logs[0]);

    // Task 1: Sort the logs by date & time and display

    printf("Task 1: Sort the logs by date & time:\n");

    sortLogs(logs, num\_logs);

    for (int i = 0; i < num\_logs; i++) {

        printf("Date: %02d/%02d/%02d, Time: %02d:%02d, Product ID: %d, Line ID: %d, Issue Code: %d\n",

               logs[i].day, logs[i].month, logs[i].year, logs[i].hour, logs[i].minute,

               logs[i].product\_id, logs[i].line\_id, logs[i].issue\_code);

    }

    printf("\n");

    // Task 2: Report issue codes by product ID and line ID, sorted by line and product ID

    printf("Task 2: Report issue codes by product ID and line ID,:\n");

    reportIssues(logs, num\_logs);

    // Task 3: Find the earliest occurrence of an issue code for a given product ID

    int product\_id\_to\_search;

    printf("Task 3: Find the earliest occurrence of an issue code for a given product ID:\n");

    printf("Enter Product ID to search for earliest occurrence of an issue code: ");

    if (scanf("%d", &product\_id\_to\_search) != 1) {

        printf("Invalid input. Please enter a valid Product ID.\n");

        return 1;

    }

    int earliest\_index = findEarliestIssue(logs, num\_logs, product\_id\_to\_search);

    if (earliest\_index != -1) {

        printf("Earliest occurrence of an issue code for Product ID %d: Issue Code %d at Date: %02d/%02d/%02d Time: %02d:%02d\n",

               product\_id\_to\_search, logs[earliest\_index].issue\_code, logs[earliest\_index].day, logs[earliest\_index].month, logs[earliest\_index].year, logs[earliest\_index].hour, logs[earliest\_index].minute);

    } else {

        printf("No issue code found for Product ID %d\n", product\_id\_to\_search);

    }

    printf("\n");

    // Task 4: Summarize the number of issues reported for each product across all lines

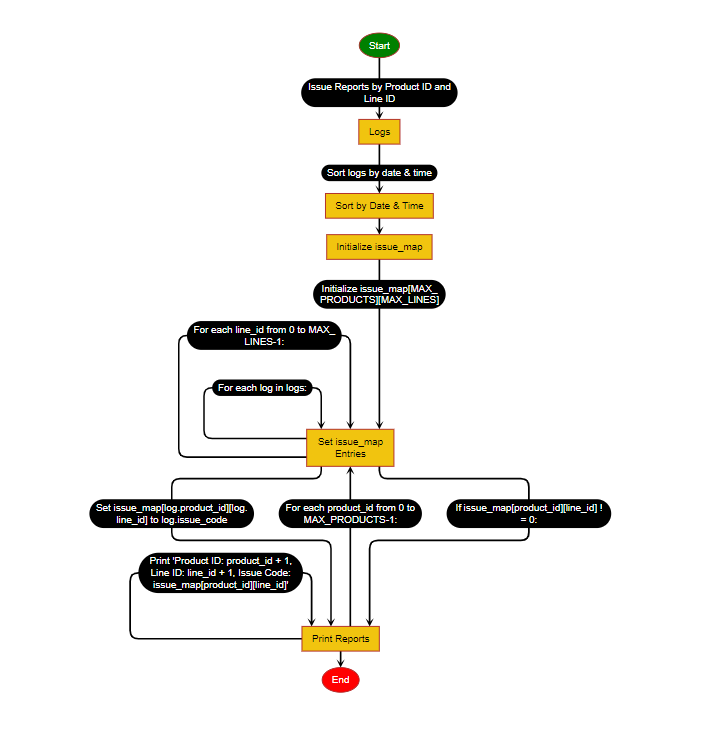
    printf("Task 4: Summarize the number of issues reported for each product across all lines:\n");

    summarizeIssues(logs, num\_logs);

    return 0;

}

**Flowchart for Task 2:**

****

**Design Process:**

This C code is designed to manage logs of issues in a manufacturing environment.

1. LogEntry Structure: The code defines a structure named LogEntry to represent each log entry. It contains information such as product ID, line ID, issue code, date, time, employee(s) involved, and descriptions of the issue and its resolution

2. Sorting Logs: The compare function compares log entries based on their dates and times. The sortLogs function sorts the log entries in chronological order using this comparison function.

3. Reporting Issues: The reportIssues function organizes and prints a report of issue codes grouped by product ID and line ID. It populates a hash map to keep track of issue codes for each product and line combination, then prints them out.

4. Finding Earliest Issue: The findEarliestIssue function searches for the earliest occurrence of an issue code for a given product ID across all lines. It returns the index of the earliest occurrence.

5. Summarizing Issues: The summarizeIssues function provides a summary of the number of issues reported for each product across all lines. It counts the occurrences of issues for each product and prints out the summary, along with detailed information about each issue.

6. Main Function: In the main function, sample logs are defined, and the tasks are executed sequentially:

- Sorting logs by date and time.

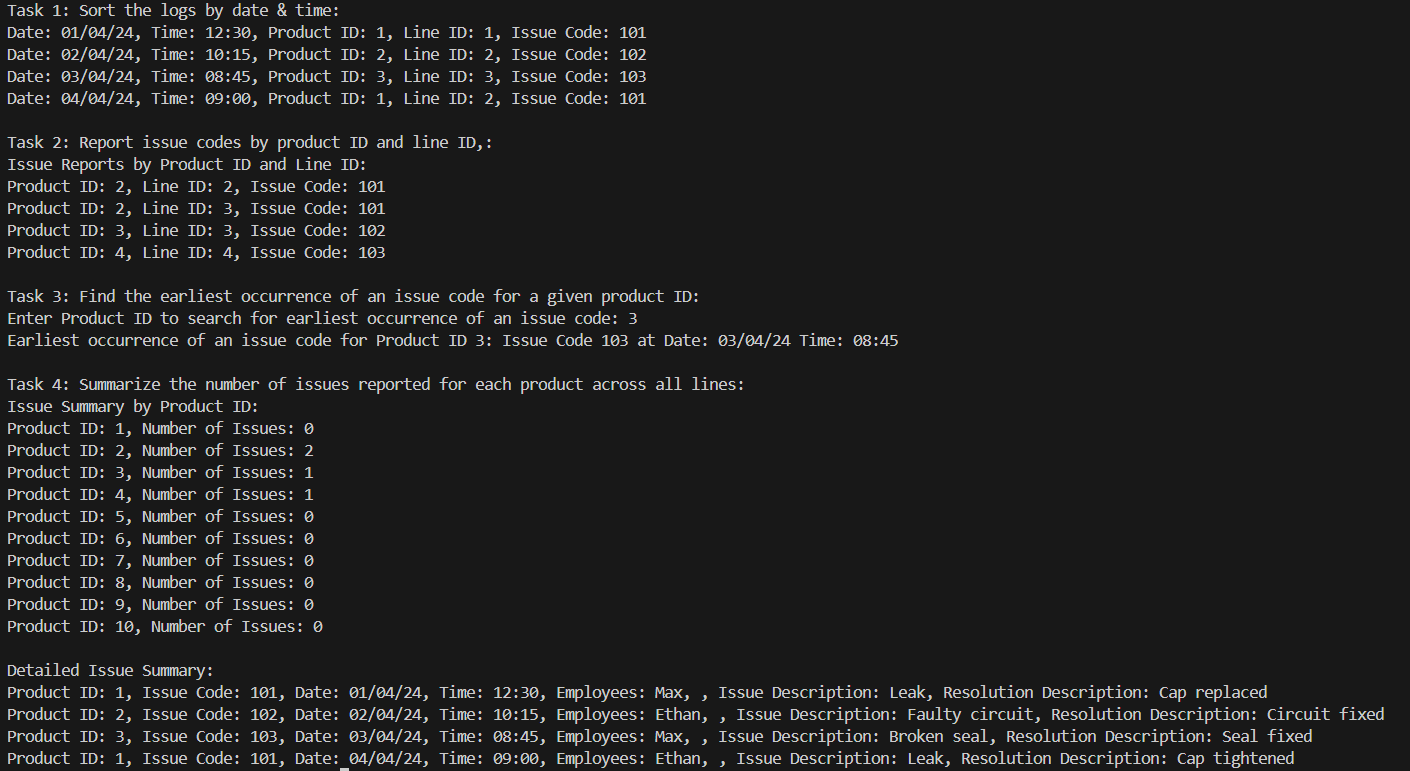
- Reporting issue codes by product ID and line ID.

- Finding the earliest occurrence of an issue code for a given product ID.

- Summarizing the number of issues reported for each product across all lines.

Overall, this code is structured to manage and analyze logs of issues in a manufacturing process, providing functionality to sort, report, find, and summarize issues efficiently.

**Output:**

****