**PROJECT REPORT**

***TU 856/1***

***CMPU 1001***

D22127506 Heeyeon Yoon

1. **Test data of each product line**

In this program, there are a total 4 production lines and each production line has 10 products. Therefore, the total number of products is 40. The data in the program consists of Line Code, Batch Code, Batch date & time, Product Id, Issue Code & description, Resolution code & description, Reporting employee id for each product. The same product ID can have different issue codes, and some products might belong to different production lines.

**Order**

: Line Code, Batch Code, Batch date & time, Product Id, Issue Code & description, Resolution code & description, Reporting employee id

**product line1**

1. 1, 330301, 211710, 1010, “011 - Engine Overheat”, “101 - Engine Coolant System Upgrade”, 2401
2. 1, 330301, 220710, 1010, "009 - Crack Found", "202 - Surface replacement", 2401
3. 1, 330301, 230810, 1010, "006 - Loose Mounting Bolts", "905 - Repair Mounting Bolts", 2401
4. 1, 330305, 210915, 1009, "002 - Electrical Short Circuit", "105 - Repair Wiring", 2405
5. 1, 330305, 220915, 1009, "009 - Crack Found", "105 - Repair Wiring", 2405
6. 1, 330305, 220915, 1009, "006 - Loose Mounting Bolts", "905 - Repair Mounting Bolts", 2405
7. 1, 330309, 231010, 1002, "003 - Fuel Pressure Variability", "109 - Repair Pumps", 2409
8. 1, 330309, 241111, 1002, "009 - Crack Found", "109 - Repair Pumps", 2409
9. 1, 330309, 251212, 1002, "006 - Loose Mounting Bolts", "905 - Repair Mounting Bolts", 2409
10. 2, 330302, 210710, 1012, "005 - Wing Surface Damage", "102 - Surface Repolishing", 2402

**product line2**

1. 2, 330302, 210810, 1012, "009 - Crack Found", "202 - Surface replacement", 2402
2. 2, 330302, 220820, 1012, "006 - Loose Mounting Bolts", "905 - Repair Mounting Bolts", 2402
3. 2, 330302, 230830, 1012, "001 - System Failure", "302 - System Check and Replacement", 2402
4. 2, 330306, 251350, 1008, "006 - Loose Mounting Bolts", "905 - Repair Mounting Bolts", 2406
5. 2, 330306, 231440, 1008, "009 - Torn Section Found", "206 - Seat Replacement", 2406
6. 2, 330306, 241020, 1008, "306 - Backrest won't go down", "905 - Repair Mounting Bolts", 2406
7. 2, 330310, 211110, 1005, "004 - Slide Stuck", "110 - Lubricate slide", 2410
8. 2, 330310, 221220, 1005, "006 - Loose Mounting Bolts", "905 - Repair Mounting Bolts", 2410
9. 2, 330310, 231330, 1005, "009 - Torn Section Found", "906 - Slide Surface Replacement", 2410
10. 1, 330309, 220107, 1002, "002 - Electrical Short Circuit", "105 - Repair Wiring", 2409

**product line3**

1. 2, 330306, 201350, 1008, "009 - Crack Found", "206 - Seat Replacement", 2406
2. 3, 330303, 201400, 1004, "006 - Loose Mounting Bolts", "905 - Repair Mounting Bolts", 2403
3. 3, 330303, 221640, 1004, "009 - Crack Found", "103 - Window Replacement", 2403
4. 3, 330303, 231300, 1004, "029 - Wind is Leaking", "103 - Window Replacement", 2403
5. 3, 330303, 201240, 1004, "039 - Cover won't go down", "113 - Cover Replacement", 2403
6. 3, 330311, 221030, 1013, "001 - System Failure", "302 - System Check and Replacement", 2411
7. 3, 330311, 231030, 1013, "021 - Broken Strap", "322 - Mask Replacement", 2411
8. 3, 330311, 241130, 1013, "031 - Lack of the Mask", "312 - Produce more mask", 2411
9. 3, 330311, 251131, 1013, "041 - Dirt on the Mask", "322 - Mask Replacement", 2411
10. 4, 330304, 251325, 1001, "010 - Landing Tire Wear", "104 - Tire Replacement", 2404

**product line4**

1. 4, 330304, 200845, 1001, "110 - Flat Tire", "104 - Tire Replacement", 2404
2. 4, 330304, 220910, 1001, "009 - Crack Found", "104 - Tire Replacement", 2404
3. 4, 330304, 241030, 1001, "006 - Loose Mounting Bolts", "905 - Repair Mounting Bolts", 2404
4. 4, 330308, 230900, 1007, "008 - Gear Stuck Open", "108 - Repair Mechanism", 2408
5. 4, 330308, 241000, 1007, "001 - System Failure", "118 - System Check", 2408
6. 4, 330308, 251100, 1007, "006 - Loose Mounting Bolts", "905 - Repair Mounting Bolts", 2408
7. 4, 330312, 211530, 1011, "009 - Crack found", "112 - Replace Flooring Material", 2412
8. 4, 330312, 220930, 1011, "006 - Loose Mounting Bolts", "905 - Repair Mounting Bolts", 2412
9. 4, 330312, 230830, 1011, "939 - Dirt Section found", "333 - Clean", 2412
10. 3, 330311, 221445, 1013, "101 - Torn section found", "111 - Mask System Check and Replacement", 2411
11. **Pseudocode for Task 1-4**
12. **These lines of pseudocode will be duplicated for all the tasks (1-4)**

// Define constants

DEFINE NUM\_LINE as 4

DEFINE NUM\_PRODUCT as 10

DEFINE LENGTH as 100

DEFINE MAX\_LOGS 40

// Struct Templates

STRUCT logs\_data{line\_code, batch\_code, date\_time, prod\_id, issue, resol, employee\_id}

STRUCT logs\_data all\_log[MAX\_LOGS]

1. **Pseudocode for Task 1**

: Make the production line logs are ordered by date & time using Merge Sort and display each line in Product id, Issue code, date & time order.

// Define function prototypes

FUNCTION mergeSort

FUNCTION merge

// Main function

FUNCTION main():

// make array of pointers to store pointers to each production line

struct logs\_data \*lines[] = {prod\_line1, prod\_line2, prod\_line3, prod\_line4}

// Call the function to order the product logs in date & time order

FOR i from 0 to NUM\_LINE:

FUNCTION CALL mergeSort(lines[i], 0, NUM\_PRODUCT -1)

ENDFOR

// Display the ordered logs

FOR i from 0 to NUM\_LINE:

FOR j from 0 to NUM\_PRODUCT:

PRINT lines[i][j].prod\_id, lines[i][j].issue\_code\_des, lines[i][j].date\_time

ENDFOR

ENDFOR

// Merge sort function

FUNCTION mergeSort (\*arr, left, right):

IF left < right:

mid = (left + right) / 2

FUNCTION CALL mergeSort(arr,left,mid)

FUNCTION CALL mergeSort(arr,mid+1,right)

FUNCTION CALL merge(arr,left,mid,right)

ENDIF

ENDFUNCTION

// Merge function

FUNCTION merge(\*arr, left, mid, right):

num1 = mid – left + 1, num2 = right – mid

// Create temporary arrays

logs\_data L[num1], R[num2]

// Copy data to temporary arrays

FOR i from 0 to num1:

L[i] = array[left + i]

ENDFOR

FOR j from 0 to num2:

R[j] = array[mid + 1 + j]

ENDFOR

int i, j = 0,

int k = left

WHILE i < num1 AND j < num2:

// Store the elements from the left array into the entire array

IF L[i].date\_time <= R[j].date\_time:

array[k] = L[i]

i ++

ENDIF

ELSE:

array[k] = R[j]

j ++

k ++

ENDWHILE

// Copy remaining elements of L[] and R[] to main array

WHILE i < num1 DO:

array[k] = L[i]

i ++, k++

ENDWHILE

WHILE j < num2 DO:

array[k] = R[j]

j ++, k++

ENDWHILE

ENDFUNCTION

1. **Pseudocode for Task 2**

: Make a single list to report issue codes by product Id and line Id for all production lines using Circular Queue functions (initializeQueue, enQueue, printIssueCode)

// Struct for the Circular Queue (Nested structure)

STRUCT CircularQueue{int head, tail, size, struct logs\_data data[MAX\_LOGS]}

// Main function

FUNCTION main():

// Create a circular queue

STRUCT CircularQueue queue

FUNCTION CALL initializeQueue(&queue)

// Combine all data into the circular queue

FOR i from 0 to the number of NUM\_PRODUCT:

FUNCTION CALL enQueue(&queue, prod\_line1[i])

FUNCTION CALL enQueue(&queue, prod\_line2[i])

FUNCTION CALL enQueue(&queue, prod\_line3[i])

FUNCTION CALL enQueue(&queue, prod\_line4[i])

ENDFOR

// Display a single report

FUNCTION CALL printIssueCode(&queue)

ENDFUNCTION

// Function to initialize Circular Queue

FUNCTION initializeQueue( \*queue):

queue->head = -1

queue->tail = -1

queue->size = 0

ENDFUNCTION

// Function to enqueue an element into Circular Queue

FUNCTION enQueue(\*queue, prod\_data):

IF (\*queue).size == 0:

// set the head to 0 to start from the beginning

queue -> head = 0

ENDIF

queue -> tail = (queue->tail + 1) % (MAX\_LOGS)

queue -> data[queue->tail] = prod\_data

queue -> size++

ENDFUNCTION

// Function to print issue codes by Product ID order and Line Code order

FUNCTION printIssueCode(\*queue):

// Create an array to store issue codes based on product ID and Line code order

STRUCT logs\_data issueCodes[MAX\_LOGS]

// Store issue codes in the array

i = queue->head

count = queue->size

WHILE count>0:

issueCodes[index++] = queue->data[i]

i = (i+1) % (MAX\_LOGS)

count - -

ENDWHILE

// Sort the array based on product ID

FOR i from 0 to the number of MAX\_LOGS:

FOR j from 0 to the number of MAX\_LOGS:

IF issueCodes[i].prod\_id > issueCodes[j].prod\_id

STRUCT logs\_data temp = issueCodes[i]

issueCodes[i] = issueCodes[j]

issueCides[j] = temp

ENDIF

ENDFOR

ENDFOR

// Print the sorted issue codes by product ID order

FOR i from 0 to the number of MAX\_LOGS:

PRINT issueCodes[i].prod\_id, issueCodes[i].issue

ENDFOR

// Sort the array based on Line code

FOR i from 0 to the number of MAX\_LOGS:

FOR j from 0 to the number of MAX\_LOGS:

IF issueCodes[i].line\_code > issueCodes[j].line\_code

STRUCT logs\_data temp = issueCodes[i]

issueCodes[i] = issueCodes[j]

issueCides[j] = temp

ENDIF

ENDFOR

ENDFOR

// Print the sorted issue codes by Line code order

FOR i from 0 to the number of MAX\_LOGS:

PRINT issueCodes[i].line\_code, issueCodes[i].issue

ENDFOR

ENDFUNCTION

1. **Pseudocode for Task 3**

: Show the earliest occurrence of an issue code for a given product id across all production lines using function mergeSort, combine\_logs, and searchEarliestOccur for Binary Search

// Main function

FUNCTION main():

// make array of pointers to store pointers to each production line

struct logs\_data \*lines[] = {prod\_line1, prod\_line2, prod\_line3, prod\_line4}

// Combine logs from all production lines

total\_logs = 0

// Combine data of each production lines (Function calls)

FOR i from 0 to the number of NUM\_LINE:

FUNCTION CALL combine\_logs(lines[i], NUM\_PRODUCT, &total\_logs)

ENDFOR

PRINT “Enter product ID”

SCAN &input\_id

// Function call for searching the earliest occurence of issuce code for the given product id

FUNCTION CALL searchEarliestOccur(all\_log, MAX\_LOGS, input\_id)

ENDFUNCTION

// Function for copying all the logs data from each production lines to the single list

FUNCTION combine\_logs(\*prod\_logs, num\_logs, \*total\_logs):

FOR i from 0 to the number of num\_logs:

all\_logs[\*total\_logs] = prod\_logs[i]

\*total\_logs ++

ENDFOR

ENDFUNCTION

// Function for searching the earliest occurrence of an issue code for a given product id

FUNCTION searchEarliesOccur(\*logs, size, inp\_id):

// Sort the array logs for binary search

FUNCTION CALL mergeSort(logs,0,size-1)

earliestIndex = -1, low = 0, high = size-1

WHILE low <= high:

mid = low + (high-low) / 2

IF logs[mid].prod\_id == inp\_id:

// Store the index of the earliest occurrence

earliestIndex = mid

// Continue search at the left side

high = mid – 1

ENDIF

ELSE IF logs[mid].prod\_id < inp\_id:

low = mid + 1

ENDELIF

ELSE:

high = mid – 1

ENEELSE

ENDWHILE

IF earliesIndex != -1:

// Traverse backwards from the earliestIndex to find the first occurrence

WHILE earliestIndex>0 AND logs[earliestIndex-1].prod\_id == inp\_id:

earliestIndex - -

ENDWHILE

PRINT logs[earliestIndex].line\_code, batch\_code, date\_time, issue

ENDIF

ENDFUNCTION

1. **Pseudocode for Task 4**

: Show all the issues for a given product id and count the number of issues using function combine\_logs, and searchIssues for Linear Search

// Main function

FUNCTION main():

// make array of pointers to store pointers to each production line

struct logs\_data \*lines[] = {prod\_line1, prod\_line2, prod\_line3, prod\_line4}

// Combine logs from all production lines

total\_logs = 0

// Combine data of each production lines (Function calls)

FOR i from 0 to the number of NUM\_LINE:

FUNCTION CALL combine\_logs(lines[i], NUM\_PRODUCT, &total\_logs)

ENDFOR

PRINT “Enter product ID”

SCAN &input\_id

// Function call for searching all the issues for the user entered product id and count them

FUNCTION CALL searchIssues(all\_log, MAX\_LOGS, input\_id)

ENDFUNCTION

// Function for copying all the logs data from each production lines to the single list

FUNCTION combine\_logs(\*prod\_logs, num\_logs, \*total\_logs):

FOR i from 0 to the number of num\_logs:

all\_logs[\*total\_logs] = prod\_logs[i]

\*total\_logs ++

ENDFOR

ENDFUNCTION

// Function for searching all the issues for given product id and count them

FUNCTION searchIssues(\*logs, size, inp\_id):

count = 0, searchPdID = -1

FOR i from 0 to the number of size:

IF logs[i].prod\_id == inp\_id:

// count all the issues and print them

count ++

PRINT count, logs[i].issue

searchPdID = i

ENDIF

ENDFOR

IF searchPdID == -1:

PRINT “Not found”

ENDIF

ELSE

PRINT “The total number of issues reported for a entered product id”, count

ENDELSE

ENDFUNCTION

all\_logs[\*total\_logs] = prod\_logs[i]

\*total\_logs ++

ENDFOR

1. **Flowchart for Task 2**
2. **Flowchart for main() Function**

****

1. **Flowchart for initializeQueue() Function**

****

1. **Flowchart for enQueue() Function**

****

1. **Flowchart for printIssueCode() Function**

****

1. **C code for Task 1 – 4**
2. **The common code for all Tasks**

: Header files of libraries, Symbolic names, Data of QA logs and products for each production lines, Structure templates, etc.

**(These lines of code are duplicated for all the tasks 1 – 4 )**

/\*

Title: Algorithm Design and Problem Solving(CMPU1001) Assignment

Author: D22127506 Heeyeon Yoon

Submission Date: 07/04/2024

Program Description: Performing 4 tasks using the data of products produced on 4 production lines of an engineering company

\*/

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** <string.h>

**#define** *NUM\_LINE* 4 // Number of production lines

**#define** *NUM\_PRODUCT* 10 // Number of products in each production line

**#define** *LENGTH* 100 // Length for characters

**#define** *MAX\_LOGS* 40 // Number of maximum QA logs

// Structure templates

// Data of QA logs

struct logs\_data

{

int *line\_code*;

int *batch\_code*;

int *date\_time*;

int *prod\_id*;

char *issue*[*LENGTH*];

char *resol*[*LENGTH*];

int *employee\_id*;

};

/\* Data consists of: Line code, Batch code, Batch Date & Time (order: date-hr-min), Product ID, Issue Code & description, Resolution code & description, Reporting employee ID \*/

// The data of 10 products in Production line 1

struct logs\_data *prod\_line1*[*NUM\_PRODUCT*] = {

{1, 330301, 211710, 1010, "011 - Engine Overheat", "101 - Engine Coolant System Upgrade", 2401},

{1, 330301, 220710, 1010, "009 - Crack Found", "202 - Surface replacement", 2401},

{1, 330301, 230810, 1010, "006 - Loose Mounting Bolts", "905 - Repair Mounting Bolts", 2401},

{1, 330305, 210915, 1009, "002 - Electrical Short Circuit", "105 - Repair Wiring", 2405},

{1, 330305, 220915, 1009, "009 - Crack Found", "105 - Repair Wiring", 2405},

{1, 330305, 220915, 1009, "006 - Loose Mounting Bolts", "905 - Repair Mounting Bolts", 2405},

{1, 330309, 231010, 1002, "003 - Fuel Pressure Variability", "109 - Repair Pumps", 2409},

{1, 330309, 241111, 1002, "009 - Crack Found", "109 - Repair Pumps", 2409},

{1, 330309, 251212, 1002, "006 - Loose Mounting Bolts", "905 - Repair Mounting Bolts", 2409},

{2, 330302, 210710, 1012, "005 - Wing Surface Damage", "102 - Surface Repolishing", 2402}

};

// The data of 10 products in Production line 2

struct logs\_data *prod\_line2*[*NUM\_PRODUCT*] = {

{2, 330302, 210810, 1012, "009 - Crack Found", "202 - Surface replacement", 2402},

{2, 330302, 220820, 1012, "006 - Loose Mounting Bolts", "905 - Repair Mounting Bolts", 2402},

{2, 330302, 230830, 1012, "001 - System Failure", "302 - System Check and Replacement", 2402},

{2, 330306, 251350, 1008, "006 - Loose Mounting Bolts", "905 - Repair Mounting Bolts", 2406},

{2, 330306, 231440, 1008, "009 - Torn Section Found", "206 - Seat Replacement", 2406},

{2, 330306, 241020, 1008, "306 - Backrest won't go down", "905 - Repair Mounting Bolts", 2406},

{2, 330310, 211110, 1005, "004 - Slide Stuck", "110 - Lubricate slide", 2410},

{2, 330310, 221220, 1005, "006 - Loose Mounting Bolts", "905 - Repair Mounting Bolts", 2410},

{2, 330310, 231330, 1005, "009 - Torn Section Found", "906 - Slide Surface Replacement", 2410},

{1, 330309, 220107, 1002, "002 - Electrical Short Circuit", "105 - Repair Wiring", 2409}

};

// The data of 10 products in Production line 3

struct logs\_data *prod\_line3*[*NUM\_PRODUCT*] = {

{2, 330306, 201350, 1008, "009 - Crack Found", "206 - Seat Replacement", 2406},

{3, 330303, 201400, 1004, "006 - Loose Mounting Bolts", "905 - Repair Mounting Bolts", 2403},

{3, 330303, 221640, 1004, "009 - Crack Found", "103 - Window Replacement", 2403},

{3, 330303, 231300, 1004, "029 - Wind is Leaking", "103 - Window Replacement", 2403},

{3, 330303, 201240, 1004, "039 - Cover won't go down", "113 - Cover Replacement", 2403},

{3, 330311, 221030, 1013, "001 - System Failure", "302 - System Check and Replacement", 2411},

{3, 330311, 231030, 1013, "021 - Broken Strap", "322 - Mask Replacement", 2411},

{3, 330311, 241130, 1013, "031 - Lack of the Mask", "312 - Produce more mask", 2411},

{3, 330311, 251131, 1013, "041 - Dirt on the Mask", "322 - Mask Replacement", 2411},

{4, 330304, 251325, 1001, "010 - Landing Tire Wear", "104 - Tire Replacement", 2404}

};

// The data of 10 products in Production line 4

struct logs\_data *prod\_line4*[*NUM\_PRODUCT*] = {

{4, 330304, 200845, 1001, "110 - Flat Tire", "104 - Tire Replacement", 2404},

{4, 330304, 220910, 1001, "009 - Crack Found", "104 - Tire Replacement", 2404},

{4, 330304, 241030, 1001, "006 - Loose Mounting Bolts", "905 - Repair Mounting Bolts", 2404},

{4, 330308, 230900, 1007, "008 - Gear Stuck Open", "108 - Repair Mechanism", 2408},

{4, 330308, 241000, 1007, "001 - System Failure", "118 - System Check", 2408},

{4, 330308, 251100, 1007, "006 - Loose Mounting Bolts", "905 - Repair Mounting Bolts", 2408},

{4, 330312, 211530, 1011, "009 - Crack found", "112 - Replace Flooring Material", 2412},

{4, 330312, 220930, 1011, "006 - Loose Mounting Bolts", "905 - Repair Mounting Bolts", 2412},

{4, 330312, 230830, 1011, "939 - Dirt Section found", "333 - Clean", 2412},

{3, 330311, 221445, 1003, "101 - Torn section found", "111 - Mask System Check and Replacement", 2411}

};

1. **C code for Task 1**

/\*

Task1: Display the sorted product logs in Product ID, Issue code, date & time order for each line ordered by date & time

The algorithm used for Task1: Merge Sort

\*/

// Function signatures

void *mergeSortByDate*(struct logs\_data \*, int, int);

void *mergeByDate*(struct logs\_data\*, int, int, int);

int *main*()

{

// Array of pointers to struct logs\_data to store pointers to each production line

struct logs\_data \**lines*[*NUM\_LINE*] = {*prod\_line1*, *prod\_line2*, *prod\_line3*, *prod\_line4*};

*printf*("\n\n------------------------------------------------------------------------------\n");

*printf*("\n\nDisplay the product logs in Product id, Issue code, date & time order \n");

*printf*("(Ordered by date & time)\n\n\n");

// Call the function to order the product logs in date & time order

**for** **(int** *i* **= 0;** *i* **<** *NUM\_LINE***;** *i***++)**

{

*mergeSortByDate*(*lines*[*i*], 0, *NUM\_PRODUCT* - 1);

}

// Display the Ordered logs

**for** **(int** *i* **= 0;** *i* **<** *NUM\_LINE***;** *i***++)**

{

*printf*("Production Line *%d*:\n", *i* + 1);

*printf*("\nDate & time\tProduct ID\tIssue Code & Description\n\n");

**for** **(int** *j* **= 0;** *j* **<** *NUM\_PRODUCT***;** *j***++)**

{

*printf*("*%d* \t*%d* \t\t*%s*\n", *lines*[*i*][*j*].*date\_time*, *lines*[*i*][*j*].*prod\_id*, *lines*[*i*][*j*].*issue*);

}

*printf*("\n------------------------------------------------------------------------------\n\n");

}

**return** 0;

}

// Function for ordering the production line logs by date & time using Merge sort

void *mergeSortByDate*(struct logs\_data \**arr*, int *left*, int *right*)

{

// Check if the size of array is bigger than one

**if** **(***left* **<** *right***)**

{

// Calculate the middle index

int *mid* = *left* + **(***right* **-** *left***)** / 2;

// Sort the first half of the array

*mergeSortByDate*(*arr*, *left*, *mid*);

// Sort the right half of the array

*mergeSortByDate*(*arr*, *mid* + 1, *right*);

// Merge the sorted halves

*mergeByDate*(*arr*, *left*, *mid*, *right*);

}

}

// Function to merge two sorted subarrays into one array by date & time order

void *mergeByDate*(struct logs\_data \**arr*, int *left*, int *mid*, int *right*)

{

// Calculate the sizes of the left and right subarrays

int *n1* = *mid* - *left* + 1;

int *n2* = *right* - *mid*;

// Make the temporary arrays to hold the left and right subarrays

struct logs\_data *L*[*n1*], *R*[*n2*];

// Copy data from the original(main) array to the left array (L)

**for** **(int** *i* **= 0;** *i* **<** *n1***;** *i***++)**

{

*L*[*i*] = *arr*[*left* + *i*];

}

// Copy data from the original(main) array to the right array (R)

**for** **(int** *j* **= 0;** *j* **<** *n2***;** *j***++)**

{

*R*[*j*] = *arr*[*mid* + 1 + *j*];

}

// Initialize indices for merging process

int *i* = 0;

int *j* = 0;

int *k* = *left*; // Index for the merged array

**while** **(***i* **<** *n1* **&&** *j* **<** *n2***)**

{

// Check if the elements of date\_time in the left side of array is faster(smaller) than the elements in the right side of array

**if** **(***L***[***i***].***date\_time* **<=** *R***[***j***].***date\_time***)**

{

// Store the element from the left subarray into the main array

*arr*[*k*] = *L*[*i*];

*i*++;

} **else**

{

// Store the element from the right subarray into the main array

*arr*[*k*] = *R*[*j*];

*j*++;

}

// increase k (move k to right side)

*k*++;

}

**while** **(***i* **<** *n1***)**

{

// copy any remaining elements from the left subarray to the main array

*arr*[*k*] = *L*[*i*];

*i*++;

*k*++;

}

**while** **(***j* **<** *n2***)**

{

// copy any remaining elements from the right subarray to the main array

*arr*[*k*] = *R*[*j*];

*j*++;

*k*++;

}

}

1. **C code for Task 2**

/\*

Task2: Display two single lists to report issue codes by each product Id and Line code order

The algorithm used for Task2: Circular Queue

\*/

// Struct for the Circular Queue

struct CircularQueue

{

int *head*, *tail*, *size*;

// nested structure

struct logs\_data *c\_array*[*MAX\_LOGS*];

};

// Function signatures

void *initializeQueue*(struct CircularQueue \*);

void *enQueue*(struct CircularQueue \*, struct *logs\_data*);

void *printIssueCode*(struct CircularQueue \*);

int *main*()

{

// Array of pointers to struct logs\_data to store pointers to each production line

struct logs\_data \**lines*[*NUM\_LINE*] = {*prod\_line1*, *prod\_line2*, *prod\_line3*, *prod\_line4*};

// create a circular queue

struct CircularQueue *queue*;

*initializeQueue*(&*queue*);

// Enqueue all data into the circular queue

**for** **(int** *i* **= 0;** *i* **<** *NUM\_PRODUCT***;** *i***++)**

{

*enQueue*(&*queue*, *prod\_line1*[*i*]);

*enQueue*(&*queue*, *prod\_line2*[*i*]);

*enQueue*(&*queue*, *prod\_line3*[*i*]);

*enQueue*(&*queue*, *prod\_line4*[*i*]);

}

// Display a single list to report issue codes by product Id and Line code

*printf*("\n\n---------------------------------------------------------------------------------------------\n");

*printf*("\n\nDisplay two single lists to report issue codes by each product Id order and Line code order\n");

*printIssueCode*(&*queue*);

**return** 0;

}

// Function to initialize Circular Queue

void *initializeQueue*(struct CircularQueue \**queue*)

{

// initialize head, tail pointer to indicate the queue is empty

*queue*->*head* = -1;

*queue*->*tail* = -1;

// initialize the size of the queue to 0 since it’s empty

*queue*->*size* = 0;

}

// Function to enqueue an element into Circular Queue

void *enQueue*(struct CircularQueue \**queue*, struct logs\_data *prod\_data*)

{

**if** **(***queue***->***size* **== 0)**

{

// If the queue is empty, set the head to 0 to start from the beginning

*queue*->*head* = 0;

}

// Calculate the position where the new element will be added in a queue

*queue*->*tail* = **(***queue***->***tail* **+ 1)** % **(***MAX\_LOGS***)**;

// Add the new data to the queue in c\_array

*queue*->*c\_array*[*queue*->*tail*] = *prod\_data*;

// Increment the size of the queue due to the addition of a new data

*queue*->*size*++;

}

// Function to print issue codes by Product ID order and Line Code order

void *printIssueCode*(struct CircularQueue \**queue*)

{

*printf*("\n\n --- Issue codes by Product ID order ---\n\n");

// Create an array to store issue codes based on product ID and line code order

struct logs\_data *issueCodes*[*MAX\_LOGS*];

int *index* = 0;

// Store issue codes in the array

int *i* = *queue*->*head*;

int *count* = *queue*->*size*;

**while** **(***count* **> 0)**

{

// Retreive issue code at position i and store it into array of issueCodes

*issueCodes*[*index*++] = *queue*->*c\_array*[*i*];

// Move to the next position in the queue

*i* = **(***i***+1)** % **(***MAX\_LOGS***)**;

*count*--;

}

// Sort the array based on product ID

**for** **(int** *i* **= 0;** *i* **< (***MAX\_LOGS***);** *i***++)**

{

**for** **(int** *j* **=** *i* **+ 1;** *j* **< (***MAX\_LOGS***);** *j***++)**

{

**if** **(***issueCodes***[***i***].***prod\_id* **>** *issueCodes***[***j***].***prod\_id***)**

{

struct logs\_data *temp* = *issueCodes*[*i*];

*issueCodes*[*i*] = *issueCodes*[*j*];

*issueCodes*[*j*] = *temp*;

}

}

}

// Print the sorted issue codes by product Id order

*printf*("\nProduct ID\tIssue Code & Description\n\n");

**for** **(int** *i* **= 0;** *i* **<** *MAX\_LOGS***;** *i***++)**

{

*printf*("*%d*\t\t*%s*\n", *issueCodes*[*i*].*prod\_id*, *issueCodes*[*i*].*issue*);

}

*printf*("\n\n------------------------------------------------------------------------------\n");

*printf*("\n\n --- Issue codes by Line Code order ---\n\n");

// Sort the array based on Line code

**for** **(int** *i* **= 0;** *i* **< (***MAX\_LOGS***);** *i***++)**

{

**for** **(int** *j* **=** *i* **+ 1;** *j* **< (***MAX\_LOGS***);** *j***++)**

{

**if** **(***issueCodes***[***i***].***line\_code* **>** *issueCodes***[***j***].***line\_code***)**

{

struct logs\_data *temp* = *issueCodes*[*i*];

*issueCodes*[*i*] = *issueCodes*[*j*];

*issueCodes*[*j*] = *temp*;

}

}

}

// Print the sorted issue codes line code order

*printf*("\nLine Code\tIssue Code & Description\n\n");

**for** **(int** *i* **= 0;** *i* **<** *MAX\_LOGS***;** *i***++)**

{

*printf*("*%d*\t\t*%s*\n", *issueCodes*[*i*].*line\_code*, *issueCodes*[*i*].*issue*);

}

*printf*("\n\n------------------------------------------------------------------------------\n");

}

1. **C code for Task 3**

/\*

Task3: Search the earliest occurrence of an issue code for a given product Id across all production lines

The algorithm used for Task3: Binary Search, Merge Sort

\*/

// Struct for a new single list to report issue codes by product Id and line Id for all production lines

struct logs\_data *all\_log*[*MAX\_LOGS*];

// Function signatures

void *combine\_logs*(struct logs\_data \*, int, int \*);

void *mergeSortByPID*(struct logs\_data \*, int, int);

void *mergeByPID*(struct logs\_data\*, int, int, int);

void *searchEarliestOccur*(struct logs\_data \*, int, int);

int *main*()

{

// Array of pointers to struct logs\_data to store pointers to each production line

struct logs\_data \**lines*[*NUM\_LINE*] = {*prod\_line1*, *prod\_line2*, *prod\_line3*, *prod\_line4*};

int *option\_selec* = 0;

int *input\_id*;

// Initialize the total number of logs to 0

// Will pass it by address through combine\_logs() function

int *total\_logs* = 0;

// Combine(Merge) logs of each production lines for task 3 and 4

**for** **(int** *i* **= 0;** *i* **<** *NUM\_LINE***;** *i***++)**

{

*combine\_logs*(*lines*[*i*], *NUM\_PRODUCT*, &*total\_logs*);

}

// Ask user to enter the product id

*printf*("\n\n------------------------------------------------------------------------------\n\n");

*printf*("\nEnter the product id: ");

*scanf*("*%d*", &*input\_id*);

// Function call for searching the earliest occurence of issuce code for the given product id

*searchEarliestOccur*(*all\_log*, *MAX\_LOGS*, *input\_id*);

**return** 0;

}

// Function for ordering the production line logs by Product ID using Merge sort

void *mergeSortByPID*(struct logs\_data \**arr*, int *left*, int *right*)

{

**if** **(***left* **<** *right***)** {

int *mid* = *left* + **(***right* **-** *left***)** / 2;

// Sort the first half side

*mergeSortByPID*(*arr*, *left*, *mid*);

// Sort the second half side

*mergeSortByPID*(*arr*, *mid* + 1, *right*);

// Merge the sorted halves

*mergeByPID*(*arr*, *left*, *mid*, *right*);

}

}

// Function to merge two sorted subarrays into one array by product id order

void *mergeByPID*(struct logs\_data \**arr*, int *left*, int *mid*, int *right*)

{

// Calculate the sizes of the left and right subarrays

int *n1* = *mid* - *left* + 1;

int *n2* = *right* - *mid*;

// Make temporary arrays to hold the subarrays

struct logs\_data *L*[*n1*], *R*[*n2*];

// Copy data from the original(main) array to the left array (L)

**for** **(int** *i* **= 0;** *i* **<** *n1***;** *i***++)**

{

*L*[*i*] = *arr*[*left* + *i*];

}

// Copy data from the original(main) array to the right array (R)

**for** **(int** *j* **= 0;** *j* **<** *n2***;** *j***++)**

{

*R*[*j*] = *arr*[*mid* + 1 + *j*];

}

int *i* = 0;

int *j* = 0;

int *k* = *left*;

**while** **(***i* **<** *n1* **&&** *j* **<** *n2***)**

{

// Check if the elements of prod\_id in the left side of array is smaller than the elements in the right side of array

**if** **(***L***[***i***].***prod\_id* **<=** *R***[***j***].***prod\_id***)**

{

// Store the elements from the left array into the entire array from the left

*arr*[*k*] = *L*[*i*];

*i*++;

} **else**

{

// Store the elements from the right array into the entire array from the left

*arr*[*k*] = *R*[*j*];

*j*++;

}

// increase k (move k to the right side)

*k*++;

}

**while** **(***i* **<** *n1***)**

{

// copy any remaining elements from the left subarray to the main array

*arr*[*k*] = *L*[*i*];

*i*++;

*k*++;

}

**while** **(***j* **<** *n2***)**

{

// copy any remaining elements from the right subarray to the main array

*arr*[*k*] = *R*[*j*];

*j*++;

*k*++;

}

}

// Function for copying all the logs data from each production lines to the single list

void *combine\_logs*(struct logs\_data \**prod\_logs*, int *num\_logs*, int \**total\_logs*)

{

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

{

*all\_log*[\**total\_logs*] = *prod\_logs*[*i*];

**(\****total\_logs***)**++;

}

}

// Function for searching the earliest occurrence of an issue code for a given product id

void *searchEarliestOccur*(struct logs\_data \**logs*, int *size*, int *inp\_prod\_id*)

{

// Sort the array logs for binary search

*mergeSortByPID*(*logs*, 0, *size* - 1);

// Initialize the earliestIndex to -1, in case the product id that user entered is not found in the logs

int *earliestIndex* = -1;

int *low* = 0;

int *high* = *size* -1;

**while** **(***low* **<=** *high***)**

{

int *mid* = *low* + **(***high* **-** *low***)** / 2;

**if** **(***logs***[***mid***].***prod\_id* **==** *inp\_prod\_id***)**

{

// Store the index of the earliest occurrence

*earliestIndex* = *mid*;

*high* = *mid* - 1; // continue search at the left side

}

**else** **if** **(***logs***[***mid***].***prod\_id* **<** *inp\_prod\_id***)**

{

*low* = *mid* + 1;

}

**else**

{

*high* = *mid* - 1;

}

}

// If the earliestIndex is not equal to -1, that means it has the issue code for user entered product id

**if** **(***earliestIndex* **!= -1)**

{

// Traverse backwards from the earliestIndex to find the first occurrence of the product id

**while** **(***earliestIndex* **> 0 &&** *logs***[***earliestIndex* **- 1].***prod\_id* **==** *inp\_prod\_id***)**

{

*earliestIndex*--;

}

*printf*("\n\nData of the earliest occurrence of issue code for product id *%d*\n\n", *inp\_prod\_id*);

*printf*("Line Code: *%d*\n", *logs*[*earliestIndex*].*line\_code*);

*printf*("Batch Code: *%d*\n", *logs*[*earliestIndex*].*batch\_code*);

*printf*("Date & Time: *%d*\n", *logs*[*earliestIndex*].*date\_time*);

*printf*("Issue: *%s*\n", *logs*[*earliestIndex*].*issue*);

*printf*("\n\n------------------------------------------------------------------------------\n\n");

}

// If the earliestIndex is equal to -1, that means it doesn't have the issue code for user entered product id

**else**

{

*printf*("\nIssue code for product id *%d* not found in the logs.\n", *inp\_prod\_id*);

}

}

1. **C code for Task 4**

/\*

Task4: Display all the issues and count the number of issues for a given product Id across all production lines

The algorithm used for Task4: Linear Search

\*/

// Struct for a new single list to report issue codes by product Id and line Id for all production lines

struct logs\_data *all\_log*[*MAX\_LOGS*];

// Function signatures

void *combine\_logs*(struct logs\_data \*, int, int \*);

void *searchIssues*(struct logs\_data \*, int, int);

int *main*()

{

// Array of pointers to struct logs\_data to store pointers to each production line

struct logs\_data \**lines*[*NUM\_LINE*] = {*prod\_line1*, *prod\_line2*, *prod\_line3*, *prod\_line4*};

int *option\_selec* = 0;

int *input\_id*;

// Initialize the total number of logs to 0

// Will pass it by address through combine\_logs() function

int *total\_logs* = 0;

// Combine(Merge) logs of each production lines for task 3 and 4

**for** **(int** *i* **= 0;** *i* **<** *NUM\_LINE***;** *i***++)**

{

*combine\_logs*(*lines*[*i*], *NUM\_PRODUCT*, &*total\_logs*);

}

// Ask user to enter the product id

*printf*("\n\n------------------------------------------------------------------------------\n\n");

*printf*("\nEnter the product id: ");

*scanf*("*%d*", &*input\_id*);

*printf*("\n");

// Function call for searching all the issues for the user entered product id and count them

*searchIssues*(*all\_log*, *MAX\_LOGS*, *input\_id*);

**return** 0;

}

// Function for copying all the logs data from each production lines to the single list

void *combine\_logs*(struct logs\_data \**prod\_logs*, int *num\_logs*, int \**total\_logs*)

{

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

{

*all\_log*[\**total\_logs*] = *prod\_logs*[*i*];

**(\****total\_logs***)**++;

}

}

// Function for searching all the issues for given product id and count them

void *searchIssues*(struct logs\_data \**logs*, int *size*, int *inp\_id*)

{

int *count* = 0;

// Initialize the searchPdID to -1, in case the product id that user entered is not found in the logs

int *searchPdID* = -1;

**for** **(int** *i* **= 0;** *i* **<** *size***;** *i***++)**

{

**if** **(***logs***[***i***].***prod\_id* **==** *inp\_id***)**

{

*count*++;

*printf*("\nIssue *%d*: *%s*\n", *count*, *logs*[*i*].*issue*);

*searchPdID* = *i*;

}

}

**if** **(***searchPdID* **== -1)**

{

*printf*("\nProduct id *%d* not found in the logs.\n", *inp\_id*);

}

**else**

{

*printf*("\n\nThe total number of issues reported for a product id *%d*: *%d*\n", *inp\_id*, *count*);

}

*printf*("\n\n------------------------------------------------------------------------------\n\n");

}

1. **The way my implementation meets the design requirements**
2. **Test plan for Task 1**

In Task1, there are huge amounts of data stored and need to sorted the data, so the running time of the algorithm should be O(NLog(N)). Therefore, I used Merge sort which is divide-and-conquer paradigm. In the code for Task1, Functions named ‘mergeSortByDate’ and ‘mergeByDate’ are used for the merge sort.

The mergeSortByDate function sorts an array based on dates and time. This function is called recursively. Firstly, it divides the array into two parts and then merges the two parts to create a sorted array. This process is executed only if the size of the array is greater than 1, which is checked using the line of code if (left < right). Therefore, if the size of the array is less than 1 (if the size of left is greater than or equal to the size of right), it is assumed to be already sorted, and further merge function calling is not needed.

The mergeByDate function divides the given array into two subarrays and merges them into one array sorted in date and time order. In this function, two temporary arrays ('L' and 'R') are created to respectively copy the left and right subarrays. Then, the loop with the condition 'while (i < n1 && j < n2)' is executed while there are elements remaining in either the left or right subarray to be copied. Within the loop, elements are merged and sorted in date and time order using the condition 'if (L[i].date\_time <= R[j].date\_time)'. Therefore, when one of the left or right subarrays is merged, the remaining elements from the other subarray are copied into the 'arr' array to complete the sorted array.

1. **Test plan for Task 2**

In Task 2, the algorithm's running time should be O(N). Therefore, I used a Circular Queue, which operates based on FIFO (First In First Out) principle. The basic operations of the Circular Queue algorithm include Enqueue, Dequeue, InitializeQueue, QueueFull, and QueueEmpty. However, in my case, I don’t need to delete items from the queue, and the array will not become full as the size of the array and the number of items are defined to be the same before the function is executed. Therefore, I didn’t use the Dequeue, QueueFull, and QueueEmpty functions in my code. Instead, only the 'initializeQueue', 'enQueue', and 'printIssueCode (the function for display the output)' functions are used for the code.

The 'initializeQueue' function initializes a Circular Queue by setting its head and tail pointers to -1 and its size to 0. This indicates the queue is empty.

The ‘enQueue’ function is used to add elements into the Circular Queue. In the enQueue function, if the queue is empty (size == 0), it sets the head to 0 to start from the beginning. Then, it updates the tail to the next position, wrapping around if necessary using this code ' (queue->tail + 1) % MAX\_LOGS' (If the queue's index exceeds the maximum size, the index wraps around to the beginning). It then adds the new element to the circular array at the tail position and increments the size of the queue.

The 'printIssueCode' function is designed to display issue codes sorted by Product ID order and Line Code order. Firstly, it creates an array 'issueCodes' to store issue codes based on product ID and line code order. Then, it iterates through the Circular Queue to fill this array with issue codes in the order they were enqueued. It then proceeds to sort and print the 'issueCodes' array based on Product ID using nested loops and elements swapping. Using the same procedure, it sorts and prints the 'issueCodes' array based on Line Code.

1. **Test plan for Task 3**

In Task3, there are huge amounts of data stored and need to sorted the data, so the running time of the algorithm should be O(Log(N)). Therefore, I used Binary Search and Merge Sort. Binary Search relies on a divide-and-conquer strategy to find a value within an already-sorted collection. Since it uses a sorted array, I employed the merge sort algorithm, which I also utilized in Task 1. For Task3, ‘combine\_logs’, ‘mergeSortByPID’, ‘mergeByPID’, and ‘searchEarliestOccir’ Functions are used in the code.

The 'combine\_logs' function is used to copy all the logs data from each production line for combining them into a single list.

The 'mergeSortByPID' and 'mergeByPID' functions are performed exactly the same way as in Task 1, but this time they are sorted by Product ID order instead of being sorted by Date and time order.

The 'searchEarliestOccur' function is used for searching the earliest occurrence of an issue code for a given product ID using binary search. At the beginning of this function, it calls the merge function as binary search requires a sorted array. Then it initializes the 'earliestIndex' to -1 in case the product Id entered by the user is not found in the logs. If found, it updates the 'earliestIndex' accordingly. After that, it utilizes the loop condition (low <= high) to search while the value of 'low' is less than or equal to the value of 'high'. During the search, if the product Id of the middle index is same as the user input product Id, it stores that middle index as the 'earliestIndex'. If the Id of the middle index is less than the input Id, it increases the 'low' index to be greater than the middle index, that enables the search to proceed to the right side. Otherwise, it decreases the 'high' index to be less than the middle index, and makes the search to proceed to the left side.

1. **Test plan for Task 4**

In Task4, the algorithm's running time should be O(N). Therefore, I used Linear Search, which is the most basic of search algorithms. This algorithm sequentially moves through the array looking for a matching value. In this code, ‘combine\_logs’, and ‘searchIssues’ Functions are used for Task 4.

The 'combine\_logs' function is the same function as the one used for Task3, and it is used to combine all the lines of code into one array.

The ‘searchIssues’ function displays all the issues and counts the number of them for a given product Id using Linear Search. Firstly, it initialize the 'searchPdID' to -1 in case the user entered product Id is not found in the logs. Within this function, a 'for' loop iterates from i=0 to the size of the array. Inside the loop, 'if (logs[i].prod\_id == inp\_id)' statements compare the user input Id with the product IDs in the array. If they are same, the function prints the issue code for the product Id and increases the count. Also, 'searchPdID' is assigned to the value of 'i', which is the latest occurrence.

**References**

GeeksforGeeks (2018). *Merge Sort - GeeksforGeeks*. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/merge-sort/.

GeeksforGeeks (2019). *Binary Search - GeeksforGeeks*. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/binary-search/.

Gupta, A. (2017). *Introduction to Circular Queue*. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/introduction-to-circular-queue/.