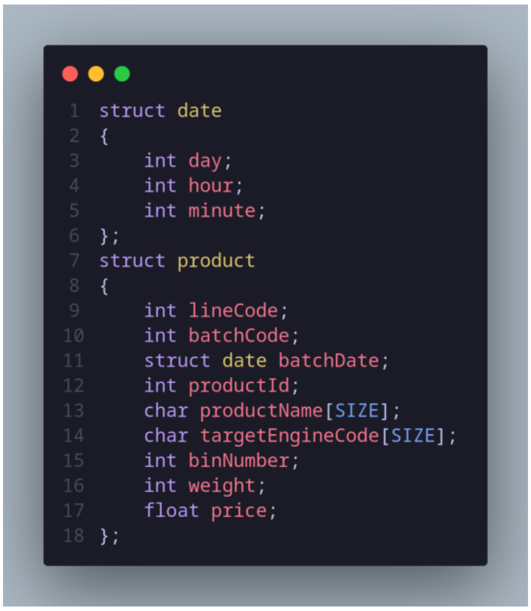
Neil Jiang - C24510496 – CMPU 1001 Project Report

Data Structure

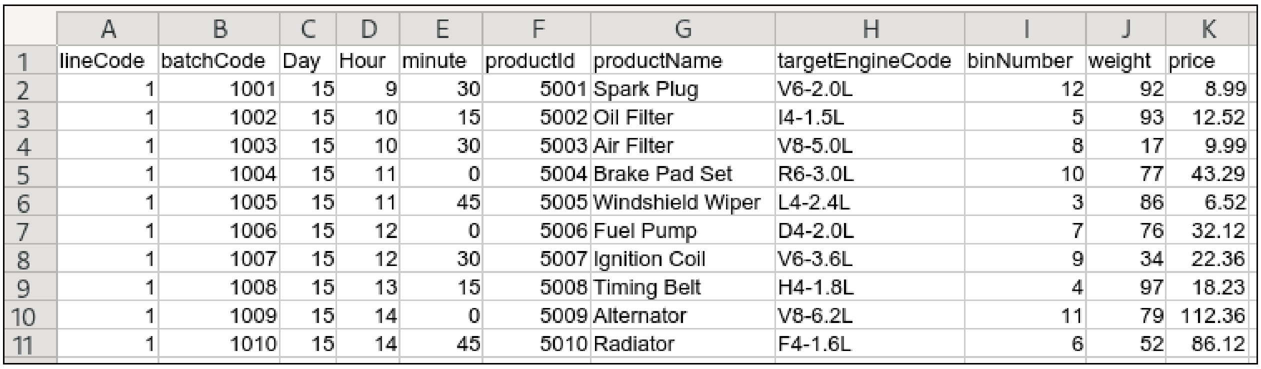


```c
struct date
{
    int day;
    int hour;
    int minute;
};
struct product
{
    int lineCode;
    int batchCode;
    struct date batchDate;
    int productId;
    char productName[SIZE];
    char targetEngineCode[SIZE];
    int binNumber;
    int weight;
    float price;
};
```

Test Data

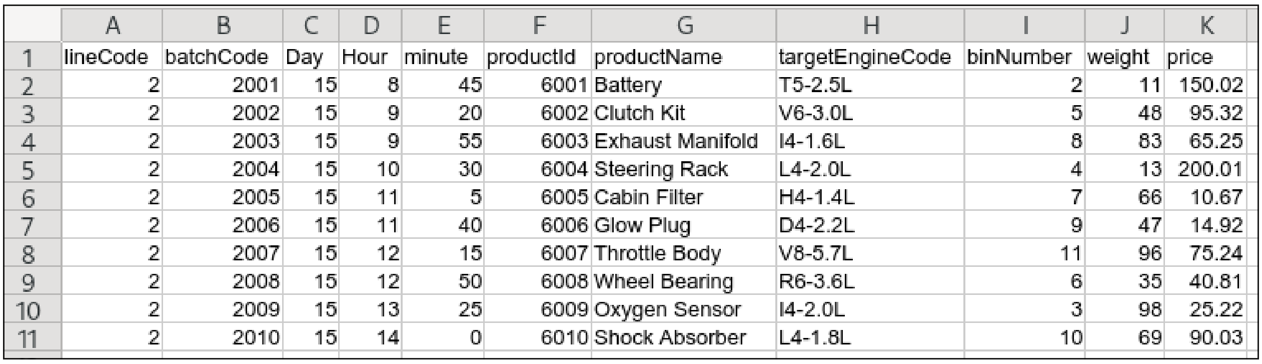
A CSV file was used to store the test data. Each line is then read using C code, and stored in its respective structure arrays.

Line 1:



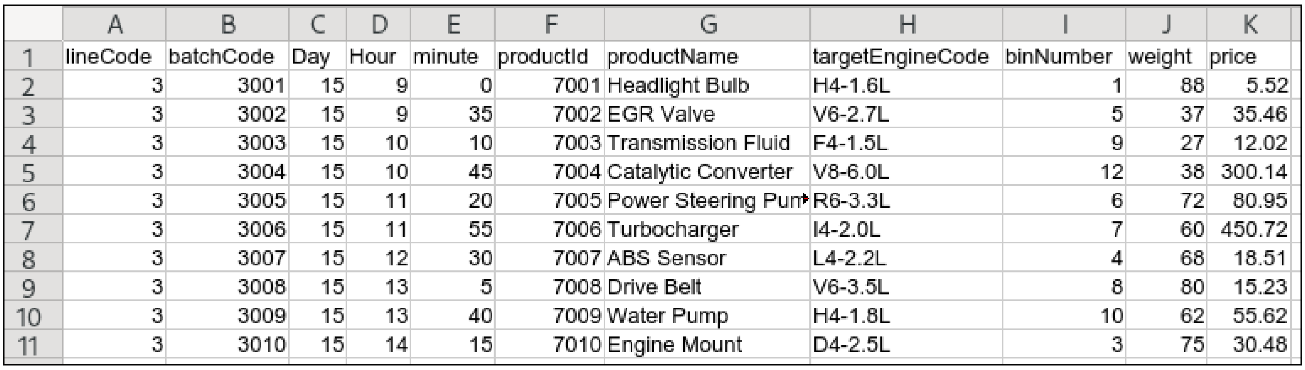
	A	B	C	D	E	F	G	H	I	J	K
1	lineCode	batchCode	Day	Hour	minute	productId	productName	targetEngineCode	binNumber	weight	price
2	1	1001	15	9	30	5001	Spark Plug	V6-2.0L	12	92	8.99
3	1	1002	15	10	15	5002	Oil Filter	I4-1.5L	5	93	12.52
4	1	1003	15	10	30	5003	Air Filter	V8-5.0L	8	17	9.99
5	1	1004	15	11	0	5004	Brake Pad Set	R6-3.0L	10	77	43.29
6	1	1005	15	11	45	5005	Windshield Wiper	L4-2.4L	3	86	6.52
7	1	1006	15	12	0	5006	Fuel Pump	D4-2.0L	7	76	32.12
8	1	1007	15	12	30	5007	Ignition Coil	V6-3.6L	9	34	22.36
9	1	1008	15	13	15	5008	Timing Belt	H4-1.8L	4	97	18.23
10	1	1009	15	14	0	5009	Alternator	V8-6.2L	11	79	112.36
11	1	1010	15	14	45	5010	Radiator	F4-1.6L	6	52	86.12

Line 2:



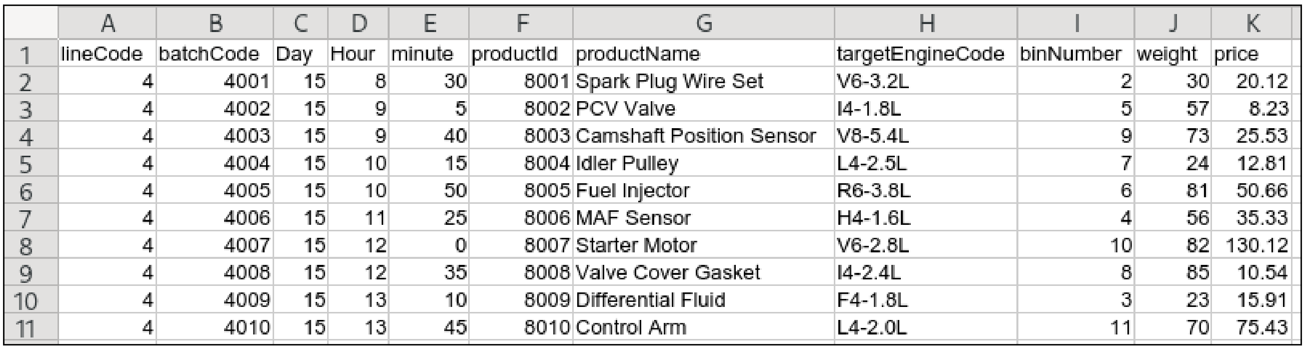
	A	B	C	D	E	F	G	H	I	J	K
1	lineCode	batchCode	Day	Hour	minute	productId	productName	targetEngineCode	binNumber	weight	price
2	2	2001	15	8	45	6001	Battery	T5-2.5L	2	11	150.02
3	2	2002	15	9	20	6002	Clutch Kit	V6-3.0L	5	48	95.32
4	2	2003	15	9	55	6003	Exhaust Manifold	I4-1.6L	8	83	65.25
5	2	2004	15	10	30	6004	Steering Rack	L4-2.0L	4	13	200.01
6	2	2005	15	11	5	6005	Cabin Filter	H4-1.4L	7	66	10.67
7	2	2006	15	11	40	6006	Glow Plug	D4-2.2L	9	47	14.92
8	2	2007	15	12	15	6007	Throttle Body	V8-5.7L	11	96	75.24
9	2	2008	15	12	50	6008	Wheel Bearing	R6-3.6L	6	35	40.81
10	2	2009	15	13	25	6009	Oxygen Sensor	I4-2.0L	3	98	25.22
11	2	2010	15	14	0	6010	Shock Absorber	L4-1.8L	10	69	90.03

Line 3:



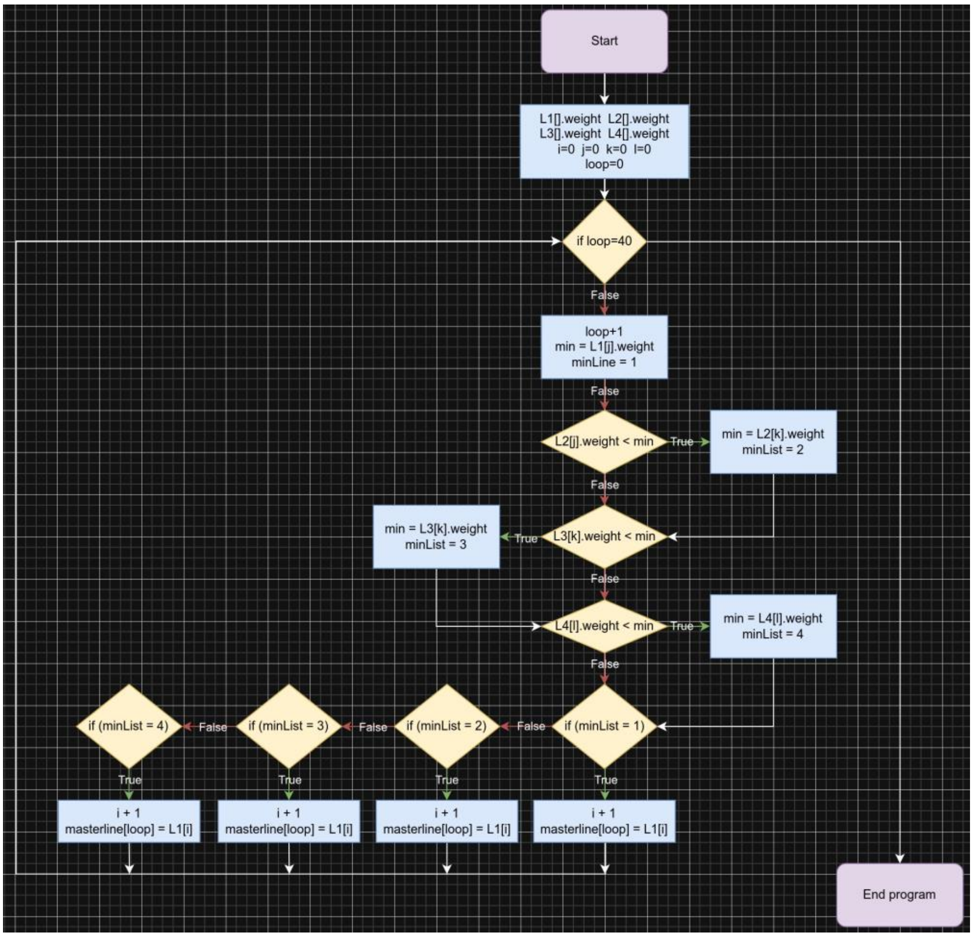
	A	B	C	D	E	F	G	H	I	J	K
1	lineCode	batchCode	Day	Hour	minute	productId	productName	targetEngineCode	binNumber	weight	price
2	3	3001	15	9	0	7001	Headlight Bulb	H4-1.6L	1	88	5.52
3	3	3002	15	9	35	7002	EGR Valve	V6-2.7L	5	37	35.46
4	3	3003	15	10	10	7003	Transmission Fluid	F4-1.5L	9	27	12.02
5	3	3004	15	10	45	7004	Catalytic Converter	V8-6.0L	12	38	300.14
6	3	3005	15	11	20	7005	Power Steering Pum	R6-3.3L	6	72	80.95
7	3	3006	15	11	55	7006	Turbocharger	I4-2.0L	7	60	450.72
8	3	3007	15	12	30	7007	ABS Sensor	L4-2.2L	4	68	18.51
9	3	3008	15	13	5	7008	Drive Belt	V6-3.5L	8	80	15.23
10	3	3009	15	13	40	7009	Water Pump	H4-1.8L	10	62	55.62
11	3	3010	15	14	15	7010	Engine Mount	D4-2.5L	3	75	30.48

Line 4:



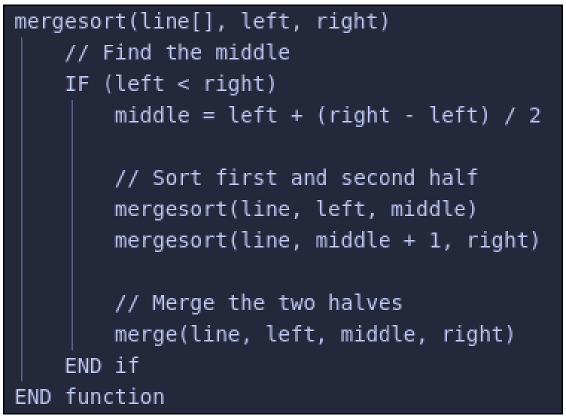
	A	B	C	D	E	F	G	H	I	J	K
1	lineCode	batchCode	Day	Hour	minute	productId	productName	targetEngineCode	binNumber	weight	price
2	4	4001	15	8	30	8001	Spark Plug Wire Set	V6-3.2L	2	30	20.12
3	4	4002	15	9	5	8002	PCV Valve	I4-1.8L	5	57	8.23
4	4	4003	15	9	40	8003	Camshaft Position Sensor	V8-5.4L	9	73	25.53
5	4	4004	15	10	15	8004	Idler Pulley	L4-2.5L	7	24	12.81
6	4	4005	15	10	50	8005	Fuel Injector	R6-3.8L	6	81	50.66
7	4	4006	15	11	25	8006	MAF Sensor	H4-1.6L	4	56	35.33
8	4	4007	15	12	0	8007	Starter Motor	V6-2.8L	10	82	130.12
9	4	4008	15	12	35	8008	Valve Cover Gasket	I4-2.4L	8	85	10.54
10	4	4009	15	13	10	8009	Differential Fluid	F4-1.8L	3	23	15.91
11	4	4010	15	13	45	8010	Control Arm	L4-2.0L	11	70	75.43

Flow Chart for Task 2



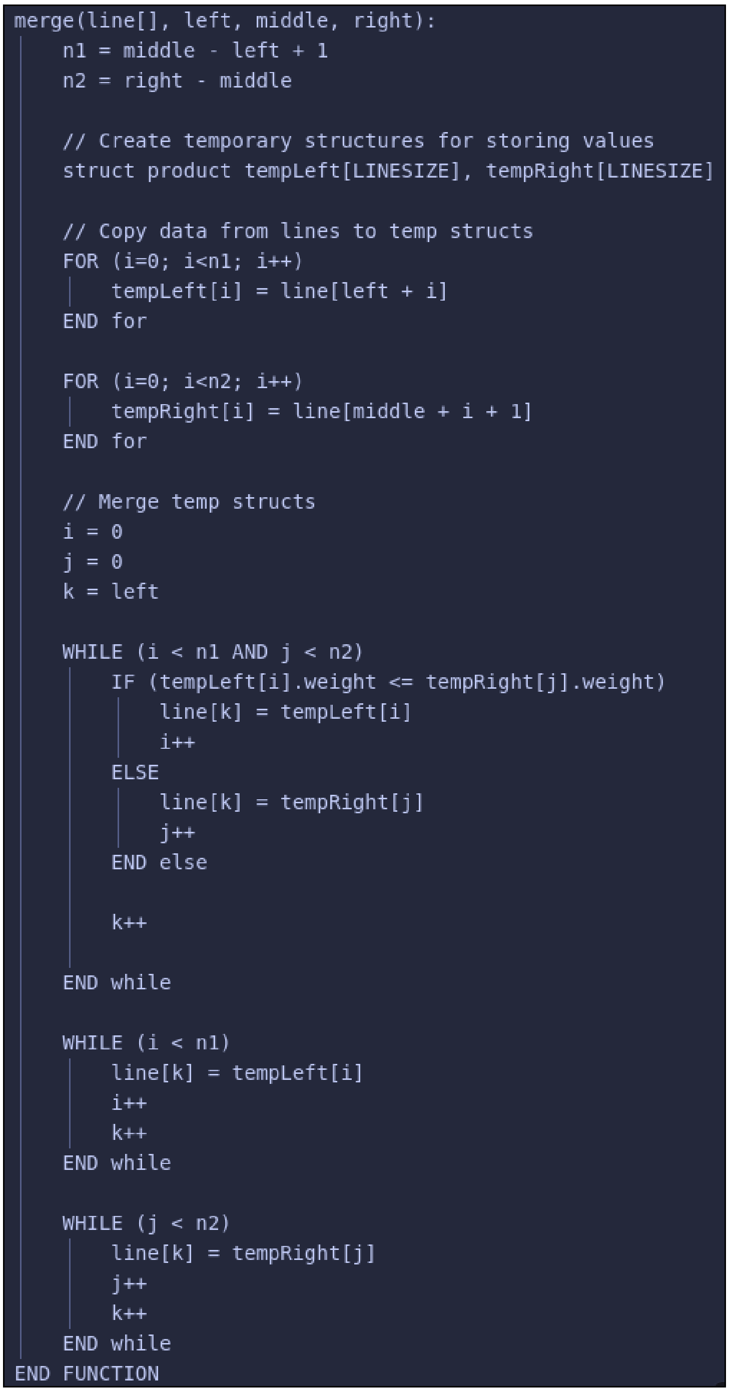
Task 1

For this task, I was asked to use an appropriate sorting algorithm to sort 4 structure arrays by their weight. The sorting algorithm must run with a time complexity of O(n log(n)) or better, so I decided to use merge sort. This is because merge sort has a worse case time complexity of O(n log(n)), which was perfect.



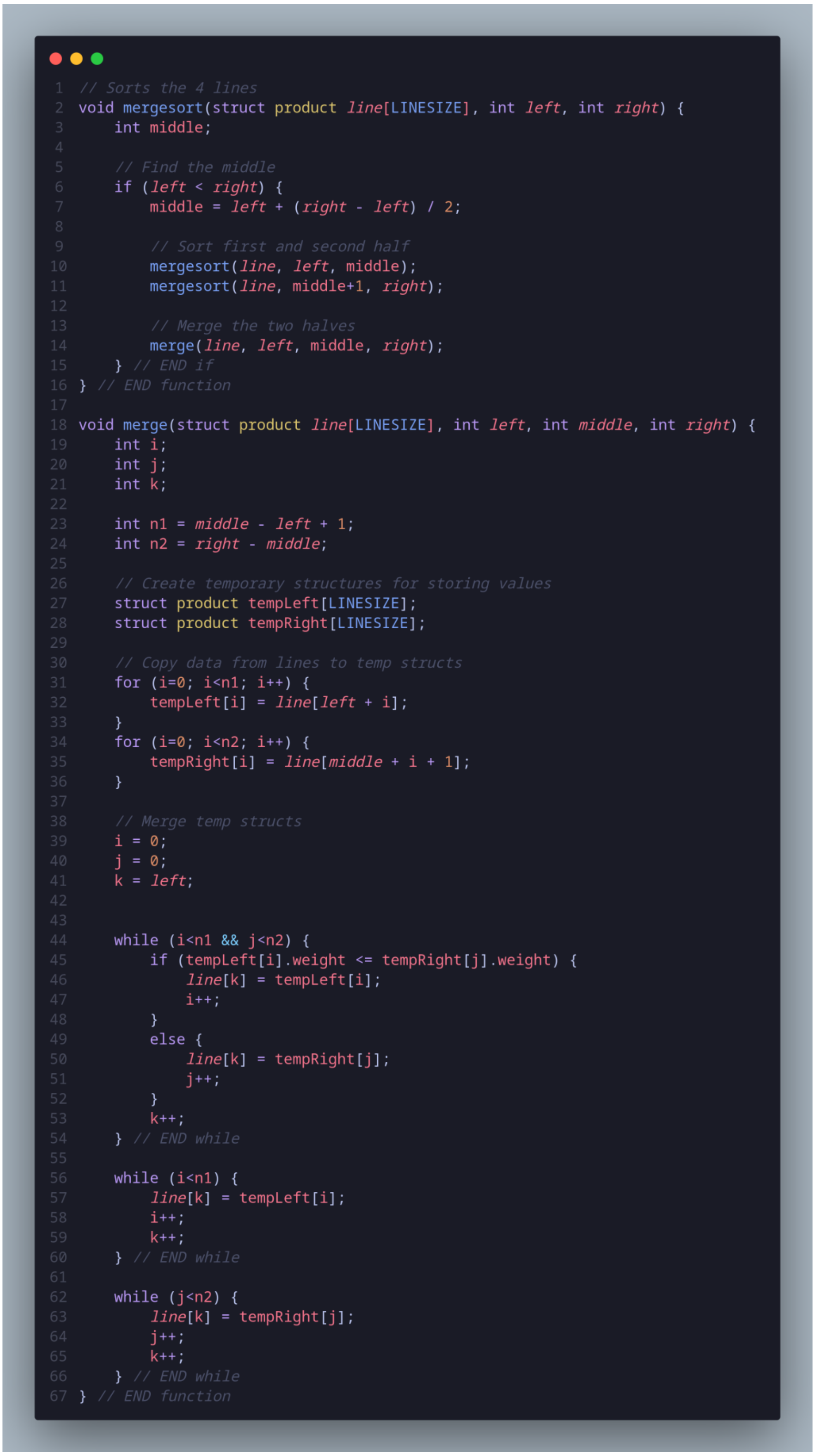
```
mergesort(line[], left, right)
    // Find the middle
    IF (left < right)
        middle = left + (right - left) / 2

        // Sort first and second half
        mergesort(line, left, middle)
        mergesort(line, middle + 1, right)

        // Merge the two halves
        merge(line, left, middle, right)
    END if
END function
```

Here is a picture of the pseudocode I used to design the algorithm:



```
merge(line[], left, middle, right):
    n1 = middle - left + 1
    n2 = right - middle

    // Create temporary structures for storing values
    struct product tempLeft[LINESIZE], tempRight[LINESIZE]

    // Copy data from lines to temp structs
    FOR (i=0; i<n1; i++)
        tempLeft[i] = line[left + i]
    END for

    FOR (i=0; i<n2; i++)
        tempRight[i] = line[middle + i + 1]
    END for

    // Merge temp structs
    i = 0
    j = 0
    k = left

    WHILE (i < n1 AND j < n2)
        IF (tempLeft[i].weight <= tempRight[j].weight)
            line[k] = tempLeft[i]
            i++
        ELSE
            line[k] = tempRight[j]
            j++
        END else

        k++

    END while

    WHILE (i < n1)
        line[k] = tempLeft[i]
        i++
        k++
    END while

    WHILE (j < n2)
        line[k] = tempRight[j]
        j++
        k++
    END while
END FUNCTION
```

Here is the algorithm implemented in C:



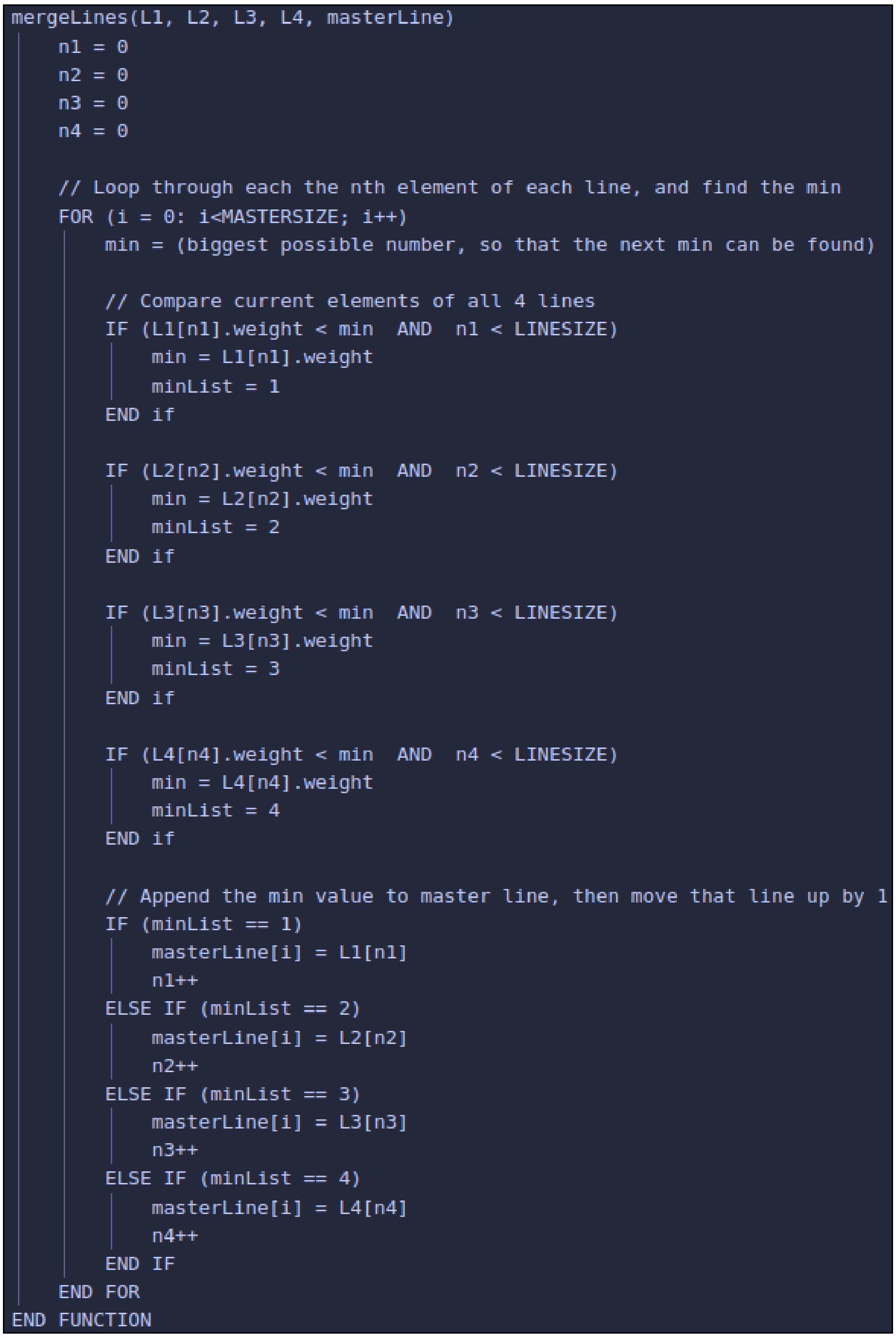
```c
// Sorts the 4 lines
void mergesort(struct product line[LINESIZE], int left, int right) {
    int middle;

    // Find the middle
    if (left < right) {
        middle = left + (right - left) / 2;

        // Sort first and second half
        mergesort(line, left, middle);
        mergesort(line, middle+1, right);

        // Merge the two halves
        merge(line, left, middle, right);
    } // END if
} // END function

void merge(struct product line[LINESIZE], int left, int middle, int right) {
    int i;
    int j;
    int k;

    int n1 = middle - left + 1;
    int n2 = right - middle;

    // Create temporary structures for storing values
    struct product tempLeft[LINESIZE];
    struct product tempRight[LINESIZE];

    // Copy data from lines to temp structs
    for (i=0; i<n1; i++) {
        tempLeft[i] = line[left + i];
    }
    for (i=0; i<n2; i++) {
        tempRight[i] = line[middle + i + 1];
    }

    // Merge temp structs
    i = 0;
    j = 0;
    k = left;


    while (i<n1 && j<n2) {
        if (tempLeft[i].weight <= tempRight[j].weight) {
            line[k] = tempLeft[i];
            i++;
        }
        else {
            line[k] = tempRight[j];
            j++;
        }
        k++;
    } // END while

    while (i<n1) {
        line[k] = tempLeft[i];
        i++;
        k++;
    } // END while

    while (j<n2) {
        line[k] = tempRight[j];
        j++;
        k++;
    } // END while
} // END function
```

Task 2

For this task, I had to design an algorithm to merge the 4 sorted structure arrays. The running speed of the algorithm must have a time complexity of O(n) or better. When I initially started designing this algorithm, I realised it was very similar to the “merge” part from merge sort. However, instead of checking and merging two arrays at a time, I expanded upon it to merge 4 arrays.

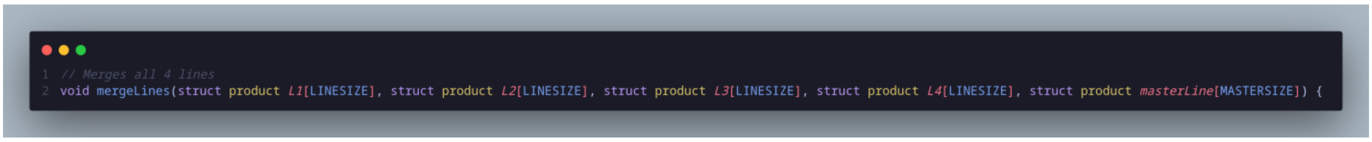
To explain how this algorithm works, it first goes through the first element of each of the lines, finding the smallest between them. Once that minimum value is found, it is appended to the “master line” (The master line is essentially a structure array that holds all the merged lines). Then, it finds the next minimum value between the lines, and appends that to the master line. This process is repeated until each product in the 4 lines has been appended to the master line. This algorithm has a wrose case time complexity of O(n), as it only uses one loop to go through each element.

Here is the pseudocode:

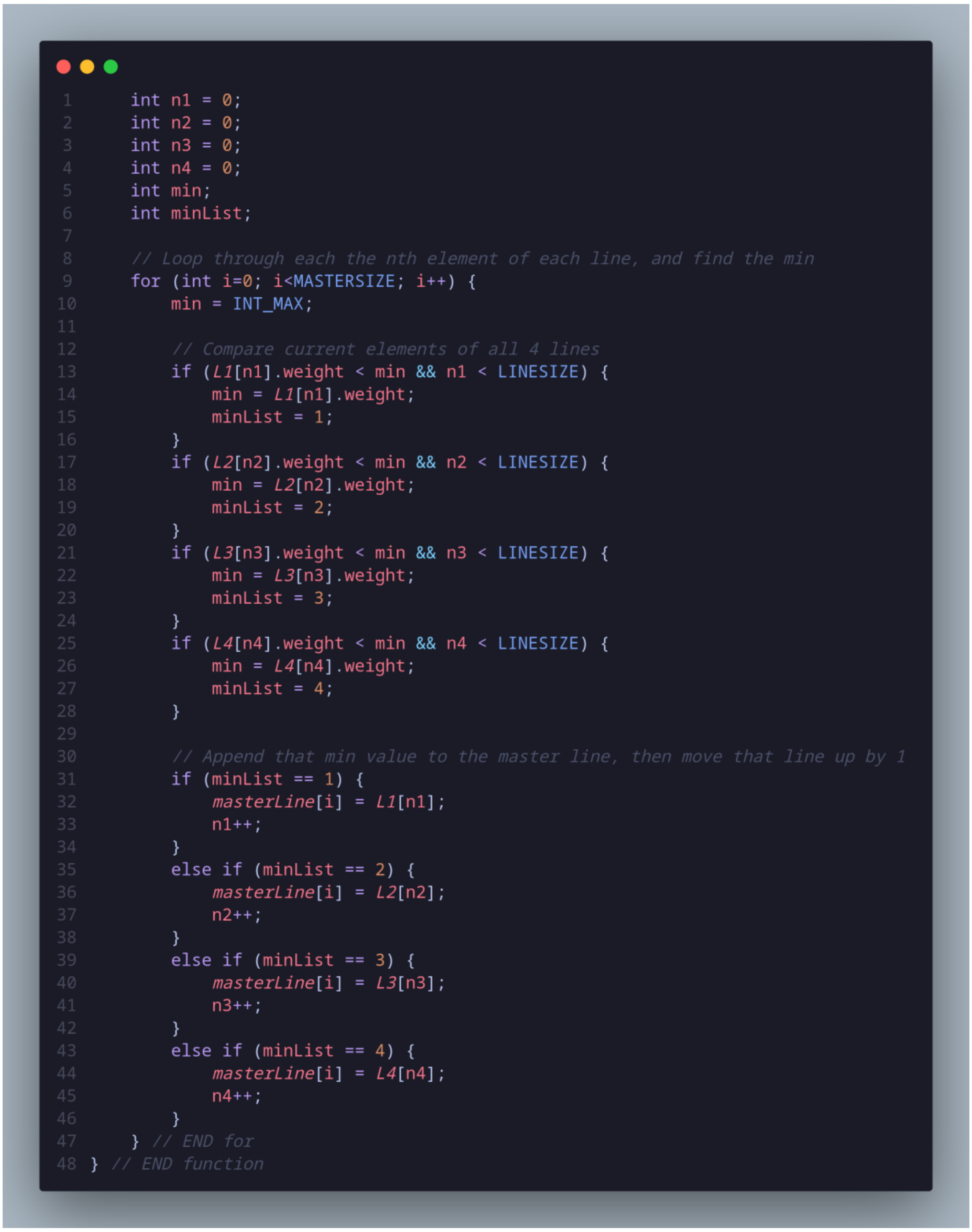


```
mergeLines(L1, L2, L3, L4, masterLine)
    n1 = 0
    n2 = 0
    n3 = 0
    n4 = 0

    // Loop through each the nth element of each line, and find the min
    FOR (i = 0: i<MASTERSIZE; i++)
        min = (biggest possible number, so that the next min can be found)

        // Compare current elements of all 4 lines
        IF (L1[n1].weight < min  AND  n1 < LINESIZE)
            min = L1[n1].weight
            minList = 1
        END if

        IF (L2[n2].weight < min  AND  n2 < LINESIZE)
            min = L2[n2].weight
            minList = 2
        END if

        IF (L3[n3].weight < min  AND  n3 < LINESIZE)
            min = L3[n3].weight
            minList = 3
        END if

        IF (L4[n4].weight < min  AND  n4 < LINESIZE)
            min = L4[n4].weight
            minList = 4
        END if

        // Append the min value to master line, then move that line up by 1
        IF (minList == 1)
            masterLine[i] = L1[n1]
            n1++
        ELSE IF (minList == 2)
            masterLine[i] = L2[n2]
            n2++
        ELSE IF (minList == 3)
            masterLine[i] = L3[n3]
            n3++
        ELSE IF (minList == 4)
            masterLine[i] = L4[n4]
            n4++
        END IF
    END FOR
END FUNCTION
```

Here is the algorithm implemented in C:



```c
// Merges all 4 lines
void mergeLines(struct product L1[LINESIZE], struct product L2[LINESIZE], struct product L3[LINESIZE], struct product L4[LINESIZE], struct product masterLine[MASTERSIZE]) {
```



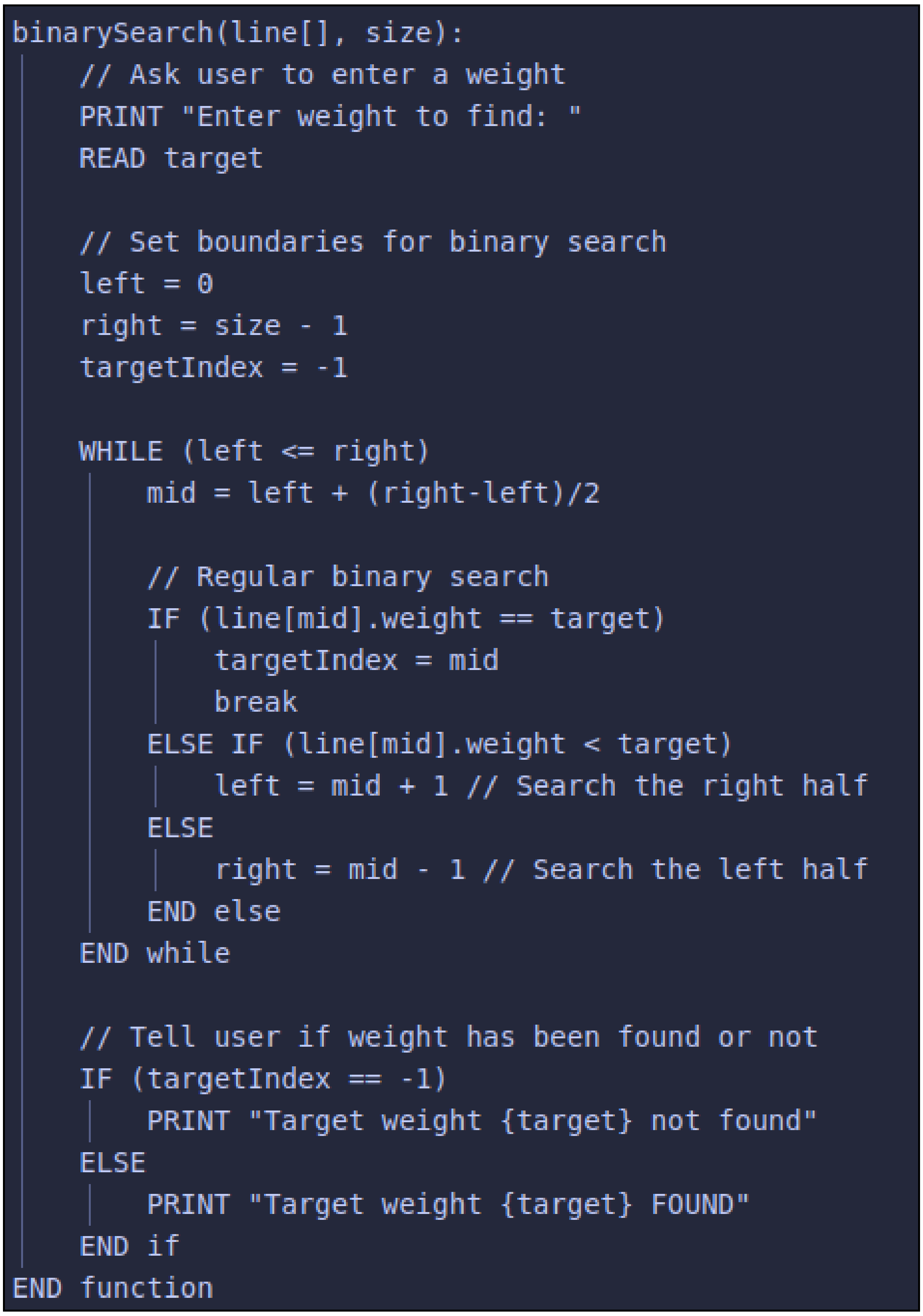
```c
    int n1 = 0;
    int n2 = 0;
    int n3 = 0;
    int n4 = 0;
    int min;
    int minList;

    // Loop through each the nth element of each line, and find the min
    for (int i=0; i<MASTERSIZE; i++) {
        min = INT_MAX;

        // Compare current elements of all 4 lines
        if (L1[n1].weight < min && n1 < LINESIZE) {
            min = L1[n1].weight;
            minList = 1;
        }
        if (L2[n2].weight < min && n2 < LINESIZE) {
            min = L2[n2].weight;
            minList = 2;
        }
        if (L3[n3].weight < min && n3 < LINESIZE) {
            min = L3[n3].weight;
            minList = 3;
        }
        if (L4[n4].weight < min && n4 < LINESIZE) {
            min = L4[n4].weight;
            minList = 4;
        }

        // Append that min value to the master line, then move that line up by 1
        if (minList == 1) {
            masterLine[i] = L1[n1];
            n1++;
        }
        else if (minList == 2) {
            masterLine[i] = L2[n2];
            n2++;
        }
        else if (minList == 3) {
            masterLine[i] = L3[n3];
            n3++;
        }
        else if (minList == 4) {
            masterLine[i] = L4[n4];
            n4++;
        }
    } // END for
} // END function
```

*image split for readability

Task 3

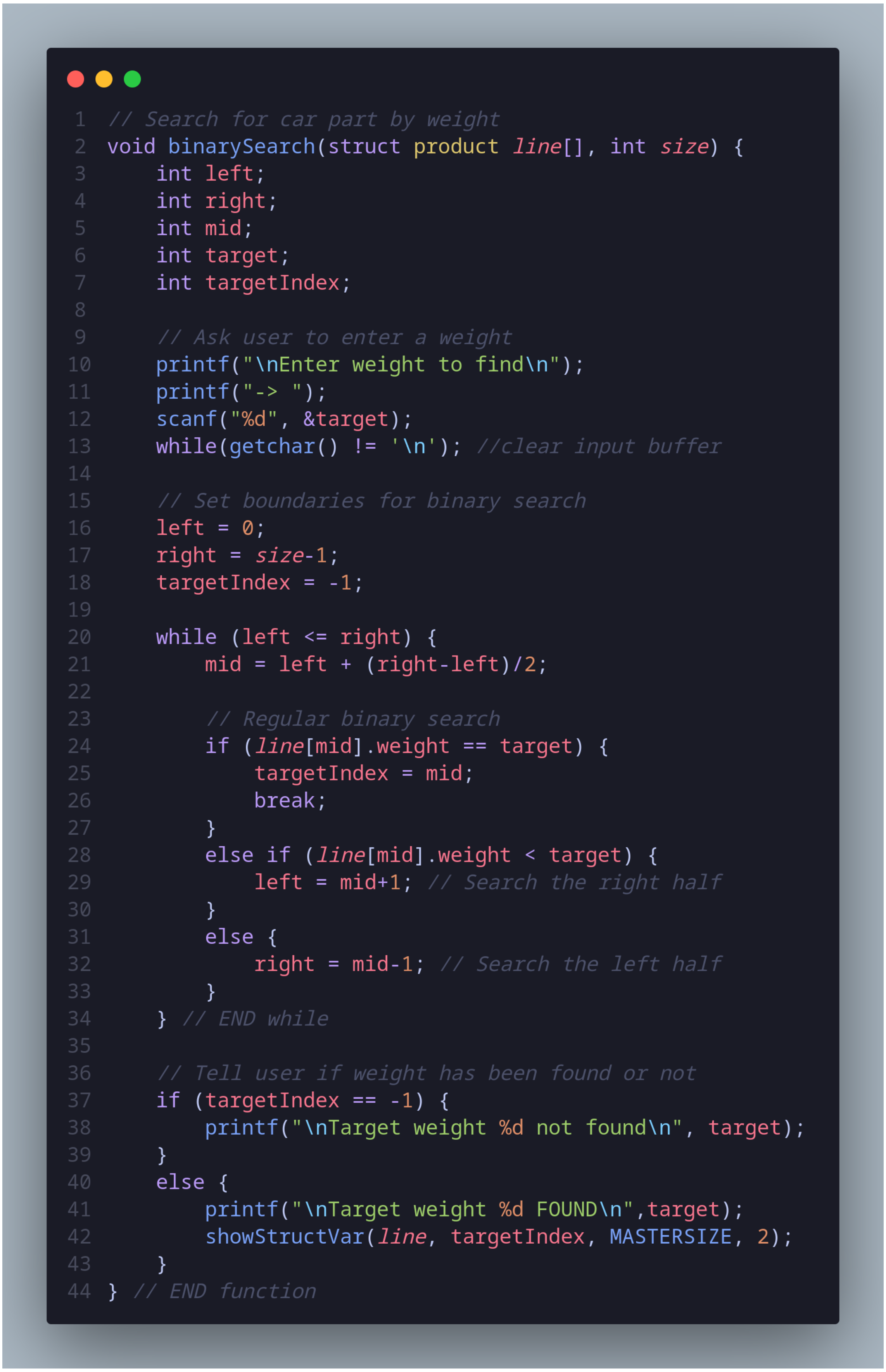
For this task, I had to implement a searching algorithm to find a specific car part by weight. The running time of the searching algorithm needed to have a time complexity of O(log(N)) or better. Due to the line of car parts being already sorted by tasks 1-2, I decided to use binary search, which has a worse case time complexity of O(log(n)). Binary search uses “divide and conquer” to repeatedly dive the array in half, until the target is found.

Here is the pseudocode:



```
binarySearch(line[], size):
    // Ask user to enter a weight
    PRINT "Enter weight to find: "
    READ target

    // Set boundaries for binary search
    left = 0
    right = size - 1
    targetIndex = -1

    WHILE (left <= right)
        mid = left + (right-left)/2

        // Regular binary search
        IF (line[mid].weight == target)
            targetIndex = mid
            break
        ELSE IF (line[mid].weight < target)
            left = mid + 1 // Search the right half
        ELSE
            right = mid - 1 // Search the left half
        END else
    END while

    // Tell user if weight has been found or not
    IF (targetIndex == -1)
        PRINT "Target weight {target} not found"
    ELSE
        PRINT "Target weight {target} FOUND"
    END if
END function
```

Here is binary search implemented in C, to work with the car parts.

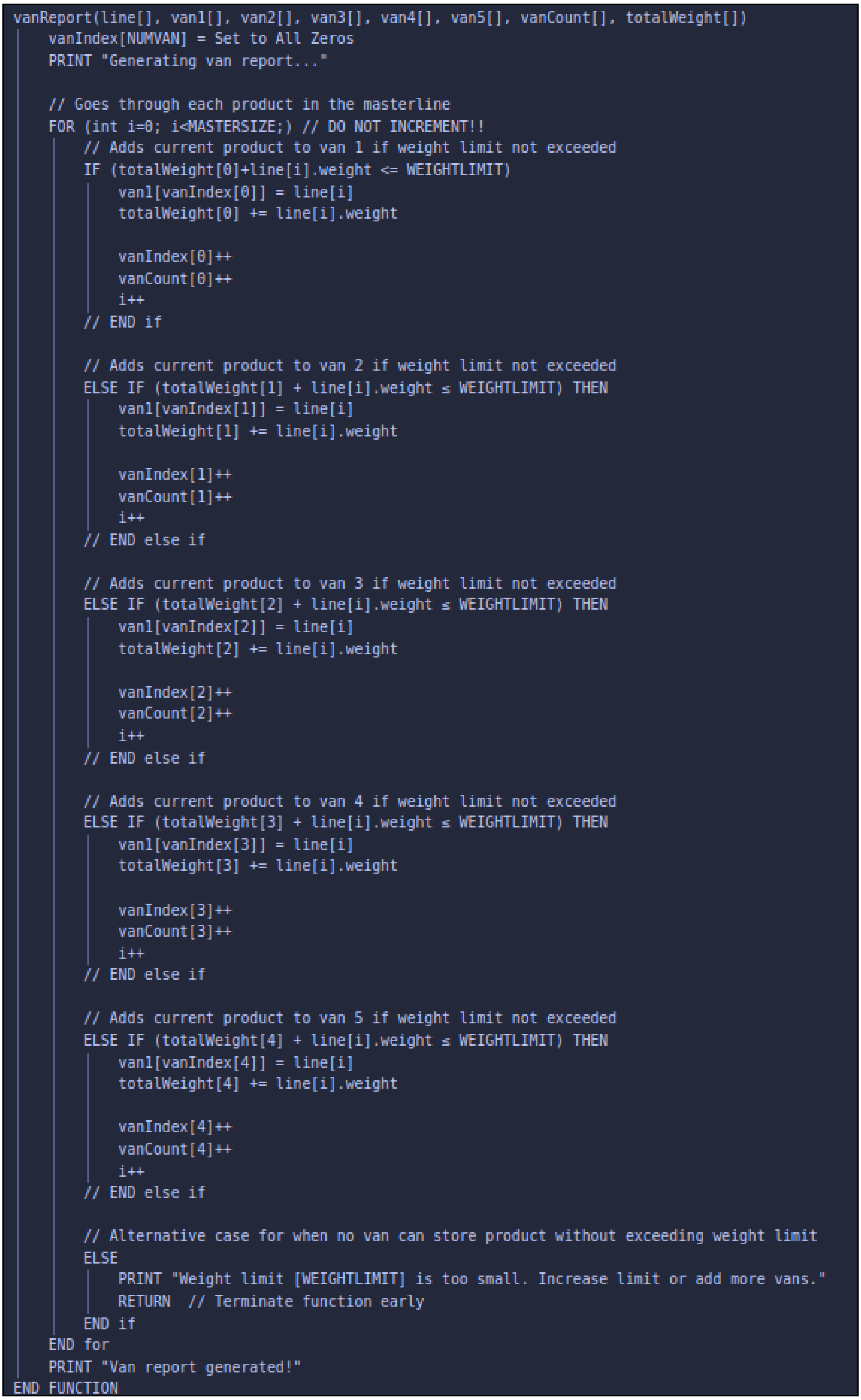


```c
// Search for car part by weight
void binarySearch(struct product line[], int size) {
    int left;
    int right;
    int mid;
    int target;
    int targetIndex;

    // Ask user to enter a weight
    printf("\nEnter weight to find\n");
    printf("-> ");
    scanf("%d", &target);
    while(getchar() != '\n'); //clear input buffer

    // Set boundaries for binary search
    left = 0;
    right = size-1;
    targetIndex = -1;

    while (left <= right) {
        mid = left + (right-left)/2;

        // Regular binary search
        if (line[mid].weight == target) {
            targetIndex = mid;
            break;
        }
        else if (line[mid].weight < target) {
            left = mid+1; // Search the right half
        }
        else {
            right = mid-1; // Search the left half
        }
    } // END while

    // Tell user if weight has been found or not
    if (targetIndex == -1) {
        printf("\nTarget weight %d not found\n", target);
    }
    else {
        printf("\nTarget weight %d FOUND\n",target);
        showStructVar(line, targetIndex, MASTERSIZE, 2);
    }
} // END function
```

Task 4

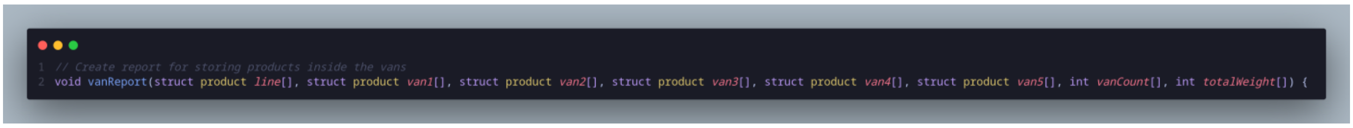
For this task, I had to design an algorithm that can create a report/delivery docket on the car parts included in the delivery for all vans. Each product must be stored in the van in weight order, however this is not a problem, as the maste rline is already sorted. The speed of this algorithm needed to be O(n) or better. To do this, I first created variables/symbolic names, containing the number of vans present, and the total weight limit for each van. Then, from the master line, the first product is stored in the first van, the second product is stored in the second van, and so on and so forth. If and when one of the vans hits the weight limit, the product will instead be stored inside the van after it. This repeats until either all the products are stored in the vans, or until the weight limit for each van is exceeded, in which the user is then informed. The worse case time complexity of this algorithm is O(n), as it only contains one loop to go through each element in the master line.

Here is the pseudocode:

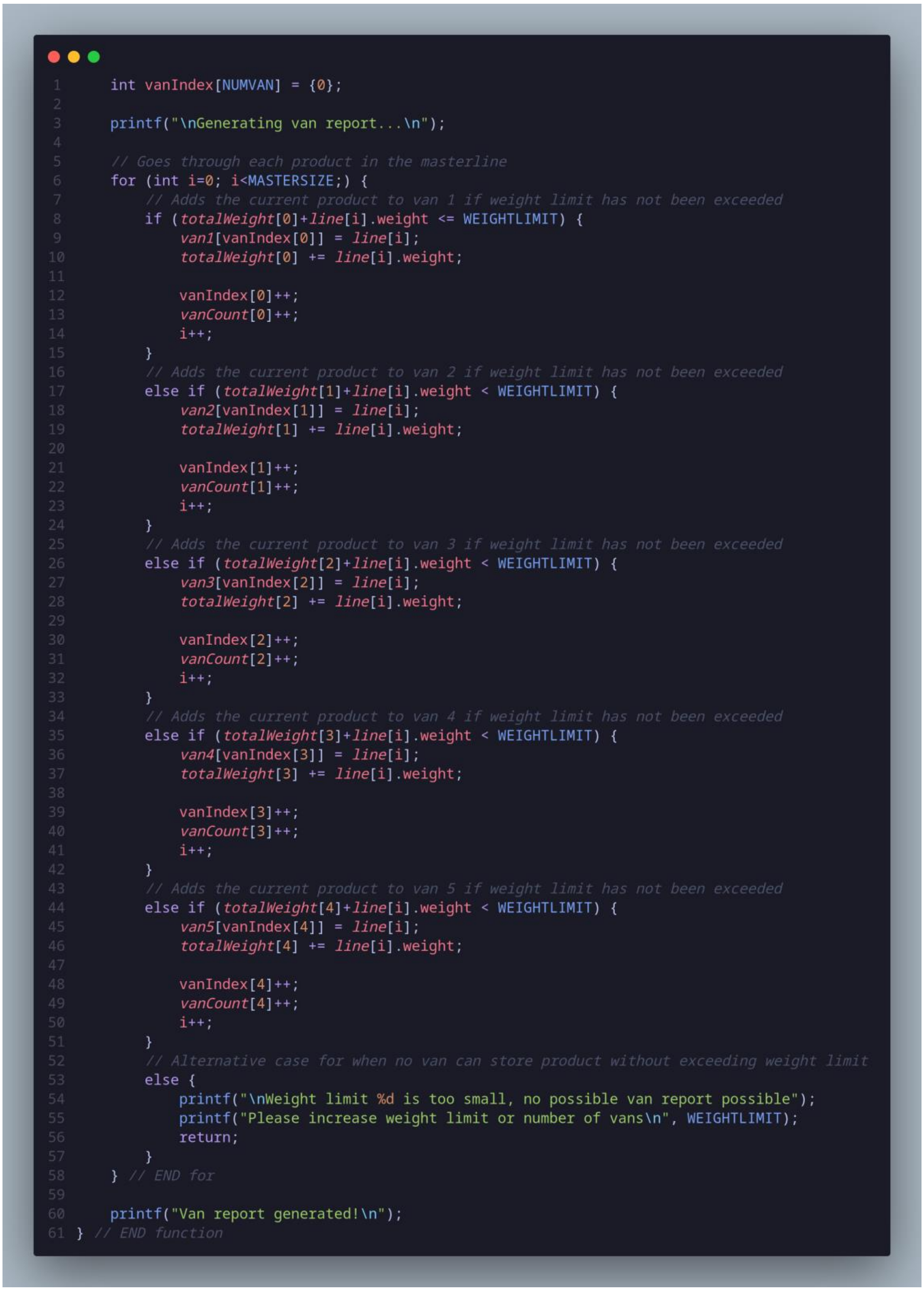


```
vanReport(line[], van1[], van2[], van3[], van4[], van5[], vanCount[], totalWeight[])
    vanIndex[NUMVAN] = Set to All Zeros
    PRINT "Generating van report..."

    // Goes through each product in the masterline
    FOR (int i=0; i<MASTERSIZE;) // DO NOT INCREMENT!!
        // Adds current product to van 1 if weight limit not exceeded
        IF (totalWeight[0]+line[i].weight <= WEIGHTLIMIT)
            van1[vanIndex[0]] = line[i]
            totalWeight[0] += line[i].weight

            vanIndex[0]++
            vanCount[0]++
            i++
        // END if

        // Adds current product to van 2 if weight limit not exceeded
        ELSE IF (totalWeight[1] + line[i].weight ≤ WEIGHTLIMIT) THEN
            van1[vanIndex[1]] = line[i]
            totalWeight[1] += line[i].weight

            vanIndex[1]++
            vanCount[1]++
            i++
        // END else if

        // Adds current product to van 3 if weight limit not exceeded
        ELSE IF (totalWeight[2] + line[i].weight ≤ WEIGHTLIMIT) THEN
            van1[vanIndex[2]] = line[i]
            totalWeight[2] += line[i].weight

            vanIndex[2]++
            vanCount[2]++
            i++
        // END else if

        // Adds current product to van 4 if weight limit not exceeded
        ELSE IF (totalWeight[3] + line[i].weight ≤ WEIGHTLIMIT) THEN
            van1[vanIndex[3]] = line[i]
            totalWeight[3] += line[i].weight

            vanIndex[3]++
            vanCount[3]++
            i++
        // END else if

        // Adds current product to van 5 if weight limit not exceeded
        ELSE IF (totalWeight[4] + line[i].weight ≤ WEIGHTLIMIT) THEN
            van1[vanIndex[4]] = line[i]
            totalWeight[4] += line[i].weight

            vanIndex[4]++
            vanCount[4]++
            i++
        // END else if

        // Alternative case for when no van can store product without exceeding weight limit
        ELSE
            PRINT "Weight limit [WEIGHTLIMIT] is too small. Increase limit or add more vans."
            RETURN  // Terminate function early
        END if
    END for
    PRINT "Van report generated!"
END FUNCTION
```

Here is the C code created from the pseudocode:



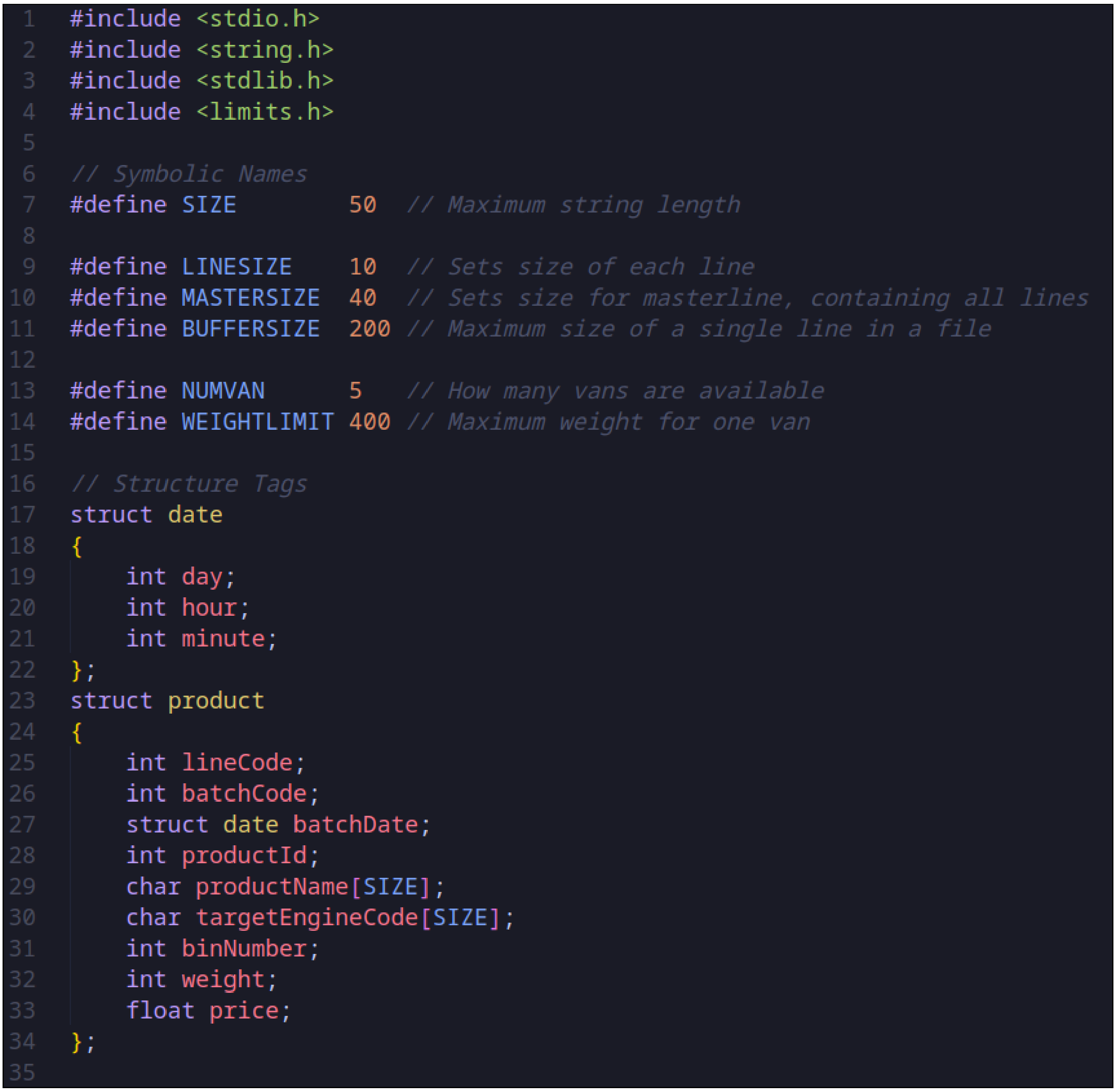
```c
// Create report for storing products inside the vans
void vanReport(struct product line[], struct product van1[], struct product van2[], struct product van3[], struct product van4[], struct product van5[], int vanCount[], int totalWeight[]) {
```



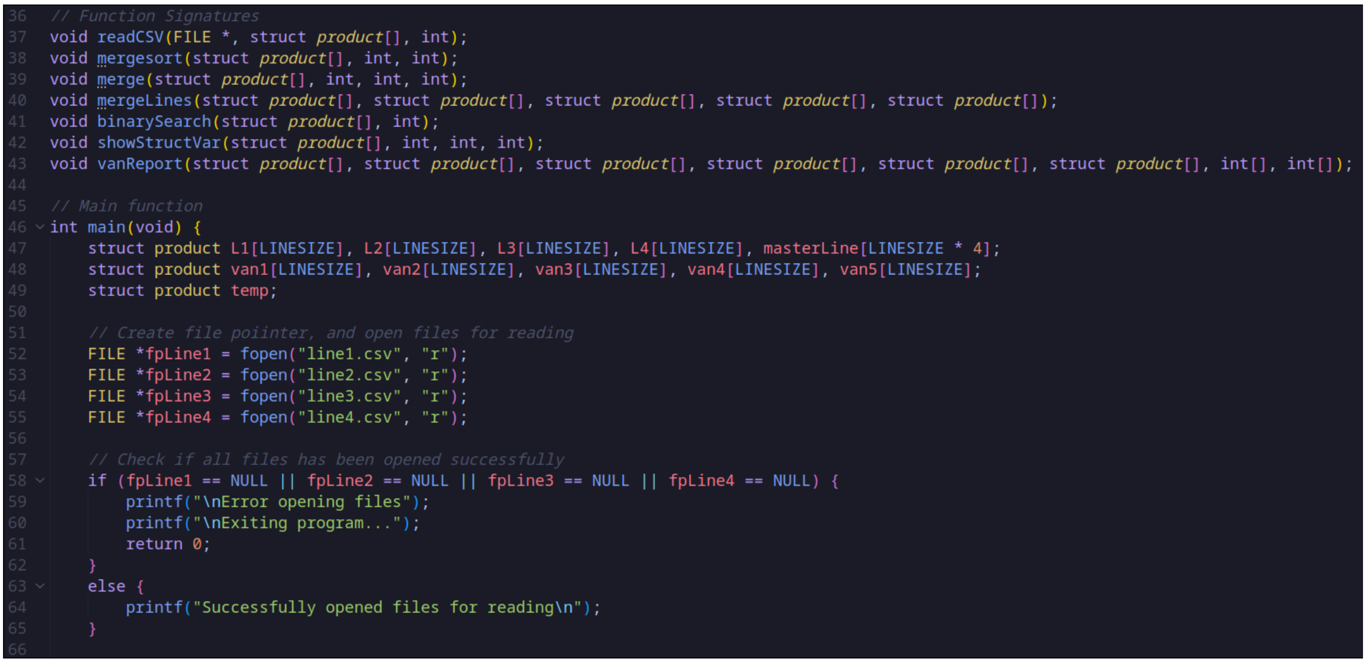
```c
    int vanIndex[NUMVAN] = {0};

    printf("\nGenerating van report...\n");

    // Goes through each product in the masterline
    for (int i=0; i<MASTERSIZE;) {
        // Adds the current product to van 1 if weight limit has not been exceeded
        if (totalWeight[0]+line[i].weight <= WEIGHTLIMIT) {
            van1[vanIndex[0]] = line[i];
            totalWeight[0] += line[i].weight;

            vanIndex[0]++;
            vanCount[0]++;
            i++;
        }
        // Adds the current product to van 2 if weight limit has not been exceeded
        else if (totalWeight[1]+line[i].weight < WEIGHTLIMIT) {
            van2[vanIndex[1]] = line[i];
            totalWeight[1] += line[i].weight;

            vanIndex[1]++;
            vanCount[1]++;
            i++;
        }
        // Adds the current product to van 3 if weight limit has not been exceeded
        else if (totalWeight[2]+line[i].weight < WEIGHTLIMIT) {
            van3[vanIndex[2]] = line[i];
            totalWeight[2] += line[i].weight;

            vanIndex[2]++;
            vanCount[2]++;
            i++;
        }
        // Adds the current product to van 4 if weight limit has not been exceeded
        else if (totalWeight[3]+line[i].weight < WEIGHTLIMIT) {
            van4[vanIndex[3]] = line[i];
            totalWeight[3] += line[i].weight;

            vanIndex[3]++;
            vanCount[3]++;
            i++;
        }
        // Adds the current product to van 5 if weight limit has not been exceeded
        else if (totalWeight[4]+line[i].weight < WEIGHTLIMIT) {
            van5[vanIndex[4]] = line[i];
            totalWeight[4] += line[i].weight;

            vanIndex[4]++;
            vanCount[4]++;
            i++;
        }
        // Alternative case for when no van can store product without exceeding weight limit
        else {
            printf("\nWeight limit %d is too small, no possible van report possible");
            printf("Please increase weight limit or number of vans\n", WEIGHTLIMIT);
            return;
        }
    } // END for

    printf("Van report generated!\n");
} // END function
```

*Image was split up for readability

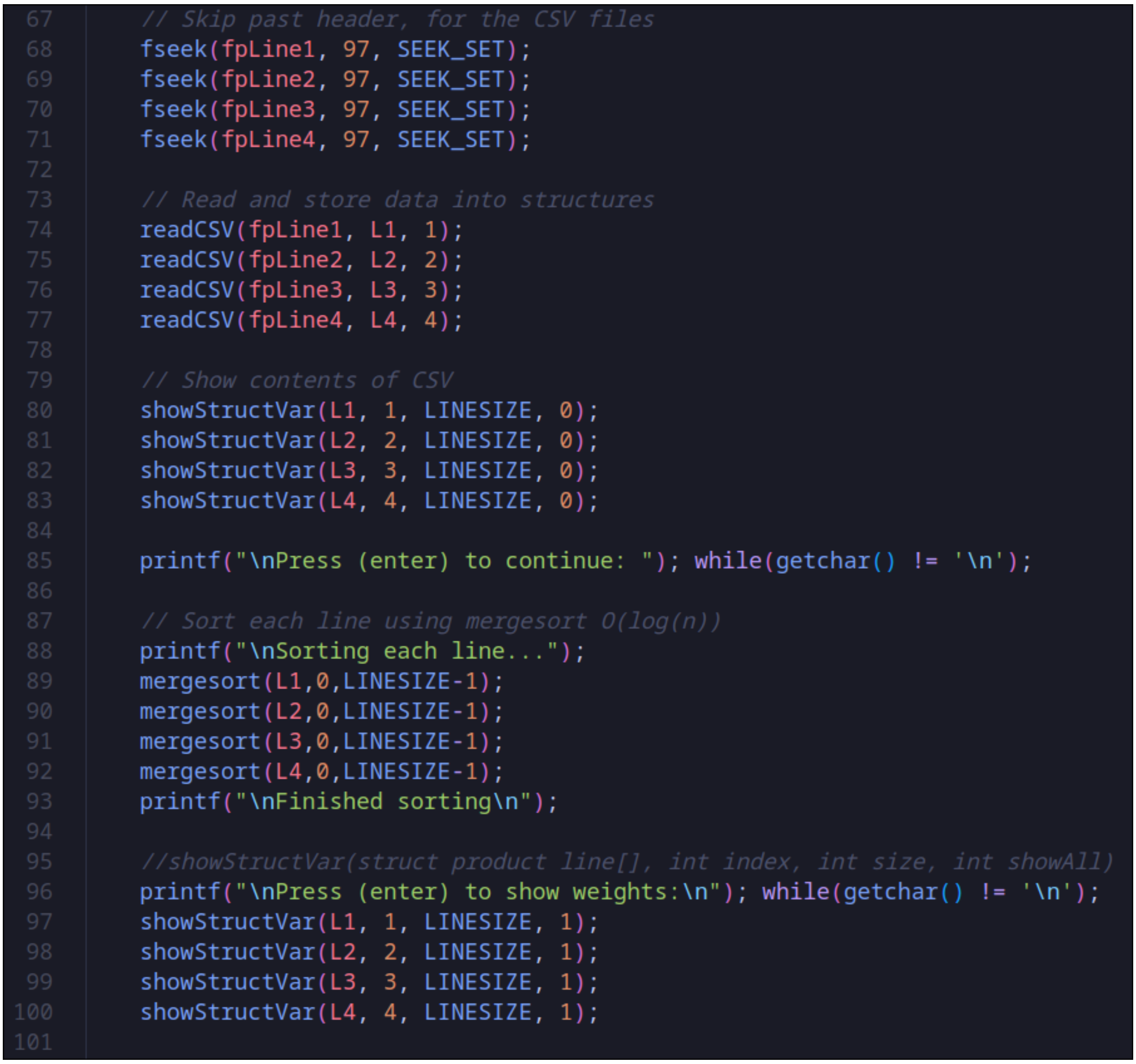
Images showing the entire code for the program



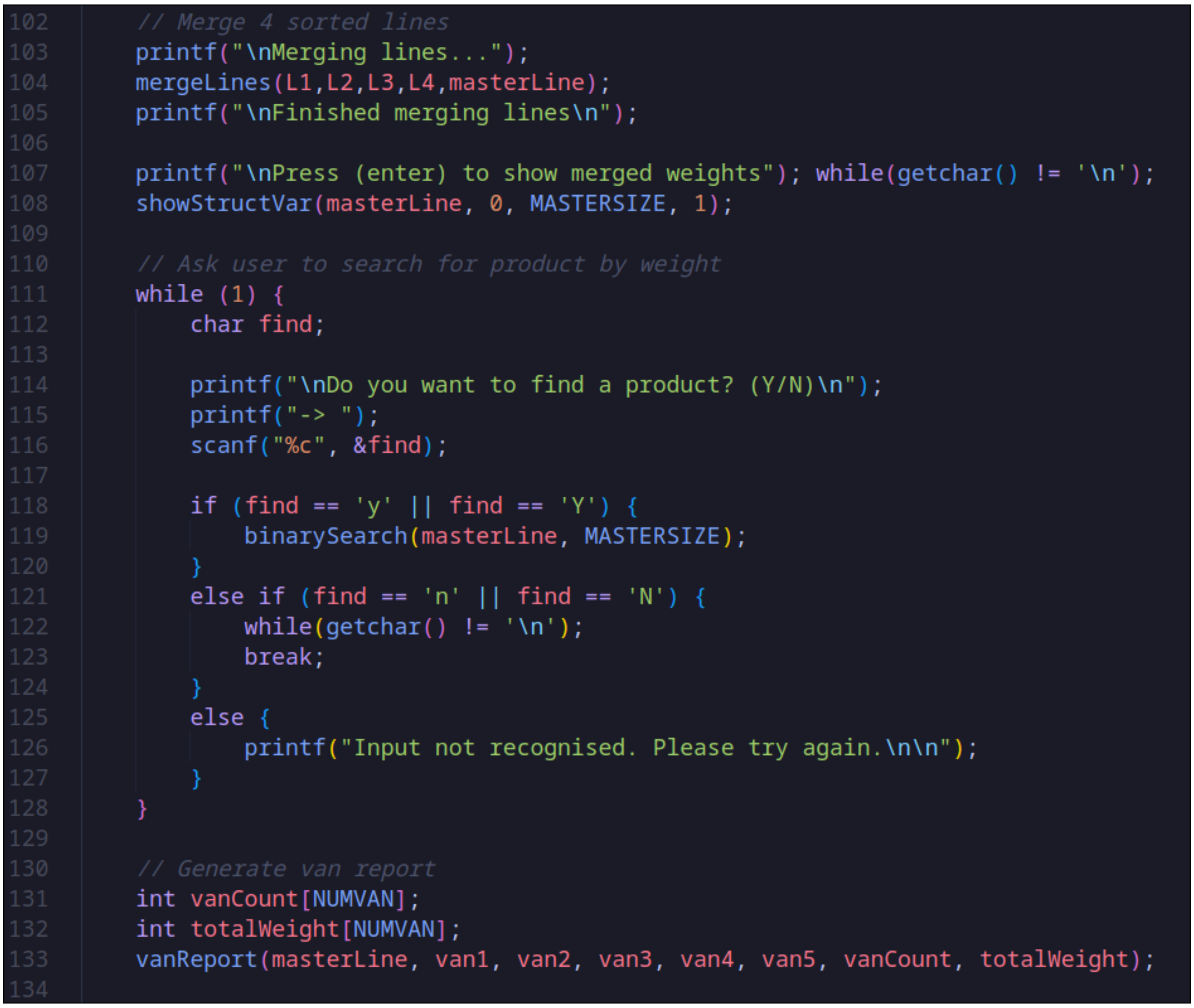
```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <limits.h>

// Symbolic Names
#define SIZE        50  // Maximum string length

#define LINESIZE    10  // Sets size of each line
#define MASTERSIZE  40  // Sets size for masterline, containing all lines
#define BUFFERSIZE  200 // Maximum size of a single line in a file

#define NUMVAN      5   // How many vans are available
#define WEIGHTLIMIT 400 // Maximum weight for one van

// Structure Tags
struct date
{
    int day;
    int hour;
    int minute;
};
struct product
{
    int lineCode;
    int batchCode;
    struct date batchDate;
    int productId;
    char productName[SIZE];
    char targetEngineCode[SIZE];
    int binNumber;
    int weight;
    float price;
};

```



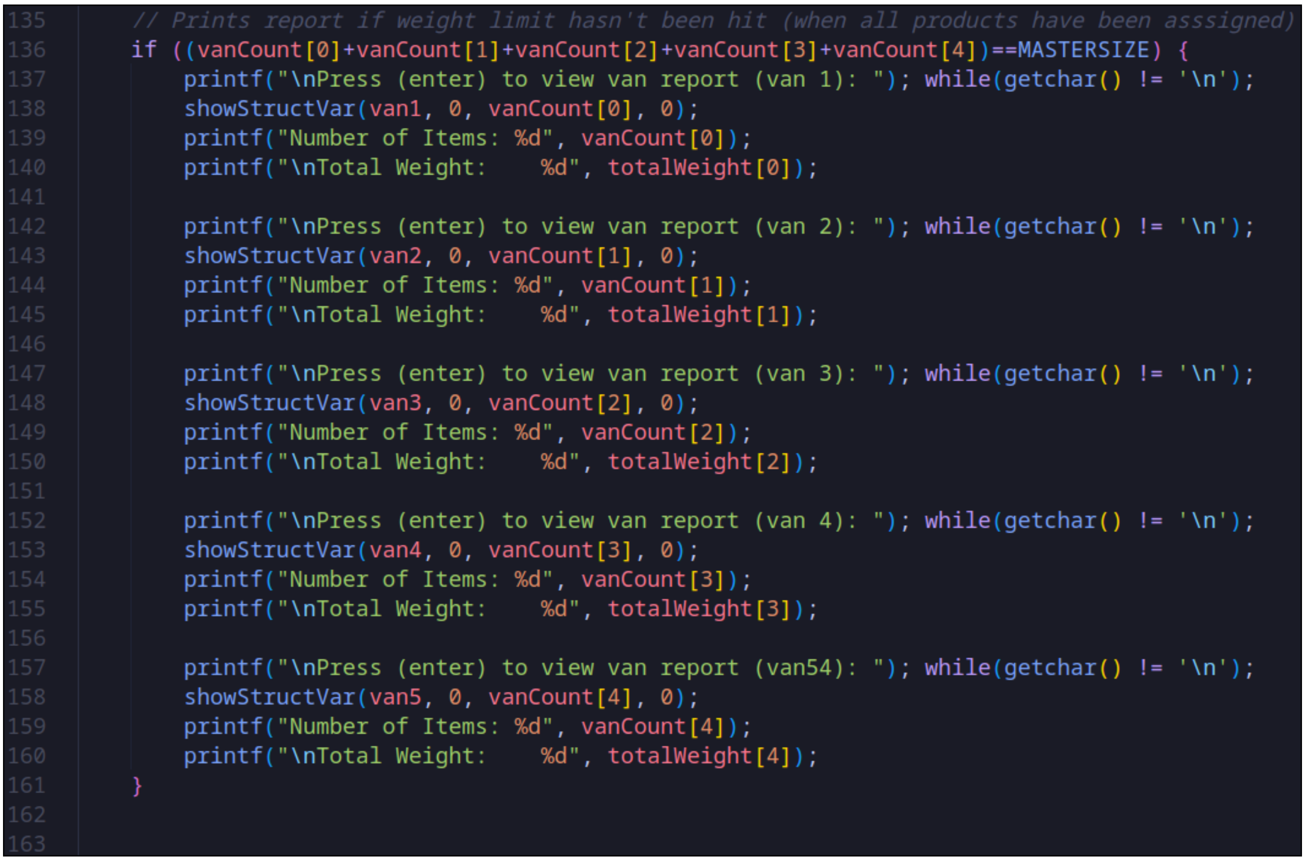
```c
// Function Signatures
void readCSV(FILE *, struct product[], int);
void mergesort(struct product[], int, int);
void merge(struct product[], int, int, int);
void mergeLines(struct product[], struct product[], struct product[], struct product[], struct product[]);
void binarySearch(struct product[], int);
void showStructVar(struct product[], int, int, int);
void vanReport(struct product[], struct product[], struct product[], struct product[], struct product[], struct product[], int[], int[]);

// Main function
int main(void) {
    struct product L1[LINESIZE], L2[LINESIZE], L3[LINESIZE], L4[LINESIZE], masterLine[LINESIZE * 4];
    struct product van1[LINESIZE], van2[LINESIZE], van3[LINESIZE], van4[LINESIZE], van5[LINESIZE];
    struct product temp;

    // Create file poiinter, and open files for reading
    FILE *fpLine1 = fopen("line1.csv", "r");
    FILE *fpLine2 = fopen("line2.csv", "r");
    FILE *fpLine3 = fopen("line3.csv", "r");
    FILE *fpLine4 = fopen("line4.csv", "r");

    // Check if all files has been opened successfully
    if (fpLine1 == NULL || fpLine2 == NULL || fpLine3 == NULL || fpLine4 == NULL) {
        printf("\nError opening files");
        printf("\nExiting program...");
        return 0;
    }
    else {
        printf("Successfully opened files for reading\n");
    }

```



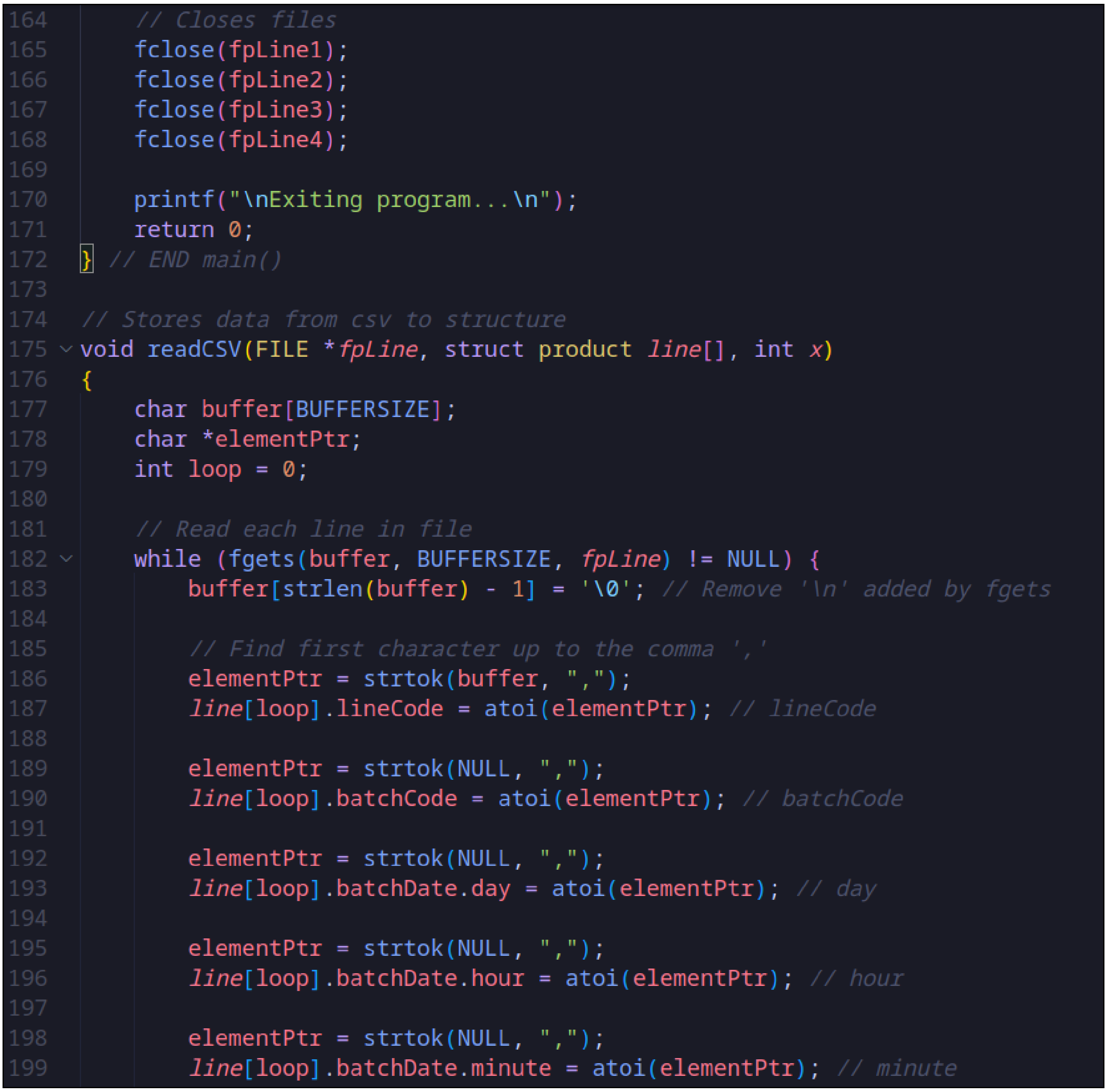
```c
    // Skip past header, for the CSV files
    fseek(fpLine1, 97, SEEK_SET);
    fseek(fpLine2, 97, SEEK_SET);
    fseek(fpLine3, 97, SEEK_SET);
    fseek(fpLine4, 97, SEEK_SET);

    // Read and store data into structures
    readCSV(fpLine1, L1, 1);
    readCSV(fpLine2, L2, 2);
    readCSV(fpLine3, L3, 3);
    readCSV(fpLine4, L4, 4);

    // Show contents of CSV
    showStructVar(L1, 1, LINESIZE, 0);
    showStructVar(L2, 2, LINESIZE, 0);
    showStructVar(L3, 3, LINESIZE, 0);
    showStructVar(L4, 4, LINESIZE, 0);

    printf("\nPress (enter) to continue: "); while(getchar() != '\n');

    // Sort each line using mergesort O(log(n))
    printf("\nSorting each line...");
    mergesort(L1,0,LINESIZE-1);
    mergesort(L2,0,LINESIZE-1);
    mergesort(L3,0,LINESIZE-1);
    mergesort(L4,0,LINESIZE-1);
    printf("\nFinished sorting\n");

    //showStructVar(struct product line[], int index, int size, int showAll)
    printf("\nPress (enter) to show weights:\n"); while(getchar() != '\n');
    showStructVar(L1, 1, LINESIZE, 1);
    showStructVar(L2, 2, LINESIZE, 1);
    showStructVar(L3, 3, LINESIZE, 1);
    showStructVar(L4, 4, LINESIZE, 1);

```



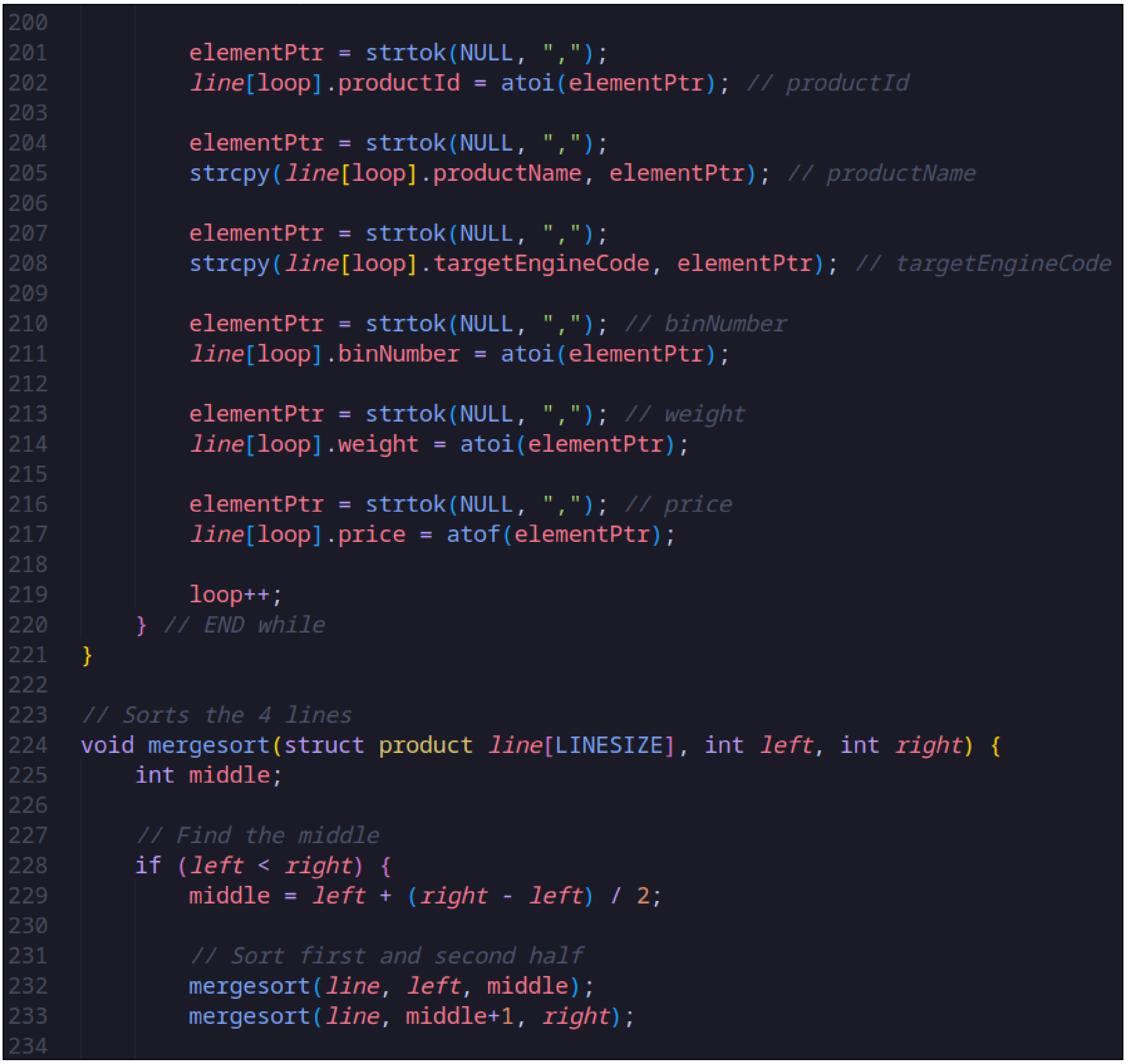
```c
    // Merge 4 sorted lines
    printf("\nMerging lines...");
    mergeLines(L1,L2,L3,L4,masterLine);
    printf("\nFinished merging lines\n");

    printf("\nPress (enter) to show merged weights"); while(getchar() != '\n');
    showStructVar(masterLine, 0, MASTERSIZE, 1);

    // Ask user to search for product by weight
    while (1) {
        char find;

        printf("\nDo you want to find a product? (Y/N)\n");
        printf("-> ");
        scanf("%c", &find);

        if (find == 'y' || find == 'Y') {
            binarySearch(masterLine, MASTERSIZE);
        }
        else if (find == 'n' || find == 'N') {
            while(getchar() != '\n');
            break;
        }
        else {
            printf("Input not recognised. Please try again.\n\n");
        }
    }

    // Generate van report
    int vanCount[NUMVAN];
    int totalWeight[NUMVAN];
    vanReport(masterLine, van1, van2, van3, van4, van5, vanCount, totalWeight);

```



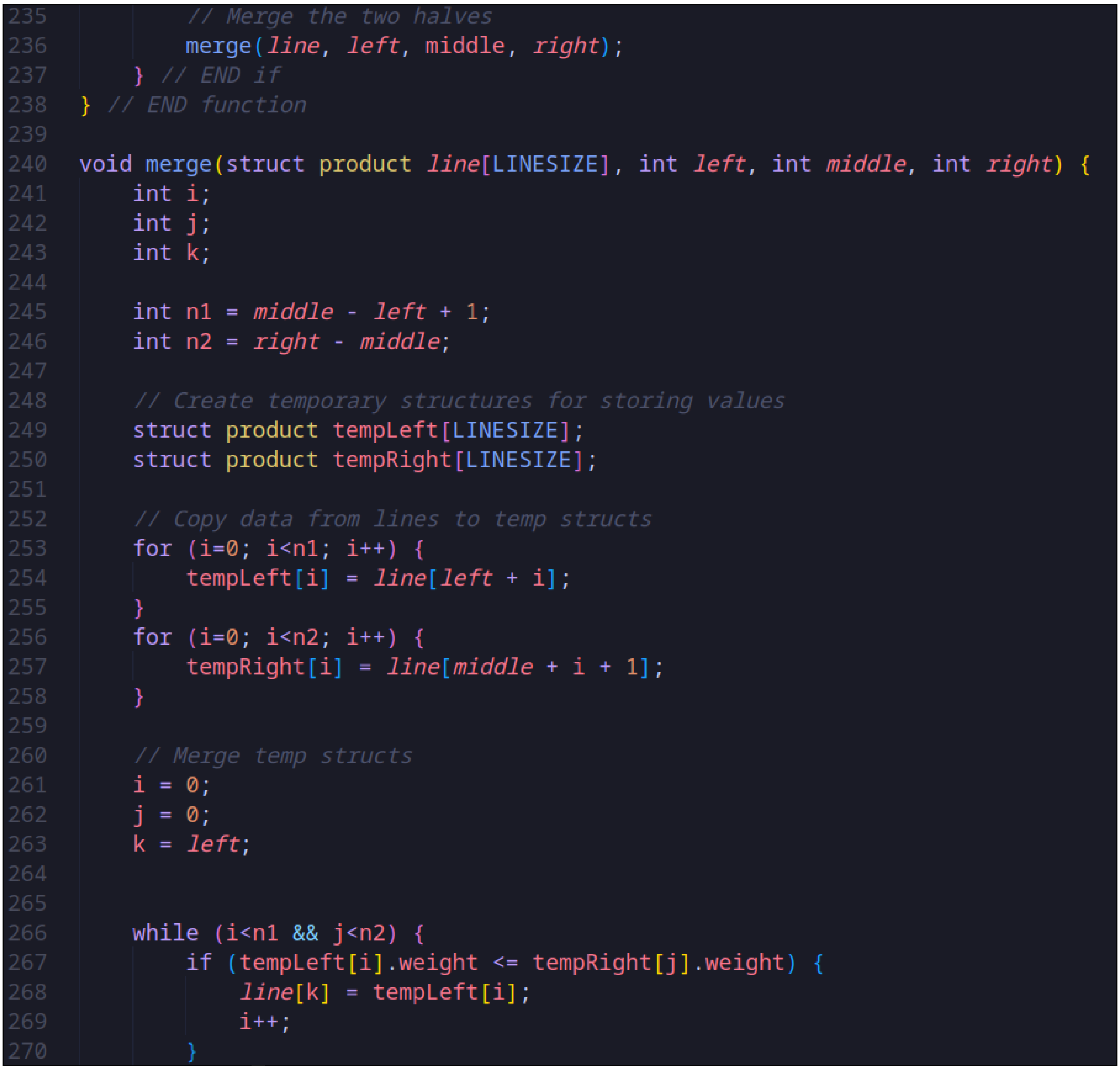
```
    // Prints report if weight limit hasn't been hit (when all products have been asssigned)
    if ((vanCount[0]+vanCount[1]+vanCount[2]+vanCount[3]+vanCount[4])==MASTERSIZE) {
        printf("\nPress (enter) to view van report (van 1): "); while(getchar() != '\n');
        showStructVar(van1, 0, vanCount[0], 0);
        printf("Number of Items: %d", vanCount[0]);
        printf("\nTotal Weight:    %d", totalWeight[0]);

        printf("\nPress (enter) to view van report (van 2): "); while(getchar() != '\n');
        showStructVar(van2, 0, vanCount[1], 0);
        printf("Number of Items: %d", vanCount[1]);
        printf("\nTotal Weight:    %d", totalWeight[1]);

        printf("\nPress (enter) to view van report (van 3): "); while(getchar() != '\n');
        showStructVar(van3, 0, vanCount[2], 0);
        printf("Number of Items: %d", vanCount[2]);
        printf("\nTotal Weight:    %d", totalWeight[2]);

        printf("\nPress (enter) to view van report (van 4): "); while(getchar() != '\n');
        showStructVar(van4, 0, vanCount[3], 0);
        printf("Number of Items: %d", vanCount[3]);
        printf("\nTotal Weight:    %d", totalWeight[3]);

        printf("\nPress (enter) to view van report (van54): "); while(getchar() != '\n');
        showStructVar(van5, 0, vanCount[4], 0);
        printf("Number of Items: %d", vanCount[4]);
        printf("\nTotal Weight:    %d", totalWeight[4]);
    }


```



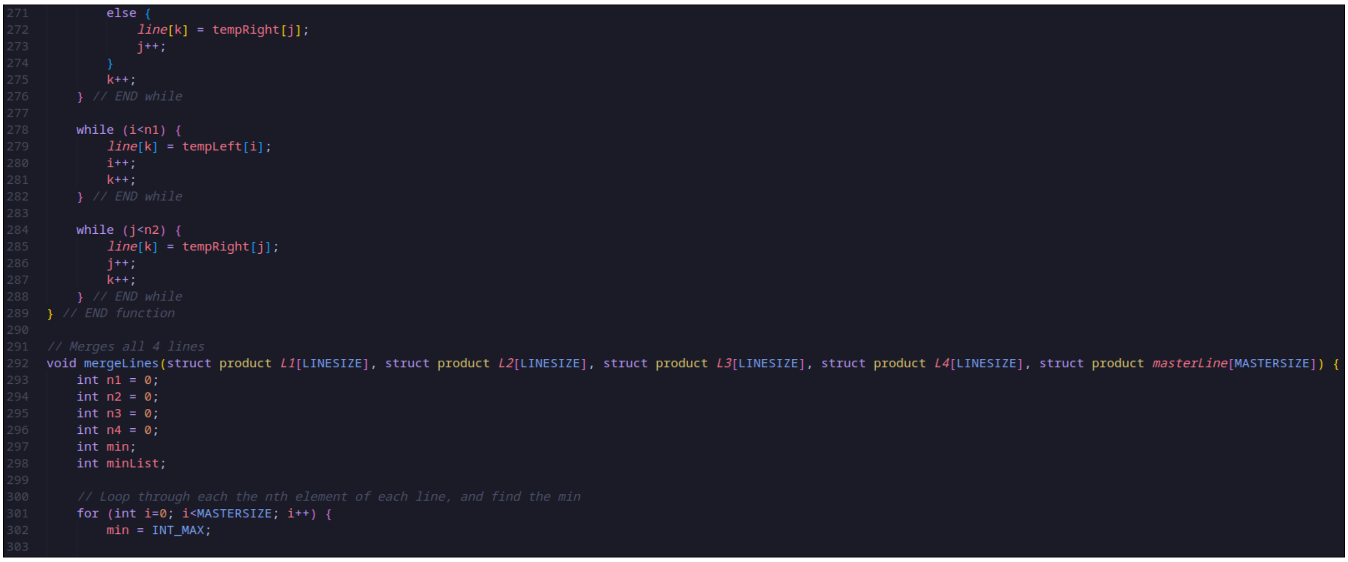
```c
    // Closes files
    fclose(fpLine1);
    fclose(fpLine2);
    fclose(fpLine3);
    fclose(fpLine4);

    printf("\nExiting program...\n");
    return 0;
} // END main()

// Stores data from csv to structure
void readCSV(FILE *fpLine, struct product line[], int x)
{
    char buffer[BUFFERSIZE];
    char *elementPtr;
    int loop = 0;

    // Read each line in file
    while (fgets(buffer, BUFFERSIZE, fpLine) != NULL) {
        buffer[strlen(buffer) - 1] = '\0'; // Remove '\n' added by fgets

        // Find first character up to the comma ','
        elementPtr = strtok(buffer, ",");
        line[loop].lineCode = atoi(elementPtr); // lineCode

        elementPtr = strtok(NULL, ",");
        line[loop].batchCode = atoi(elementPtr); // batchCode

        elementPtr = strtok(NULL, ",");
        line[loop].batchDate.day = atoi(elementPtr); // day

        elementPtr = strtok(NULL, ",");
        line[loop].batchDate.hour = atoi(elementPtr); // hour

        elementPtr = strtok(NULL, ",");
        line[loop].batchDate.minute = atoi(elementPtr); // minute
```



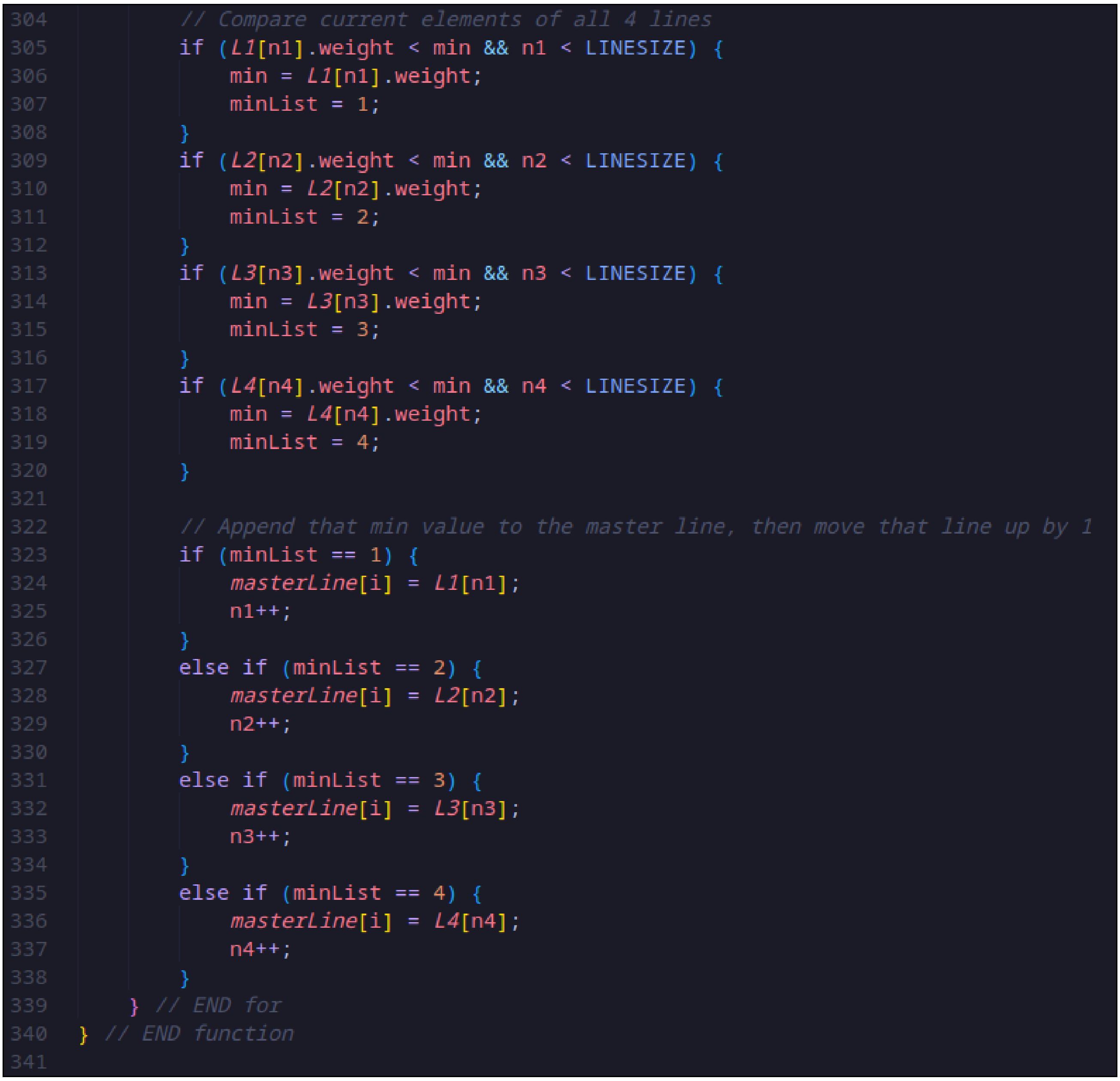
```c

        elementPtr = strtok(NULL, ",");
        line[loop].productId = atoi(elementPtr); // productId

        elementPtr = strtok(NULL, ",");
        strcpy(line[loop].productName, elementPtr); // productName

        elementPtr = strtok(NULL, ",");
        strcpy(line[loop].targetEngineCode, elementPtr); // targetEngineCode

        elementPtr = strtok(NULL, ","); // binNumber
        line[loop].binNumber = atoi(elementPtr);

        elementPtr = strtok(NULL, ","); // weight
        line[loop].weight = atoi(elementPtr);

        elementPtr = strtok(NULL, ","); // price
        line[loop].price = atof(elementPtr);

        loop++;
    } // END while
}

// Sorts the 4 lines
void mergesort(struct product line[LINESIZE], int left, int right) {
    int middle;

    // Find the middle
    if (left < right) {
        middle = left + (right - left) / 2;

        // Sort first and second half
        mergesort(line, left, middle);
        mergesort(line, middle+1, right);

```



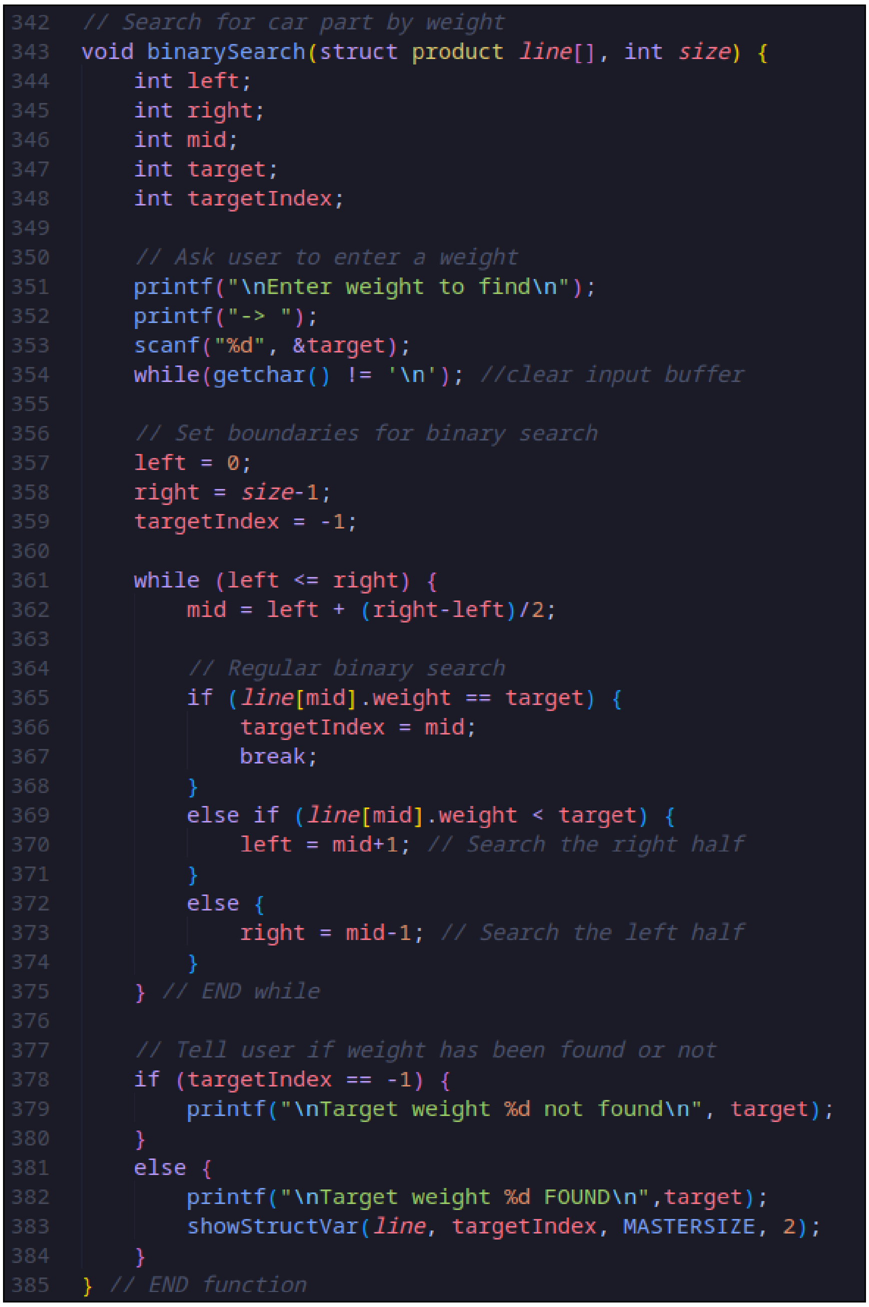
```c
        // Merge the two halves
        merge(line, left, middle, right);
    } // END if
} // END function

void merge(struct product line[LINESIZE], int left, int middle, int right) {
    int i;
    int j;
    int k;

    int n1 = middle - left + 1;
    int n2 = right - middle;

    // Create temporary structures for storing values
    struct product tempLeft[LINESIZE];
    struct product tempRight[LINESIZE];

    // Copy data from lines to temp structs
    for (i=0; i<n1; i++) {
        tempLeft[i] = line[left + i];
    }
    for (i=0; i<n2; i++) {
        tempRight[i] = line[middle + i + 1];
    }

    // Merge temp structs
    i = 0;
    j = 0;
    k = left;


    while (i<n1 && j<n2) {
        if (tempLeft[i].weight <= tempRight[j].weight) {
            line[k] = tempLeft[i];
            i++;
        }
```



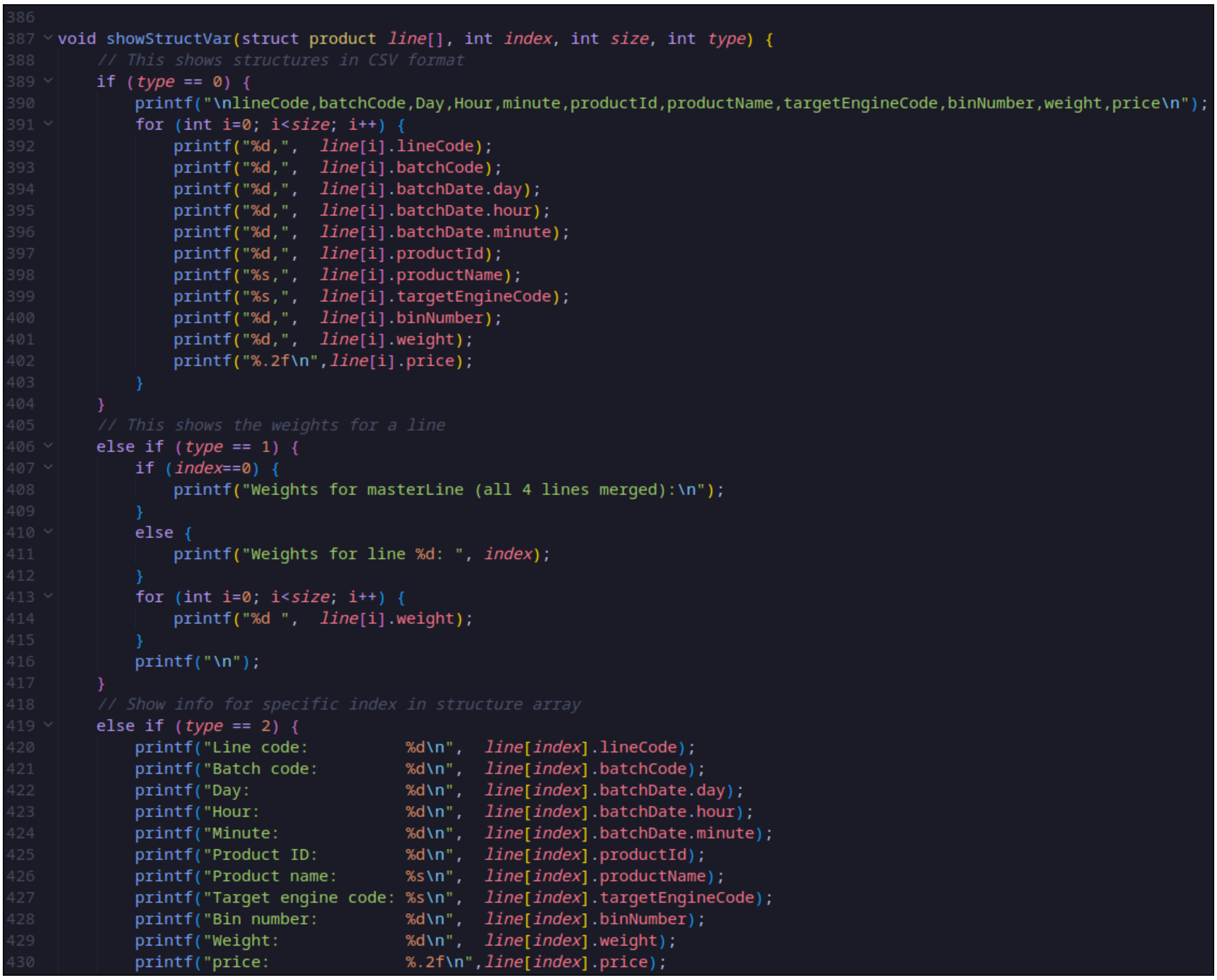
```c
        else {
            line[k] = tempRight[j];
            j++;
        }
        k++;
    } // END while

    while (i<n1) {
        line[k] = tempLeft[i];
        i++;
        k++;
    } // END while

    while (j<n2) {
        line[k] = tempRight[j];
        j++;
        k++;
    } // END while
} // END function

// Merges all 4 lines
void mergeLines(struct product L1[LINESIZE], struct product L2[LINESIZE], struct product L3[LINESIZE], struct product L4[LINESIZE], struct product masterLine[MASTERSIZE]) {
    int n1 = 0;
    int n2 = 0;
    int n3 = 0;
    int n4 = 0;
    int min;
    int minList;

    // Loop through each the nth element of each line, and find the min
    for (int i=0; i<MASTERSIZE; i++) {
        min = INT_MAX;

```



```
        // Compare current elements of all 4 lines
        if (L1[n1].weight < min && n1 < LINESIZE) {
            min = L1[n1].weight;
            minList = 1;
        }
        if (L2[n2].weight < min && n2 < LINESIZE) {
            min = L2[n2].weight;
            minList = 2;
        }
        if (L3[n3].weight < min && n3 < LINESIZE) {
            min = L3[n3].weight;
            minList = 3;
        }
        if (L4[n4].weight < min && n4 < LINESIZE) {
            min = L4[n4].weight;
            minList = 4;
        }

        // Append that min value to the master line, then move that line up by 1
        if (minList == 1) {
            masterLine[i] = L1[n1];
            n1++;
        }
        else if (minList == 2) {
            masterLine[i] = L2[n2];
            n2++;
        }
        else if (minList == 3) {
            masterLine[i] = L3[n3];
            n3++;
        }
        else if (minList == 4) {
            masterLine[i] = L4[n4];
            n4++;
        }
    } // END for
} // END function
```

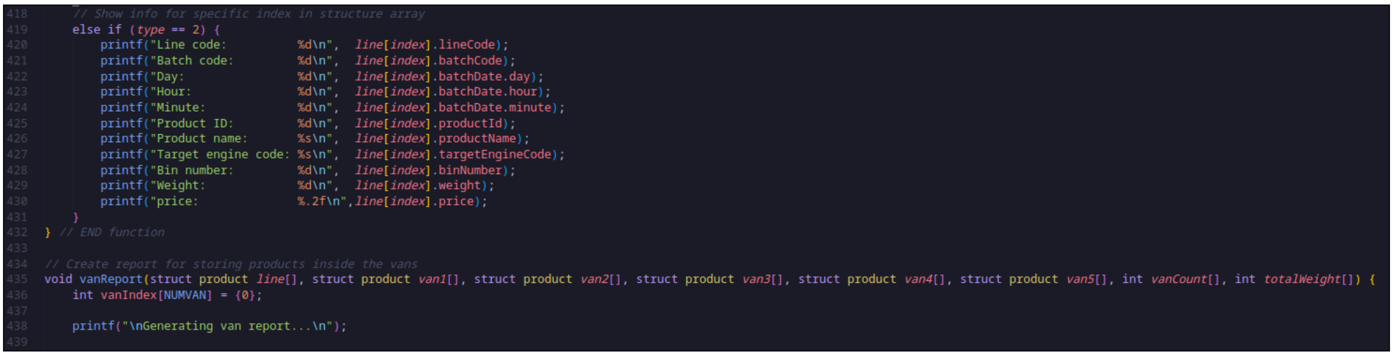


```c
// Search for car part by weight
void binarySearch(struct product line[], int size) {
    int left;
    int right;
    int mid;
    int target;
    int targetIndex;

    // Ask user to enter a weight
    printf("\nEnter weight to find\n");
    printf("-> ");
    scanf("%d", &target);
    while(getchar() != '\n'); //clear input buffer

    // Set boundaries for binary search
    left = 0;
    right = size-1;
    targetIndex = -1;

    while (left <= right) {
        mid = left + (right-left)/2;

        // Regular binary search
        if (line[mid].weight == target) {
            targetIndex = mid;
            break;
        }
        else if (line[mid].weight < target) {
            left = mid+1; // Search the right half
        }
        else {
            right = mid-1; // Search the left half
        }
    } // END while

    // Tell user if weight has been found or not
    if (targetIndex == -1) {
        printf("\nTarget weight %d not found\n", target);
    }
    else {
        printf("\nTarget weight %d FOUND\n",target);
        showStructVar(line, targetIndex, MASTERSIZE, 2);
    }
} // END function
```

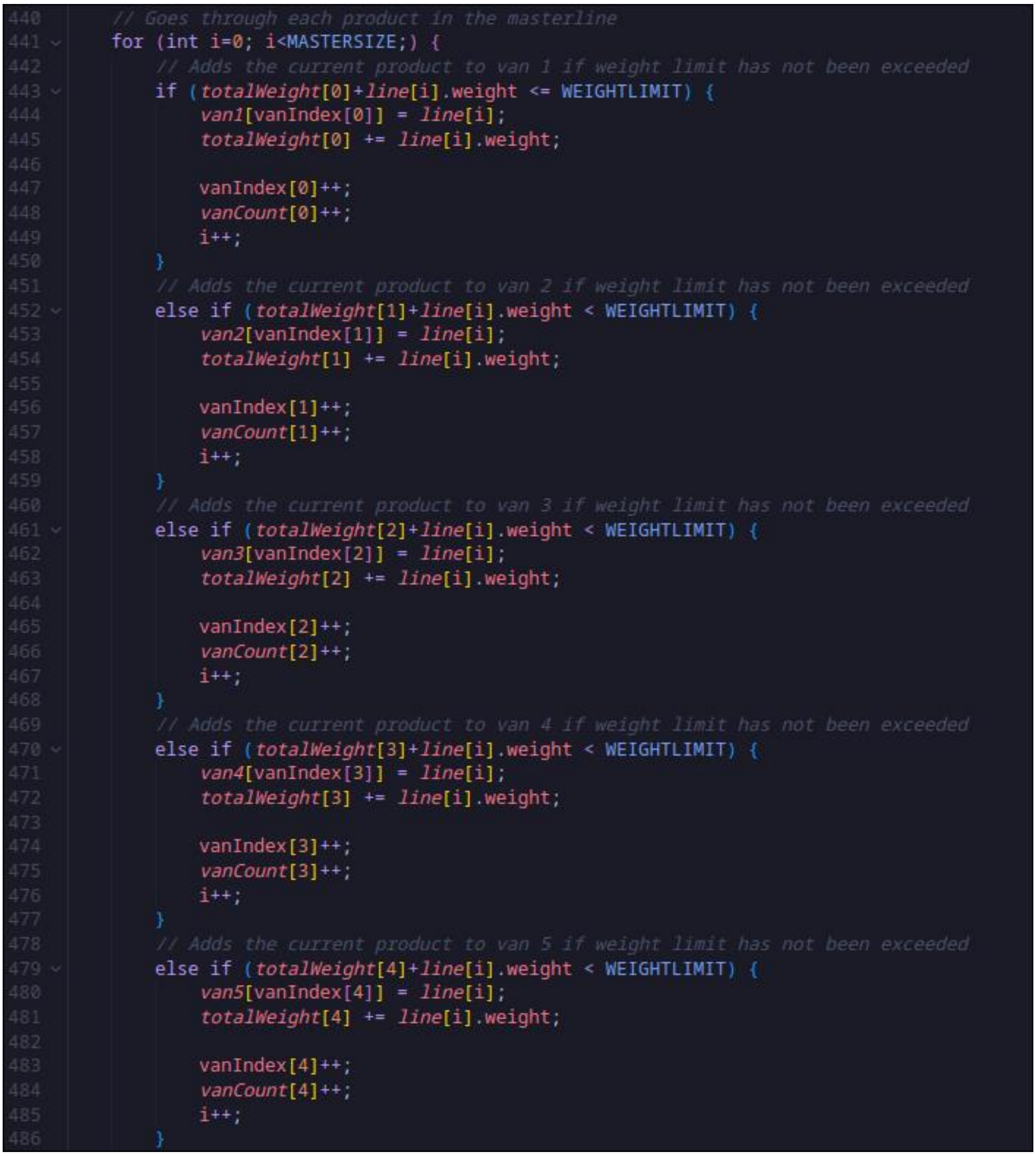


```c

void showStructVar(struct product line[], int index, int size, int type) {
    // This shows structures in CSV format
    if (type == 0) {
        printf("\nlineCode,batchCode,Day,Hour,minute,productId,productName,targetEngineCode,binNumber,weight,price\n");
        for (int i=0; i<size; i++) {
            printf("%d,",  line[i].lineCode);
            printf("%d,",  line[i].batchCode);
            printf("%d,",  line[i].batchDate.day);
            printf("%d,",  line[i].batchDate.hour);
            printf("%d,",  line[i].batchDate.minute);
            printf("%d,",  line[i].productId);
            printf("%s,",  line[i].productName);
            printf("%s,",  line[i].targetEngineCode);
            printf("%d,",  line[i].binNumber);
            printf("%d,",  line[i].weight);
            printf("%.2f\n",line[i].price);
        }
    }
    // This shows the weights for a line
    else if (type == 1) {
        if (index==0) {
            printf("Weights for masterLine (all 4 lines merged):\n");
        }
        else {
            printf("Weights for line %d: ", index);
        }
        for (int i=0; i<size; i++) {
            printf("%d ",  line[i].weight);
        }
        printf("\n");
    }
    // Show info for specific index in structure array
    else if (type == 2) {
        printf("Line code:          %d\n",  line[index].lineCode);
        printf("Batch code:         %d\n",  line[index].batchCode);
        printf("Day:                %d\n",  line[index].batchDate.day);
        printf("Hour:               %d\n",  line[index].batchDate.hour);
        printf("Minute:             %d\n",  line[index].batchDate.minute);
        printf("Product ID:         %d\n",  line[index].productId);
        printf("Product name:       %s\n",  line[index].productName);
        printf("Target engine code: %s\n",  line[index].targetEngineCode);
        printf("Bin number:         %d\n",  line[index].binNumber);
        printf("Weight:             %d\n",  line[index].weight);
        printf("price:              %.2f\n",line[index].price);
```

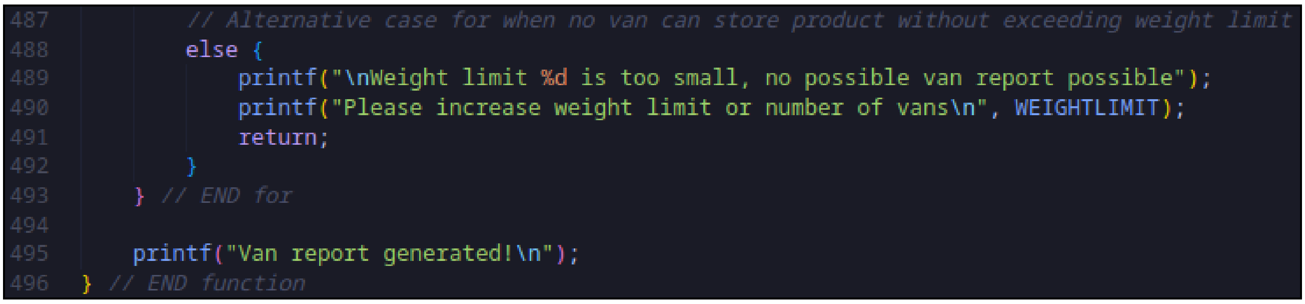
```c

void showStructVar(struct product line[], int index, int size, int type) {
    // This shows structures in CSV format
    if (type == 0) {
        printf("\nlineCode,batchCode,Day,Hour,minute,productId,productName,targetEngineCode,binNumber,weight,price\n");
        for (int i=0; i<size; i++) {
            printf("%d,",  line[i].lineCode);
            printf("%d,",  line[i].batchCode);
            printf("%d,",  line[i].batchDate.day);
            printf("%d,",  line[i].batchDate.hour);
            printf("%d,",  line[i].batchDate.minute);
            printf("%d,",  line[i].productId);
            printf("%s,",  line[i].productName);
            printf("%s,",  line[i].targetEngineCode);
            printf("%d,",  line[i].binNumber);
            printf("%d,",  line[i].weight);
            printf("%.2f\n",line[i].price);
        }
    }
    // This shows the weights for a line
    else if (type == 1) {
        if (index==0) {
            printf("Weights for masterLine (all 4 lines merged):\n");
        }
        else {
            printf("Weights for line %d: ", index);
        }
        for (int i=0; i<size; i++) {
            printf("%d ",  line[i].weight);
        }
        printf("\n");
    }
```



```c
    // Show info for specific index in structure array
    else if (type == 2) {
        printf("Line code:          %d\n",  line[index].lineCode);
        printf("Batch code:         %d\n",  line[index].batchCode);
        printf("Day:                %d\n",  line[index].batchDate.day);
        printf("Hour:               %d\n",  line[index].batchDate.hour);
        printf("Minute:             %d\n",  line[index].batchDate.minute);
        printf("Product ID:         %d\n",  line[index].productId);
        printf("Product name:       %s\n",  line[index].productName);
        printf("Target engine code: %s\n",  line[index].targetEngineCode);
        printf("Bin number:         %d\n",  line[index].binNumber);
        printf("Weight:             %d\n",  line[index].weight);
        printf("price:              %.2f\n",line[index].price);
    }
} // END function

// Create report for storing products inside the vans
void vanReport(struct product line[], struct product van1[], struct product van2[], struct product van3[], struct product van4[], struct product van5[], int vanCount[], int totalWeight[]) {
    int vanIndex[NUMVAN] = {0};

    printf("\nGenerating van report...\n");

```



```c
    // Goes through each product in the masterline
    for (int i=0; i<MASTERSIZE;) {
        // Adds the current product to van 1 if weight limit has not been exceeded
        if (totalWeight[0]+line[i].weight <= WEIGHTLIMIT) {
            van1[vanIndex[0]] = line[i];
            totalWeight[0] += line[i].weight;

            vanIndex[0]++;
            vanCount[0]++;
            i++;
        }
        // Adds the current product to van 2 if weight limit has not been exceeded
        else if (totalWeight[1]+line[i].weight < WEIGHTLIMIT) {
            van2[vanIndex[1]] = line[i];
            totalWeight[1] += line[i].weight;

            vanIndex[1]++;
            vanCount[1]++;
            i++;
        }
        // Adds the current product to van 3 if weight limit has not been exceeded
        else if (totalWeight[2]+line[i].weight < WEIGHTLIMIT) {
            van3[vanIndex[2]] = line[i];
            totalWeight[2] += line[i].weight;

            vanIndex[2]++;
            vanCount[2]++;
            i++;
        }
        // Adds the current product to van 4 if weight limit has not been exceeded
        else if (totalWeight[3]+line[i].weight < WEIGHTLIMIT) {
            van4[vanIndex[3]] = line[i];
            totalWeight[3] += line[i].weight;

            vanIndex[3]++;
            vanCount[3]++;
            i++;
        }
        // Adds the current product to van 5 if weight limit has not been exceeded
        else if (totalWeight[4]+line[i].weight < WEIGHTLIMIT) {
            van5[vanIndex[4]] = line[i];
            totalWeight[4] += line[i].weight;

            vanIndex[4]++;
            vanCount[4]++;
            i++;
        }
```



```c
        // Alternative case for when no van can store product without exceeding weight limit
        else {
            printf("\nWeight limit %d is too small, no possible van report possible");
            printf("Please increase weight limit or number of vans\n", WEIGHTLIMIT);
            return;
        }
    } // END for

    printf("Van report generated!\n");
} // END function
```