ASIA PACIFIC UNIVERSITY OF TECHNOLOGY & INNOVATION

CT018-3-1-ICP
INTRODUCTION TO C PROGRAMMING

TITLE	INDIVIDUAL ASSIGNMENT
STUDENT NAME	MOHAMMAD FAWZAN ALIM
STUDENT ID	TP064501
INTAKE	APD1F2106CE
LECTURER NAME	SUPRIYA SINGH
SUBMISSION DATE	27 FEBRUARY 2022

TABLE OF CONTENTS

INTRODUCTION

Database management systems are programs that gets used to store, retrieve, process data. Database management system allows users to create, edit, update and delete data in the database efficiently and systematically (Appdynamics, n.d.). Database can manipulate data and make analytical reports with automatically in a very short time. These reports are very accurate and helps the management of a business make smart decisions.

An automobile manufacturing company in Sungai Tunas manufactures various models of passenger cars. For every model, they have divisions and sections. With such growth, it is becoming hard to keep track of data. So, they want an inventory management program to automate the system and store the data following same structure. With this simple program, they will be able to easily update their inventory, produce various reports and search the whole database.

We have to use the programming knowledge taught in Introduction to C Programming module to create a C program to meet the requirement. To meet the requirement of the program, we can use either linked list or struct array. I decided to use linked list in this program. Even though it is hard to work with, it gives more flexibility to the program.

ASSUMPTION

For this assignment, we have assumed that the automobile manufacturing company manufactures more than one model. Each model has its own warehouse to store parts. Assembly team requests parts from its own warehouse. An assembly team of a model cannot request parts from a different warehouse. Every model of the company has a unique code which acts like an ID.

For every model, there are a fixed number of sections. Every part is under a section of a model. A part has a unique ID across the company, and it cannot be used by multiple warehouses. Even if they are identical, system must recognize them as different parts under different model. Meaning they will have different IDs, will be stored in different warehouses and will be used by different assembly team.

Parts getting used in each warehouse must come from a supplier. Though a supplier can supply multiple parts for different models, but a part cannot be supplied by multiple suppliers. The supplier details need to be recorded separately in the database.

There is no need for authentication in the program as it is assumed that the people who will use it are ethical and will not misuse their given authorization. It is also assumed that there will only be one instance of the program running in one master PC for all the warehouses.

PROGRAM DESIGN

PSEUDOCODE

```
PROGRAM Inventory Management System

BEGIN
    DECLARE menuChoice, models[10][2], suppliers[25][2], parts[250][10]

    REPEAT
        models = CALL readExistingModels(models)
        suppliers = CALL readExistingSuppliers(suppliers)

        PRINT "HOME MENU:"
        PRINT "1. Update inventory"
        PRINT "2. Inventory tracking"
        PRINT "3. Search inventory"
        PRINT "4. Add a new model"
        PRINT "5. Add a new part"
        PRINT "6. Delete a model"
        PRINT "7. Delete a part"
        PRINT "0. Exit"

        PRINT "Enter your choice of operation: "
        ACCEPT menuChoice

        IF (menuChoice = 1) THEN
            parts = CALL updateInventory(models, parts)
        ELSE
            IF (menuChoice = 2) THEN
                CALL inventoryTracking(models, suppliers, parts)
            ELSE
                IF (menuChoice = 3) THEN
                    CALL searchInventory(models, suppliers, parts)
                ELSE
                    IF (menuChoice = 4) THEN
                        models = CALL addNewModel(models)
                    ELSE
                        IF (menuChoice = 5) THEN
                            suppliers, parts = CALL addNewPart(models, suppliers, parts)
                        ELSE
                            IF (menuChoice = 6) THEN
                                models = CALL deleteModel(models)
                            ELSE
                                IF (menuChoice = 7) THEN
                                    parts = CALL deletePart(models, parts)
                                ELSE
                                    RETURN
                                ENDIF
                            ENDIF
                        ENDIF
                    ENDIF
                ENDIF
            ENDIF
        ENDIF
    UNTIL (menuChoice <> 0)
END
```

```
FUNCTION addNewModel(models)
    DECLARE i, l, filename
    l = CALL length(models)
    PRINT "Enter model code: "
    ACCEPT models[l][0]
    PRINT "Enter model name: "
    ACCEPT model[l][1]

    filename = models[l][1] + ".txt"

    SORT models IN ALPHABETICAL ORDER

    OPENFILE "carmodels.txt" FOR WRITE
        WRITEFILE "CODE"
        WRITEFILE "NAME"
        WRITEFILE NEWLINE
        i = 0
        DOWHILE (END OF FILE NOT REACHED)
            WRITEFILE models[i][0]
            WRITEFILE models[i][1]
            WRITEFILE NEWLINE

            i = i + 1
        ENDDO
    CLOSEFILE "carmodels.txt"

    OPENFILE filename FOR WRITE
        WRITEFILE "PART ID"
        WRITEFILE "MODEL CODE"
        WRITEFILE "MODEL NAME"
        WRITEFILE "SECTION CODE"
        WRITEFILE "SECTION NAME"
        WRITEFILE "PART CODE"
        WRITEFILE "PART NAME"
        WRITEFILE "QUANTITY"
        WRITEFILE "SUPPLIER CODE"
        WRITEFILE "SUPPLIER NAME"
        WRITEFILE NEWLINE
    CLOSEFILE filename
    RETURN models
ENDFUNCTION
```

```
FUNCTION readExistingModels(models)
    DECLARE i = 0

    OPENFILE "carModels.txt" FOR READ
    SKIPFILE 1 LINE

    DOWHILE (END OF FILE NOT REACHED)
        READFILE models[i][0]
        READFILE models[i][1]
        i = i + 1
    ENDDO
    CLOSEFILE "carmodels.txt"
    RETURN models
ENDFUNCTION
```

```
FUNCTION deleteModel(models)
    IF (CALL length(models) = 0) THEN
        PRINT "No available models. Add a model first."
        RETURN models
    ENDIF

    DECLARE i, filename
    i = CALL getModelCode(models)
    filename = models[i][0] + ".txt"

    DOWHILE (i < CALL length(models))
        models[i] = models[i+1]
        i = i + 1
    ENDDO

    OPENFILE "carModels.txt" FOR WRITE
        WRITEFILE "CODE"
        WRITEFILE "NAME"
        WRITEFILE NEWLINE
        i = 0
        DOWHILE (i < CALL length(models))
            WRITEFILE models[i][0]
            WRITEFILE models[i][1]
            WRITEFILE NEWLINE
            i = i + 1
        ENDDO
    CLOSEFILE "carModels.txt"

    DELETEFILE filename
    RETURN models
ENDFUNCTION
```

```
FUNCTION getModelCode(models)
    DECLARE i, modelCode

    PRINT "Available Models: "
    PRINT NEWLINE
    PRINT "CODE"
    PRINT "NAME"
    PRINT NEWLINE
    i = 0
    DOWHILE (i < CALL length(models))
        PRINT models[i][0]
        PRINT models[i][1]
        PRINT NEWLINE
        i = i + 1
    ENDDO

    PRINT "Choose model code: "
    ACCEPT modelCode

    i = 0
    DOWHILE (models[i][0] <> modelCode)
        i = i + 1
    ENDDO

    RETURN i
ENDFUNCTION
```

```
FUNCTION readExistingSuppliers(suppliers)
    DECLARE i = 0

    OPENFILE "suppliers.txt" FOR READ
    SKIPFILE 1 LINE

    DOWHILE (END OF FILE NOT REACHED)
        READFILE supplier[i][0]
        READFILE supplier[i][1]
        i = i + 1
    ENDDO
    CLOSEFILE "suppliers.txt"
    RETURN suppliers
ENDFUNCTION
```

```
FUNCTION readExistingParts(model, parts)
    DECLARE i, filename

    filename = model[0] + ".txt"
    OPENFILE filename FOR READ
        i = 0
        SKIPFILE 1 LINE
        DOWHILE (END OF FILE NOT REACHED)
            READFILE parts[i][0]
            READFILE parts[i][1]
            READFILE parts[i][2]
            READFILE parts[i][3]
            READFILE parts[i][4]
            READFILE parts[i][5]
            READFILE parts[i][6]
            READFILE parts[i][7]
            READFILE parts[i][8]
            READFILE parts[i][9]

            i = i + 1
        ENDDO
    CLOSEFILE
    RETURN parts
ENDFUNCTION
```

```
FUNCTION freePartList(parts)
    DECLARE i
    i = CALL length(parts)

    DOWHILE (i - 1 >= 0)
        parts[i] = NULL
        i = i - 1
    ENDDO

    RETURN parts
ENDFUNCTION
```

```
FUNCTION length(array)
    DECLARE i = 0

    DOWHILE (array[i] <> NULL)
        i = i + 1
    ENDDO
    RETURN i
ENDFUNCTION
```

```
FUNCTION addNewPart(models, suppliers, parts)
    IF (CALL length(models) = 0) THEN
        PRINT "No available models. Add a model first"
        RETURN suppliers, parts
    ENDIF
    DECLARE i, j, k, l, filename

    i = CALL getModelCode(models)
    parts = CALL readExistingParts(models[i], parts)
    j = CALL length(parts)

    parts[j][1] = models[i][0]
    parts[j][2] = models[i][1]

    PRINT "Available Section Code:"
    PRINT "BD. Body Work"
    PRINT "CS. Chassis"
    PRINT "EN. Engine"
    PRINT "TR. Transmission System"
    PRINT "OT. Others"

    PRINT "Enter section code: "
    ACCEPT parts[j][3]

    IF (parts[j][3] = "BD") THEN
        parts[j][4] = "Body Work"
    ELSE
        IF (parts[j][3] = "CS") THEN
            parts[j][4] = "Chassis"
        ELSE
            IF (parts[j][3] = "EN") THEN
                parts[j][4] = "Engine"
            ELSE
                IF (parts[j][3] = "TR") THEN
                    parts[j][4] = "Transmission System"
                ELSE
                    parts[j][4] = "Others"
                ENDIF
            ENDIF
        ENDIF
    ENDIF
```

```
PRINT "Enter part code: "
ACCEPT parts[j][5]

PRINT "Enter part name: "
ACCEPT parts[j][6]

PRINT "Enter Quantity: "
ACCEPT parts[j][7]

PRINT "Existing suppliers: "
PRINT NEWLINE
PRINT "CODE"
PRINT "NAME"
PRINT NEWLINE
k = 0
l = CALL length(suppliers)
DOWHILE (k < l)
    PRINT suppliers[k][0]
    PRINT suppliers[k][1]
    PRINT NEWLINE
    k = k + 1
ENDDO

PRINT "Does your supplier already exist? [Y/N]"
ACCEPT supplierExist

IF (supplierExist = 'Y') THEN
    PRINT "Choose supplier code: "
    ACCEPT parts[j][8]
    k = 0
    DOWHILE (parts[j][8] <> suppliers[k][0])
        k = k + 1
    ENDDO
    parts[j][9] = suppliers[k][1]
ELSE
    PRINT "Enter supplier code: "
    ACCEPT suppliers[l][0]
    PRINT "Enter supplier name: "
    ACCEPT suppliers[l][1]

    parts[j][8] = suppliers[l][0]
    parts[j][9] = suppliers[l][1]

    SORT suppliers IN ALPHABETICAL ORDER
ENDIF
```

```
    parts[j][0] = parts[j][1] + "-" + parts[j][3] + "-" + parts[j][5]

    SORT parts IN ALPHABETICAL ORDER

    filename = models[i][0] + ".txt"

    OPENFILE filename FOR WRITE
        WRITEFILE "PART ID"
        WRITEFILE "MODEL CODE"
        WRITEFILE "MODEL NAME"
        WRITEFILE "SECTION CODE"
        WRITEFILE "SECTION NAME"
        WRITEFILE "PART CODE"
        WRITEFILE "PART NAME"
        WRITEFILE "QUANTITY"
        WRITEFILE "SUPPLIER CODE"
        WRITEFILE "SUPPLIER NAME"
        WRITEFILE NEWLINE

        k = 0
        DOWHILE (k <= j)
            WRITEFILE parts[k][0]
            WRITEFILE parts[k][1]
            WRITEFILE parts[k][2]
            WRITEFILE parts[k][3]
            WRITEFILE parts[k][4]
            WRITEFILE parts[k][5]
            WRITEFILE parts[k][6]
            WRITEFILE parts[k][7]
            WRITEFILE parts[k][8]
            WRITEFILE parts[k][9]
            WRITEFILE NEWLINE

            k = k + 1
        ENDDO
    CLOSEFILE
```

```
    OPENFILE "suppliers.txt"
        WRITEFILE "CODE"
        WRITEFILE "NAME"
        WRITEFILE NEWLINE
        k = 0
        DOWHILE ( k <= l)
            WRITEFILE suppliers[k][0]
            WRITEFILE suppliers[k][1]
            WRITEFILE NEWLINE

            k = k + 1
        ENDDO
    CLOSEFILE

    parts = CALL freePartList(parts)

    RETURN suppliers, parts
ENDFUNCTION
```

```
FUNCTION deletePart(models, parts)
    IF (CALL length(models) = 0) THEN
        PRINT "No available models. Please add a model first."
        RETURN parts
    ENDIF

    DECLARE i, j, partChoice, filename
    i = CALL getModelCode(models)

    parts = CALL readExistingParts(models[i], parts)

    IF (CALL length(parts) = 0) THEN
        PRINT "No available parts for this model. Please add a part first."
        RETURN parts
    ENDIF

    PRINT "Available Parts:"
    PRINT NEWLINE

    PRINT "PART ID"
    PRINT "MODEL NAME"
    PRINT "SECTION NAME"
    PRINT "PART CODE"
    PRINT "PART NAME"
    PRINT "QUANTITY"
    PRINT "SUPPLIER NAME"
    PRINT NEWLINE

    j = 0
    DOWHILE (j < CALL length(parts))
        PRINT parts[j][0]
        PRINT parts[j][2]
        PRINT parts[j][4]
        PRINT parts[j][5]
        PRINT parts[j][6]
        PRINT parts[j][7]
        PRINT parts[j][9]
        PRINT NEWLINE
        j = j + 1
    ENDDO

    PRINT "Choose part code: "
    ACCEPT partChoice

    j = 0
    DOWHILE (part[j][5] <> partChoice)
        j = j + 1
    ENDDO

    DOWHILE (j < (CALL length(part)))
        part[j] = part[j+1]
        j = j + 1
    ENDDO
```

```
    filename = models[i][0] + ".txt"

    OPENFILE filename FOR WRITE
        FILEWRITE "PART ID"
        FILEWRITE "MODEL CODE"
        FILEWRITE "MODEL NAME"
        FILEWRITE "SECTION CODE"
        FILEWRITE "SECTION NAME"
        FILEWRITE "PART CODE"
        FILEWRITE "PART NAME"
        FILEWRITE "QUANTITY"
        FILEWRITE "SUPPLIER CODE"
        FILEWRITE "SUPPLIER NAME"
        FILEWRITE NEWLINE
        j = 0
        DOWHILE (j < CALL length(parts))
            FILEWRITE parts[j][0]
            FILEWRITE parts[j][1]
            FILEWRITE parts[j][2]
            FILEWRITE parts[j][3]
            FILEWRITE parts[j][4]
            FILEWRITE parts[j][5]
            FILEWRITE parts[j][6]
            FILEWRITE parts[j][7]
            FILEWRITE parts[j][8]
            FILEWRITE parts[j][9]
            FILEWRITE NEWLINE
            j = j + 1
        ENDDO

    CLOSEFILE
    RETURN parts
ENDFUNCTION
```

```
FUNCTION updateInventory(models, parts)
    IF (CALL length(models) = 0) THEN
        PRINT "No available models. Add a model first."
        RETURN parts
    ENDIF

    DECLARE i, j, partCode, filename, qty

    i = CALL getModelCode(models)
    parts = CALL readExistingParts(models[i], parts)

    IF (CALL length(parts) = 0) THEN
        PRINT "No available parts for this model. Add a part first."
        RETURN parts
    ENDIF

    PRINT "Available Parts:"
    PRINT "PART ID"
    PRINT "MODEL NAME"
    PRINT "SECTION NAME"
    PRINT "PART CODE"
    PRINT "PART NAME"
    PRINT "QUANTITY"
    PRINT "SUPPLIER NAME"

    j = 0
    DOWHILE (j < CALL length(parts))
        PRINT parts[j][0]
        PRINT parts[j][2]
        PRINT parts[j][4]
        PRINT parts[j][5]
        PRINT parts[j][6]
        PRINT parts[j][7]
        PRINT parts[j][9]
        PRINT NEWLINE

        j = j + 1
    ENDDO

    PRINT "Choose part code: "
    ACCEPT partCode

    j = 0
    DOWHILE (parts[j][0] <> partCode)
        j = j + 1
    ENDDO
```

```
    PRINT "1. Add quantity"
    PRINT "2. Subtract quantity"
    PRINT "Choose operation: "
    ACCEPT choice
    PRINT "Enter quantity: "
    ACCEPT quantity

    IF (choice = 1)
        parts[j][7] = parts[j][7] + quantity
    ELSE
        parts[j][7] = parts[j][7] - quantity
    ENDIF

    filename = models[i][0] + ".txt"
    OPENFILE filename FOR WRITE
        FILEWRITE "PART ID"
        FILEWRITE "MODEL CODE"
        FILEWRITE "MODEL NAME"
        FILEWRITE "SECTION CODE"
        FILEWRITE "SECTION NAME"
        FILEWRITE "PART CODE"
        FILEWRITE "PART NAME"
        FILEWRITE "QUANTITY"
        FILEWRITE "SUPPLIER CODE"
        FILEWRITE "SUPPLIER NAME"
        FILEWRITE NEWLINE

        j = 0
        DOWHILE (j < CALL length(parts))
            FILEWRITE parts[j][0]
            FILEWRITE parts[j][1]
            FILEWRITE parts[j][2]
            FILEWRITE parts[j][3]
            FILEWRITE parts[j][4]
            FILEWRITE parts[j][5]
            FILEWRITE parts[j][6]
            FILEWRITE parts[j][7]
            FILEWRITE parts[j][8]
            FILEWRITE parts[j][9]
            FILEWRITE NEWLINE
            j = j + 1
        ENDDO
    CLOSEFILE

    parts = CALL freePartList(parts)
    RETURN parts
ENDFUNCTION
```

```
FUNCTION searchInventory(models, suppliers, parts)
    IF (CALL length(models) = 0) THEN
        RETURN
    ENDIF

    DECLARE i, j, k, found

    PRINT "Enter search keyword: "
    ACCEPT keyword

    PRINT "PART ID"
    PRINT "MODEL NAME"
    PRINT "SECTION NAME"
    PRINT "PART CODE"
    PRINT "PART NAME"
    PRINT "QUANTITY"
    PRINT "SUPPLIER NAME"
    PRINT NEWLINE

    i = 0
    DOWHILE (i < CALL length(models))
        parts = CALL readExistingParts(models[i], parts)

        j = 0
        DOWHILE (j < CALL length(parts))
            found = 0
            k = 0
            DOWHILE (k < 10)
                IF (parts[j][k] = keyword) THEN
                    found = 1
                    END THE LOOP
                ENDIF
                k = k + 1
            ENDDO

            IF (found = 1) THEN
                PRINT parts[j][0]
                PRINT parts[j][2]
                PRINT parts[j][4]
                PRINT parts[j][5]
                PRINT parts[j][6]
                PRINT parts[j][7]
                PRINT parts[j][9]
                PRINT NEWLINE
            ENDIF
            j = j + 1
        ENDDO

        parts = CALL freePartList(parts)
        i = i + 1
    ENDDO
    RETURN
ENDFUNCTION
```

```
FUNCTION inventoryTracking(models, suppliers, parts)
    DECLARE i, j, k, qty
    IF (CALL length(models) = 0) THEN
        PRINT "No available models. Add a model first."
        RETURN
    ENDIF

    PRINT "Print Options: "
    PRINT "1. Print all parts from all model"
    PRINT "2. Print all parts from a model"
    PRINT "3. Print all parts from a section of a model"
    PRINT "4. Print all parts from a supplier"
    PRINT "5. Print all parts with low quantity from a model"

    PRINT "Enter your choice: "
    ACCEPT optionChoice

    IF (opetionChoice = 1) THEN
        PRINT "PART ID"
        PRINT "MODEL NAME"
        PRINT "SECTION NAME"
        PRINT "PART CODE"
        PRINT "PART NAME"
        PRINT "QUANTITY"
        PRINT "SUPPLIER NAME"
        PRINT NEWLINE

        i = 0

        DOWHILE (i < CALL length(models))
            j = 0
            parts = CALL readExistingParts(models[i], parts)
            DOWHILE (j < CALL length(parts))
                PRINT part[j][0]
                PRINT part[j][2]
                PRINT part[j][4]
                PRINT part[j][5]
                PRINT part[j][6]
                PRINT part[j][7]
                PRINT part[j][9]
                PRINT NEWLINE
                j = j + 1
            ENDDO

            parts = CALL freePartList(parts)

            i = i + 1
        ENDDO
    ELSE
```

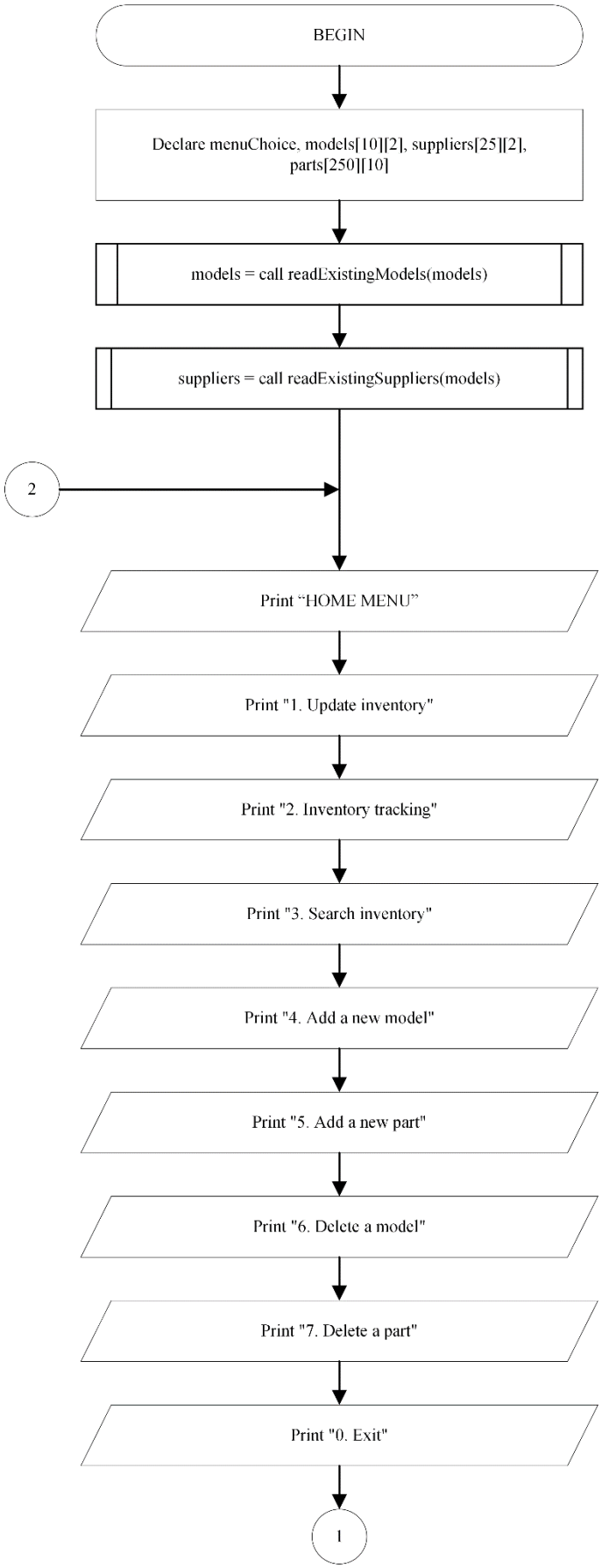
```
        IF (optionChoice = 2) THEN
            i = CALL getModelCode(models)

            parts = CALL readExistingParts(models[i], parts)

            PRINT "PART ID"
            PRINT "MODEL NAME"
            PRINT "SECTION NAME"
            PRINT "PART CODE"
            PRINT "PART NAME"
            PRINT "QUANTITY"
            PRINT "SUPPLIER NAME"
            PRINT NEWLINE

            j = 0
            DOWHILE (j < CALL length(part))
                PRINT parts[j][0]
                PRINT parts[j][2]
                PRINT parts[j][4]
                PRINT parts[j][5]
                PRINT parts[j][6]
                PRINT parts[j][7]
                PRINT parts[j][9]
                PRINT NEWLINE
                j = j + 1
            ENDDO

            parts = CALL freePartList(parts)
        ELSE
```

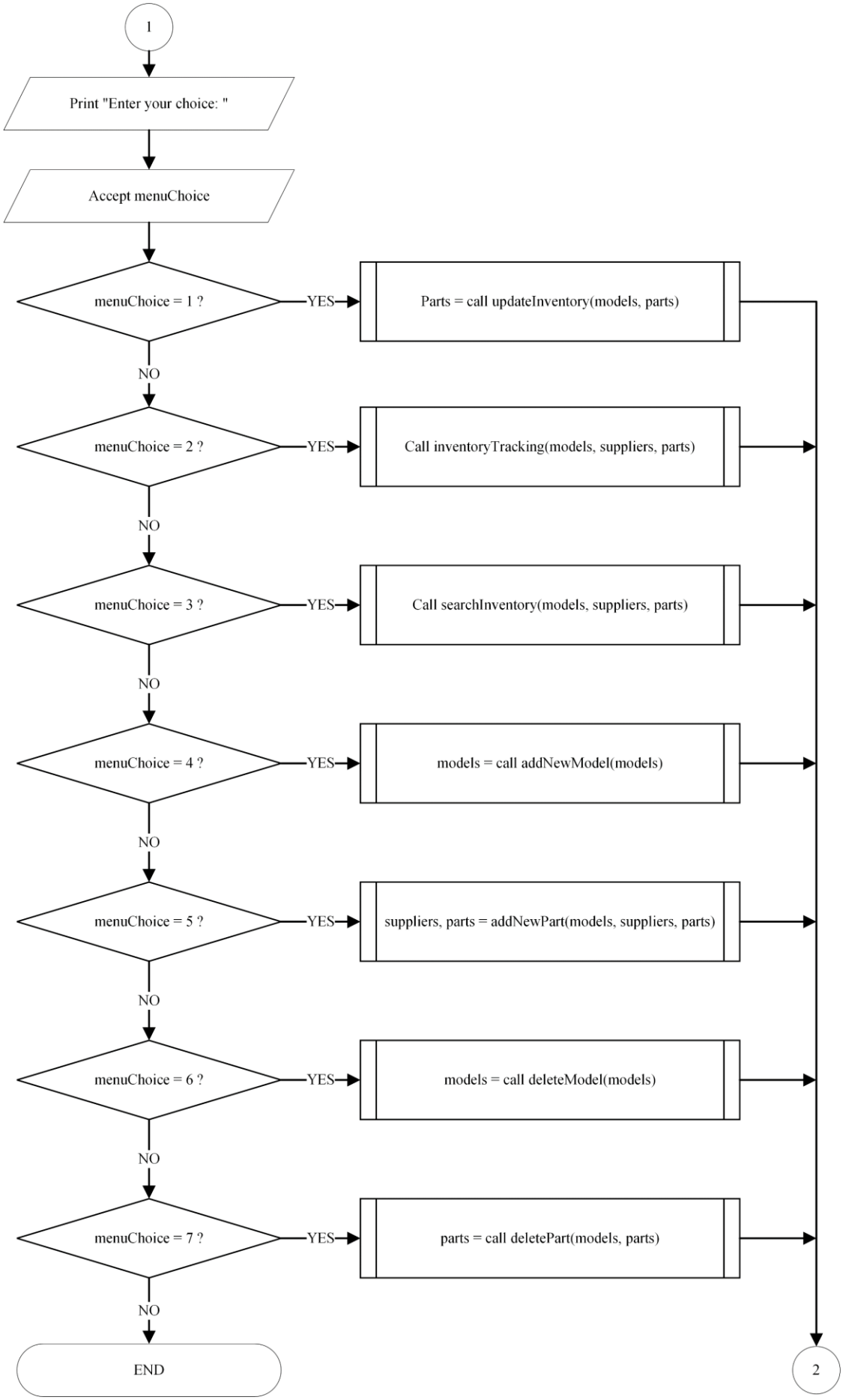
```
        IF (optionChoice = 3) THEN
            i = CALL getModelCode(models)

            parts = CALL readExistingParts(models[i], parts)

            PRINT "Available sections: ");
            PRINT "BD. Body");
            PRINT "CS. Chassis");
            PRINT "EN. Engine");
            PRINT "TR. Transmission System");
            PRINT "OT. Others");

            PRINT "Choose Section Code: "
            ACCEPT sectionChoice

            PRINT "PART ID"
            PRINT "MODEL NAME"
            PRINT "SECTION NAME"
            PRINT "PART CODE"
            PRINT "PART NAME"
            PRINT "QUANTITY"
            PRINT "SUPPLIER NAME"
            PRINT NEWLINE

            j = 0
            DOWHILE (j < CALL length(parts))
                IF (parts[j][3] = sectionChoice) THEN
                    PRINT parts[j][0]
                    PRINT parts[j][2]
                    PRINT parts[j][4]
                    PRINT parts[j][5]
                    PRINT parts[j][6]
                    PRINT parts[j][7]
                    PRINT parts[j][9]
                    PRINT NEWLINE
                ENDIF
                j = j + 1
            ENDDO
            parts = CALL freePartList(parts)
        ELSE
```

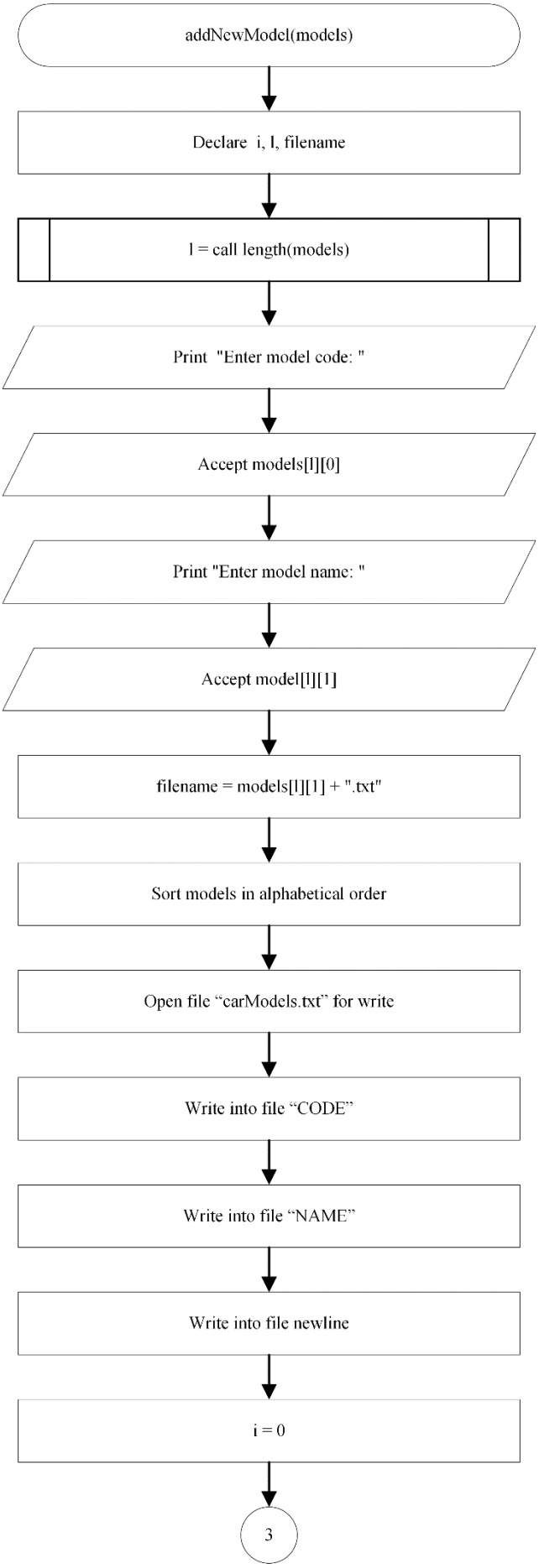
```
                IF (optionChoice = 4) THEN
                    IF (CALL length(supplier) = 0) THEN
                        PRINT "No available suppliers. Add a part from a supplier first."
                        RETURN
                    ENDIF

                    PRINT "Existing Suppliers:"
                    PRINT NEWLINE
                    PRINT "CODE"
                    PRINT "NAME"
                    PRINT NEWLINE
                    i = 0
                    DOWHILE (i < CALL length(supplier))
                        PRINT supplier[i][0]
                        PRINT supplier[i][1]
                        PRINT NEWLINE
                        i = i + 1
                    ENDDO

                    PRINT "Choose supplier code: "
                    ACCEPT supplierChoice

                    PRINT "PART ID"
                    PRINT "MODEL NAME"
                    PRINT "SECTION NAME"
                    PRINT "PART CODE"
                    PRINT "PART NAME"
                    PRINT "QUANTITY"
                    PRINT "SUPPLIER NAME"
                    PRINT NEWLINE

                    j = 0
                    DOWHILE (j < CALL length(models))
                        k = 0
                        parts = CALL readExistingParts(models[i], parts)

                        DOWHILE (k < CALL length(part))
                            IF (parts[k][8] = supplierChoice) THEN
                                PRINT parts[k][0]
                                PRINT parts[k][2]
                                PRINT parts[k][4]
                                PRINT parts[k][5]
                                PRINT parts[k][6]
                                PRINT parts[k][7]
                                PRINT parts[k][9]
                                PRINT NEWLINE
                            ENDIF
                            k = k + 1
                        ENDDO

                        parts = CALL freePartList(parts)
                        j = j + 1
                    ENDDO
                ELSE
```

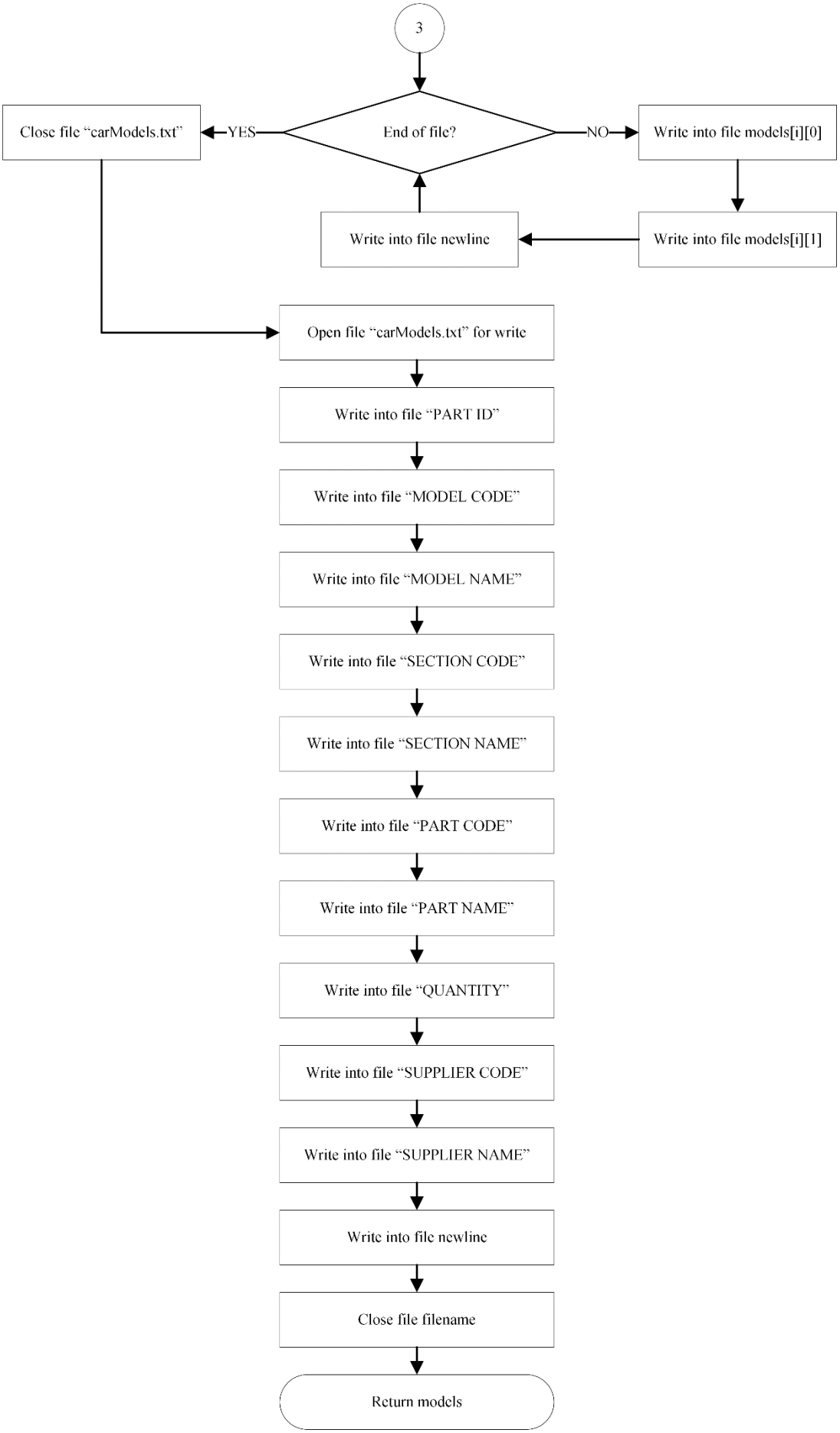
```
                    i = CALL getModelCode(models)
                    parts = CALL readExistingParts(models[i], parts)

                    PRINT "Enter low quantity threshhold: "
                    ACCEPT qty

                    PRINT "PART ID"
                    PRINT "MODEL NAME"
                    PRINT "SECTION NAME"
                    PRINT "PART CODE"
                    PRINT "PART NAME"
                    PRINT "QUANTITY"
                    PRINT "SUPPLIER NAME"
                    PRINT NEWLINE

                    j = 0
                    DOWHILE (j < CALL length(parts))
                        IF (parst[j][7] < qty) THEN
                            PRINT parts[j][0]
                            PRINT parts[j][2]
                            PRINT parts[j][4]
                            PRINT parts[j][5]
                            PRINT parts[j][6]
                            PRINT parts[j][7]
                            PRINT parts[j][9]
                            PRINT NEWLINE
                        ENDIF
                        j = j + 1
                    ENDDO
                    parts = CALL freePartList(parts)
                ENDIF
            ENDIF
        ENDIF
    ENDIF
    RETURN
ENDFUNCTION
```

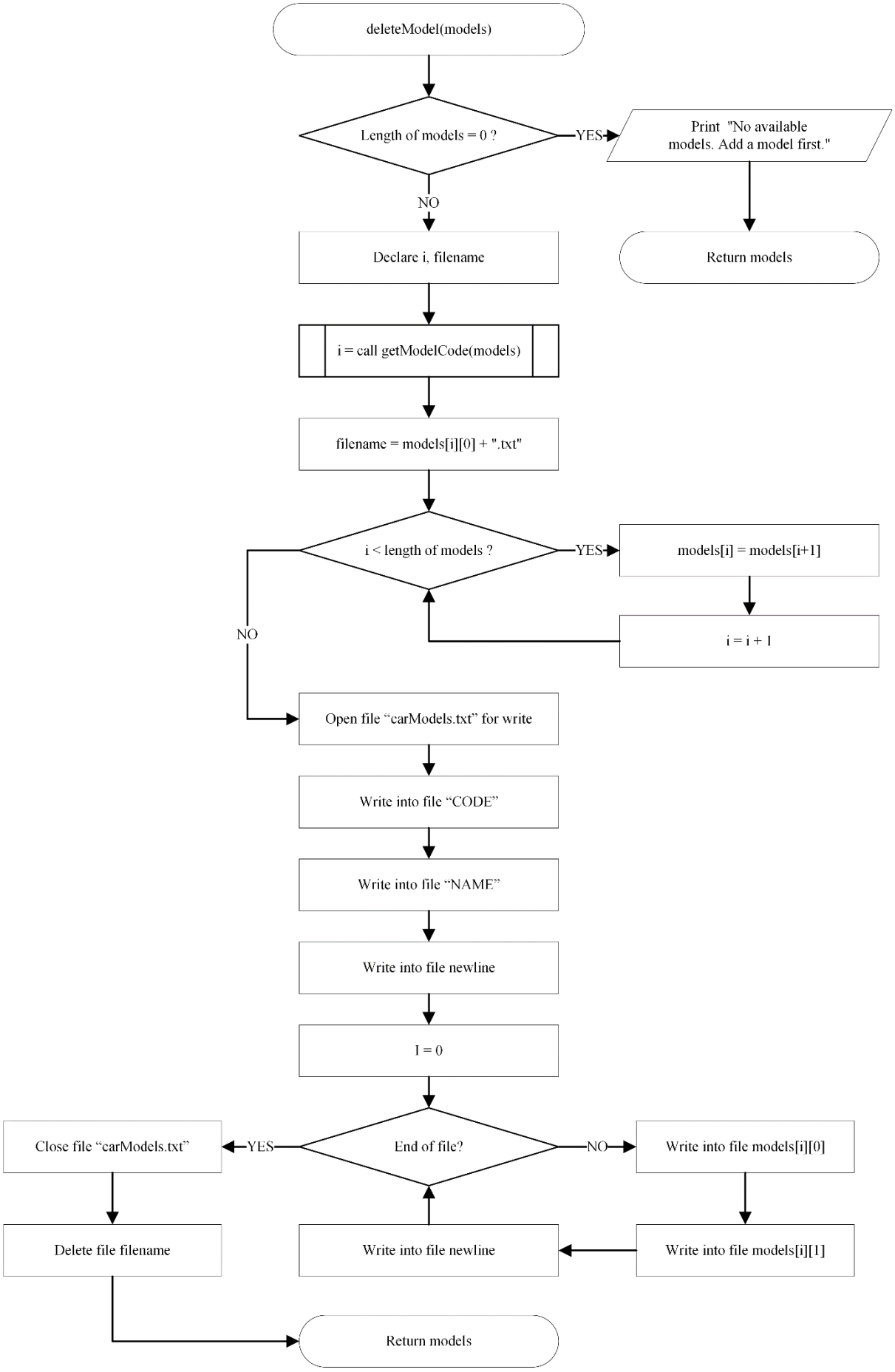
FLOWCHART

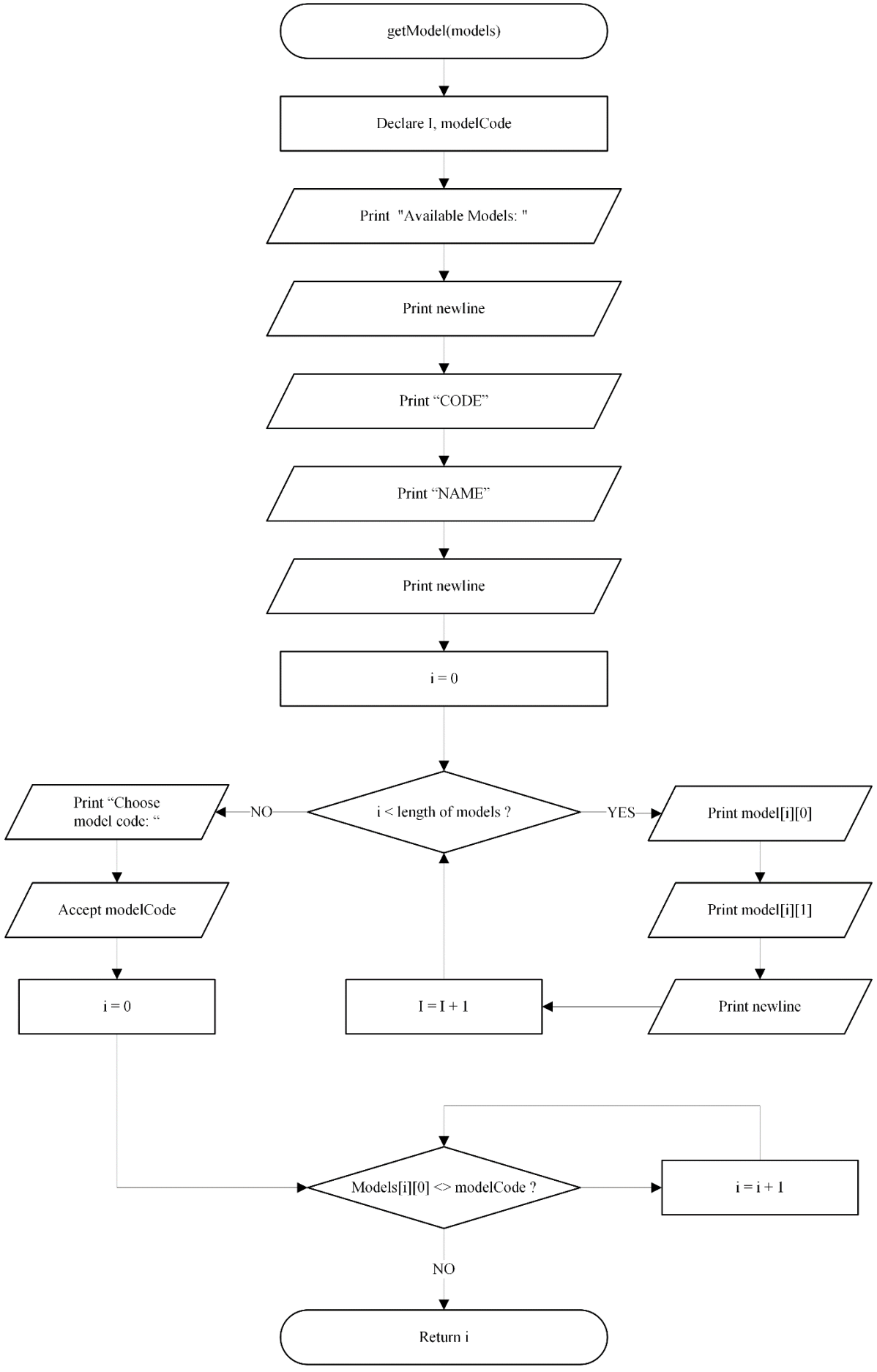

BEGIN
Declare menuChoice, models[10][2], suppliers[25][2], parts[250][10]
models = call readExistingModels(models)
suppliers = call readExistingSuppliers(models)
2
Print “HOME MENU”
Print "1. Update inventory"
Print "2. Inventory tracking"
Print "3. Search inventory"
Print "4. Add a new model"
Print "5. Add a new part"
Print "6. Delete a model"
Print "7. Delete a part"
Print "0. Exit"
1

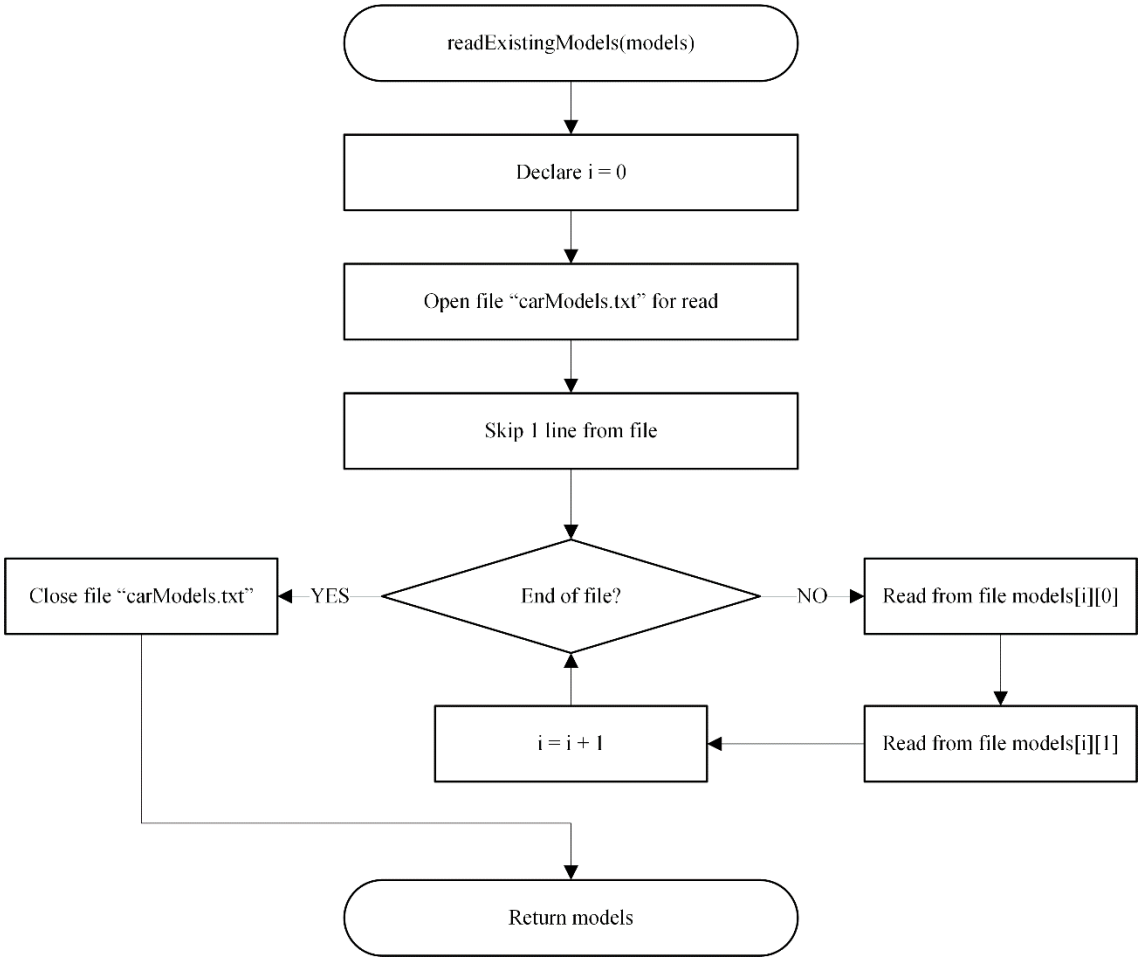

1
Print "Enter your choice: "
Accept menuChoice
menuChoice = 1 ?
YES
Parts = call updateInventory(models, parts)
NO
menuChoice = 2 ?
YES
Call inventoryTracking(models, suppliers, parts)
NO
menuChoice = 3 ?
YES
Call searchInventory(models, suppliers, parts)
NO
menuChoice = 4 ?
YES
models = call addNewModel(models)
NO
menuChoice = 5 ?
YES
suppliers, parts = addNewPart(models, suppliers, parts)
NO
menuChoice = 6 ?
YES
models = call deleteModel(models)
NO
menuChoice = 7 ?
YES
parts = call deletePart(models, parts)
NO
END
2

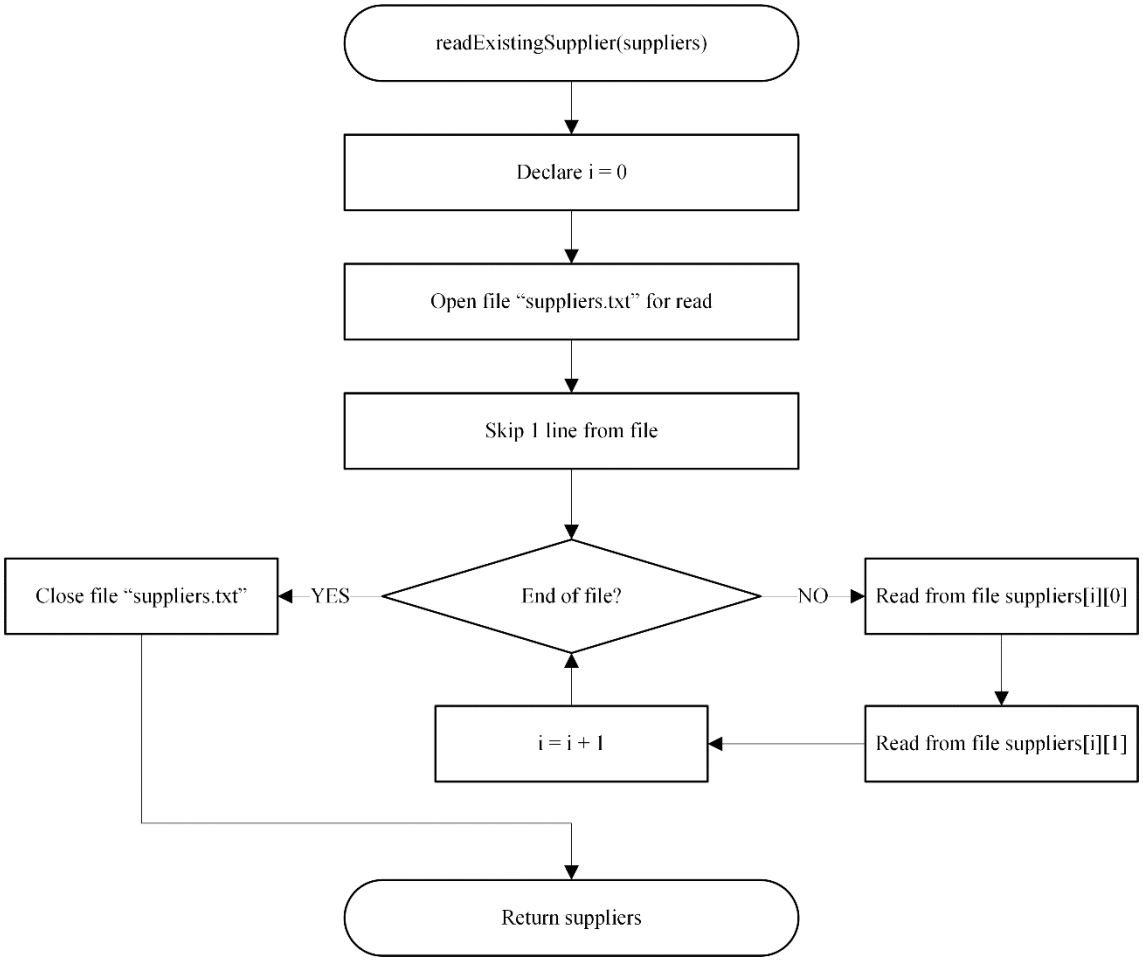

addNewModel(models)
Declare i, l, filename
l = call length(models)
Print "Enter model code: "
Accept models[l][0]
Print "Enter model name: "
Accept model[l][1]
filename = models[l][1] + ".txt"
Sort models in alphabetical order
Open file “carModels.txt” for write
Write into file “CODE”
Write into file “NAME”
Write into file newline
i = 0
3

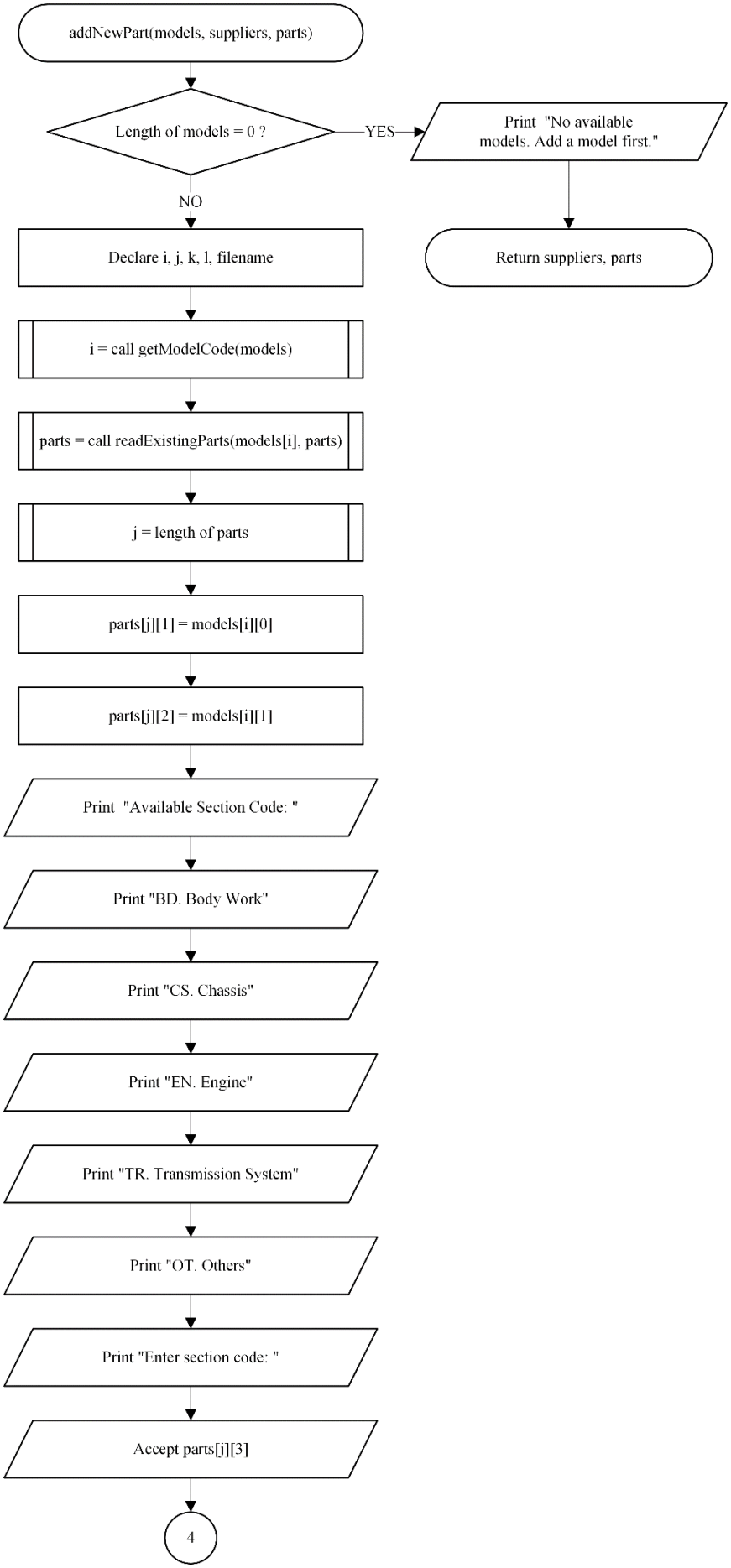

3
End of file?
YES
NO
Close file "carModels.txt"
Write into file models[i][0]
Write into file models[i][1]
Write into file newline
Open file "carModels.txt" for write
Write into file "PART ID"
Write into file "MODEL CODE"
Write into file "MODEL NAME"
Write into file "SECTION CODE"
Write into file "SECTION NAME"
Write into file "PART CODE"
Write into file "PART NAME"
Write into file "QUANTITY"
Write into file "SUPPLIER CODE"
Write into file "SUPPLIER NAME"
Write into file newline
Close file filename
Return models

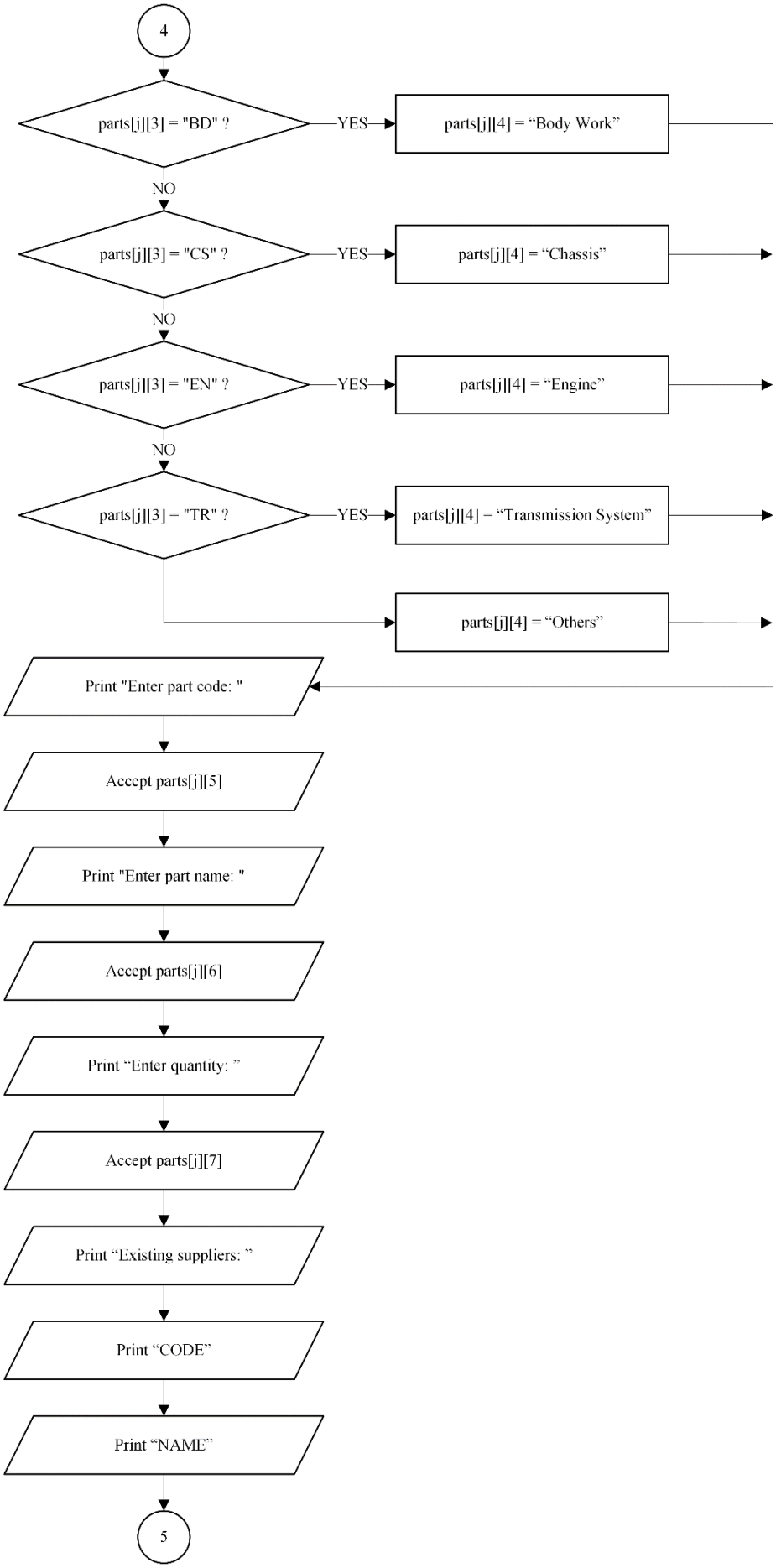

deleteModel(models)
Length of models = 0 ?
YES
Print "No available models. Add a model first."
NO
Return models
Declare i, filename
i = call getModelCode(models)
filename = models[i][0] + ".txt"
i < length of models ?
YES
models[i] = models[i+1]
i = i + 1
NO
Open file “carModels.txt” for write
Write into file “CODE”
Write into file “NAME”
Write into file newline
I = 0
Close file “carModels.txt”
YES
End of file?
NO
Write into file models[i][0]
Delete file filename
Write into file newline
Write into file models[i][1]
Return models

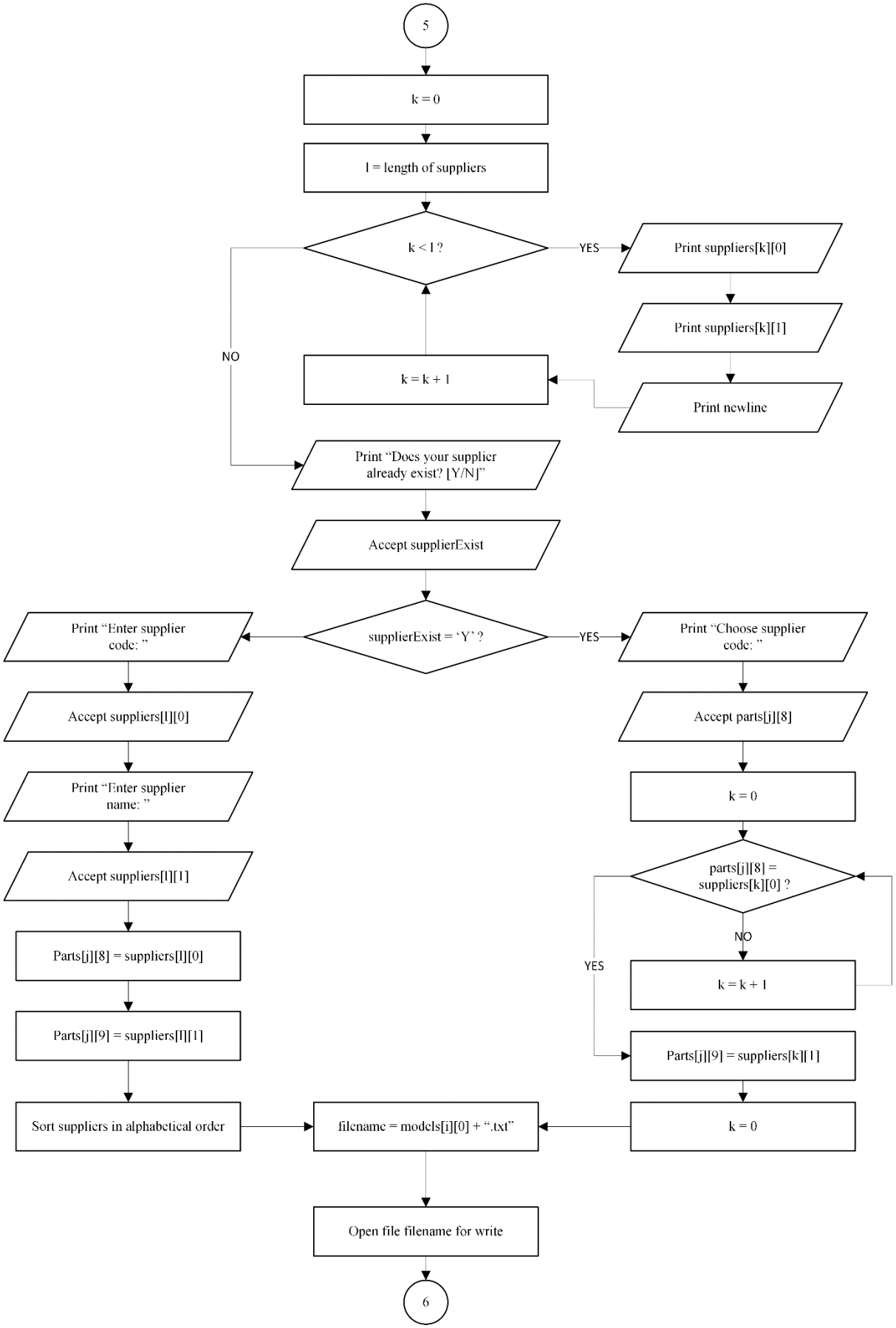

getModel(models)
Declare I, modelCode
Print "Available Models: "
Print newline
Print "CODE"
Print "NAME"
Print newline
i = 0
i < length of models ?
NO
YES
Print "Choose model code: "
Print model[i][0]
Accept modelCode
Print model[i][1]
i = 0
I = I + 1
Print newline
Models[i][0] <> modelCode ?
i = i + 1
NO
Return i

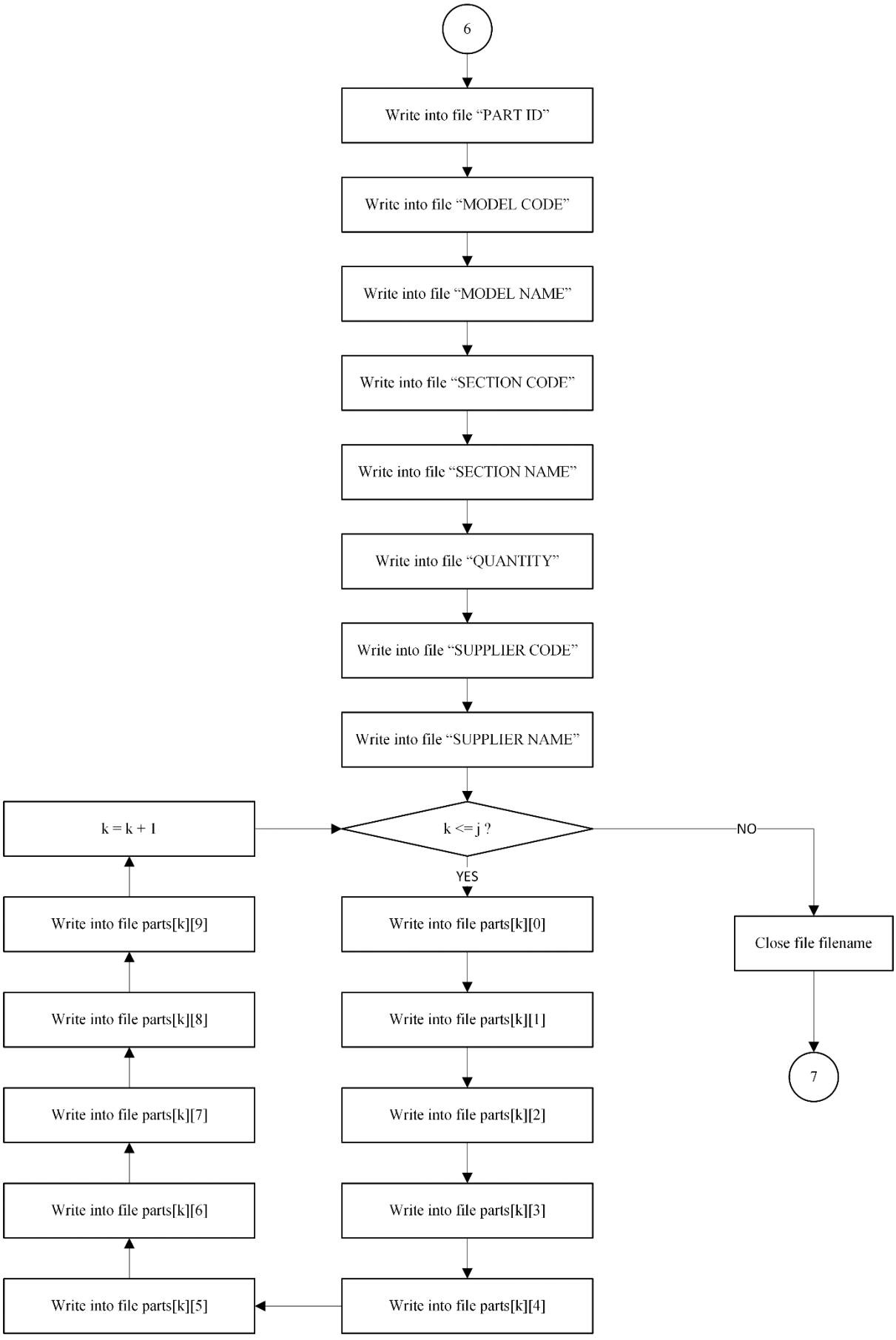

readExistingModels(models)
Declare i = 0
Open file “carModels.txt” for read
Skip 1 line from file
End of file?
YES
Close file “carModels.txt”
NO
Read from file models[i][0]
Read from file models[i][1]
i = i + 1
Return models

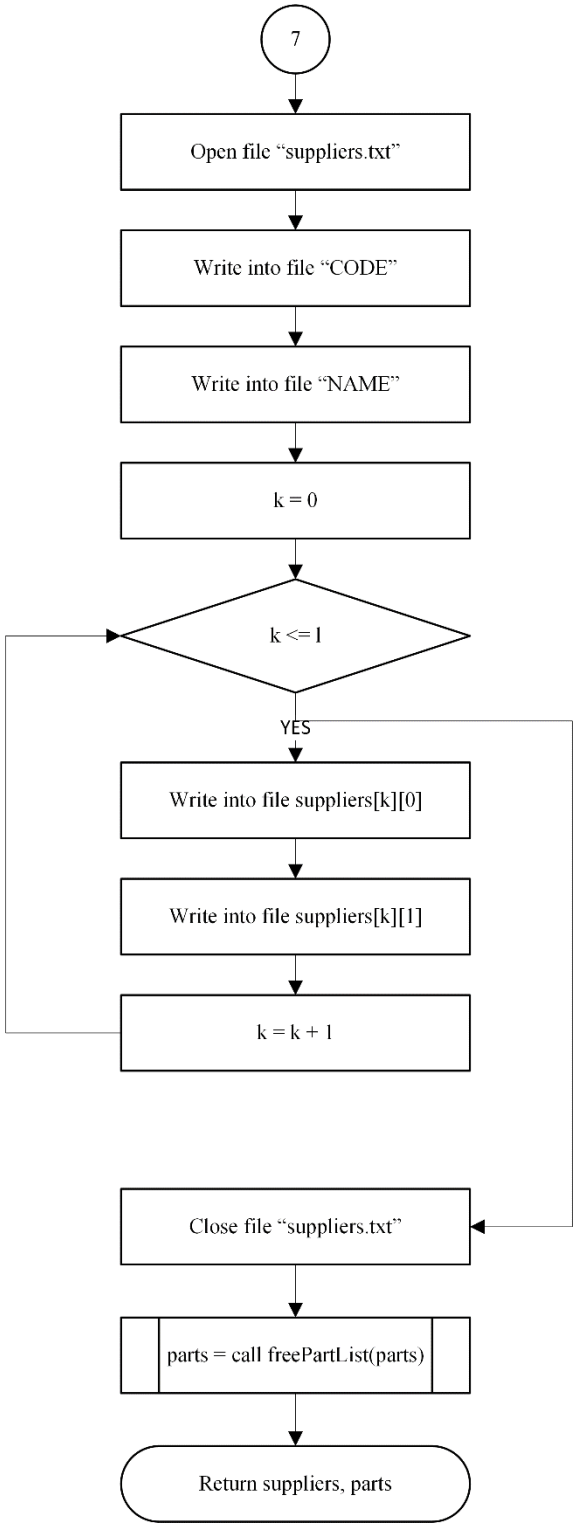

readExistingSupplier(suppliers)
Declare i = 0
Open file "suppliers.txt" for read
Skip 1 line from file
End of file?
YES
Close file "suppliers.txt"
NO
Read from file suppliers[i][0]
Read from file suppliers[i][1]
i = i + 1
Return suppliers

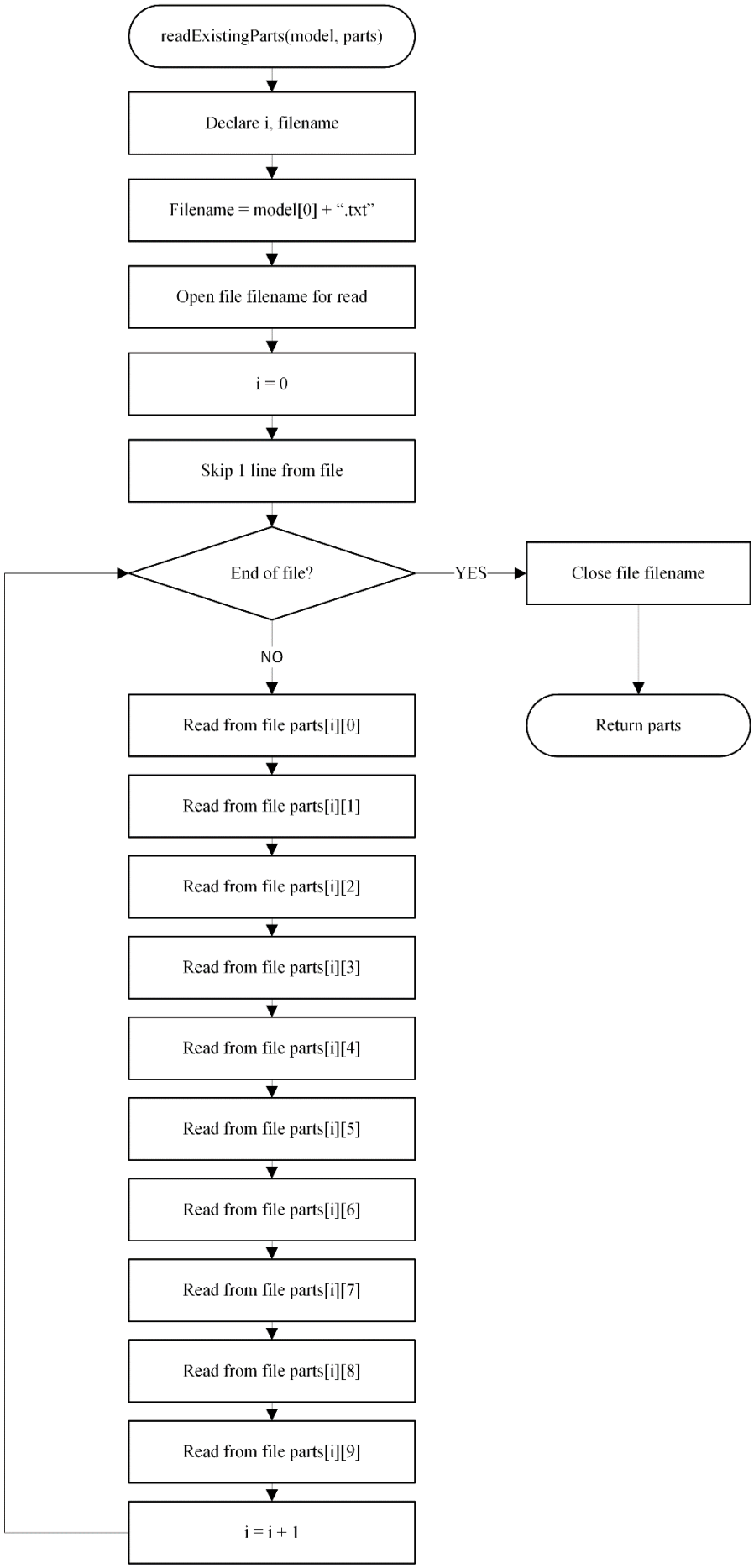

addNewPart(models, suppliers, parts)
Length of models = 0 ?
YES
Print "No available models. Add a model first."
NO
Return suppliers, parts
Declare i, j, k, l, filename
i = call getModelCode(models)
parts = call readExistingParts(models[i], parts)
j = length of parts
parts[j][1] = models[i][0]
parts[j][2] = models[i][1]
Print "Available Section Code: "
Print "BD. Body Work"
Print "CS. Chassis"
Print "EN. Engine"
Print "TR. Transmission System"
Print "OT. Others"
Print "Enter section code: "
Accept parts[j][3]
4

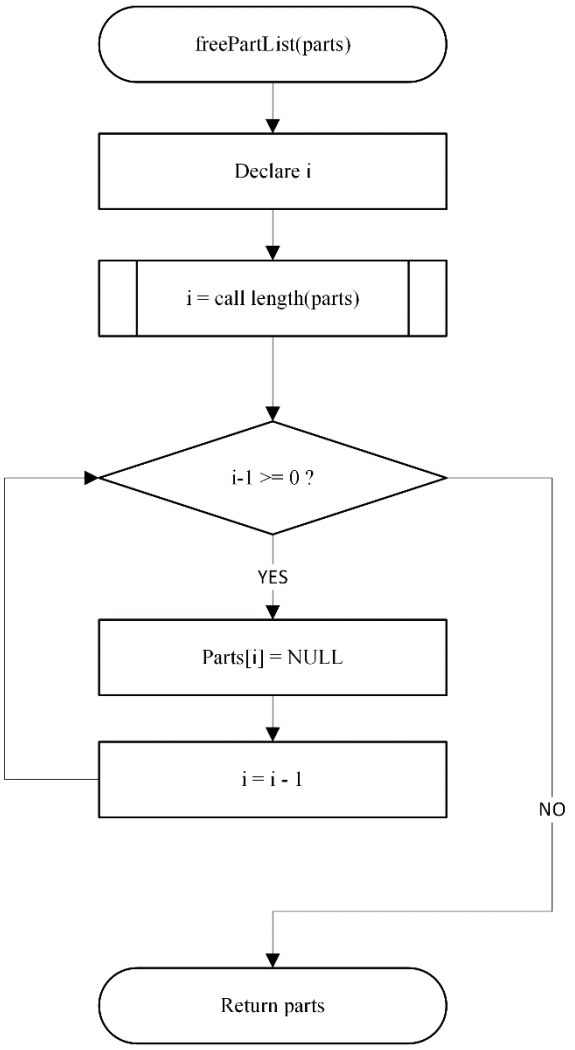

4
parts[j][3] = "BD" ?
YES
parts[j][4] = “Body Work”
NO
parts[j][3] = "CS" ?
YES
parts[j][4] = “Chassis”
NO
parts[j][3] = "EN" ?
YES
parts[j][4] = “Engine”
NO
parts[j][3] = "TR" ?
YES
parts[j][4] = “Transmission System”
parts[j][4] = “Others”
Print "Enter part code: "
Accept parts[j][5]
Print "Enter part name: "
Accept parts[j][6]
Print “Enter quantity: ”
Accept parts[j][7]
Print “Existing suppliers: ”
Print “CODE”
Print “NAME”
5

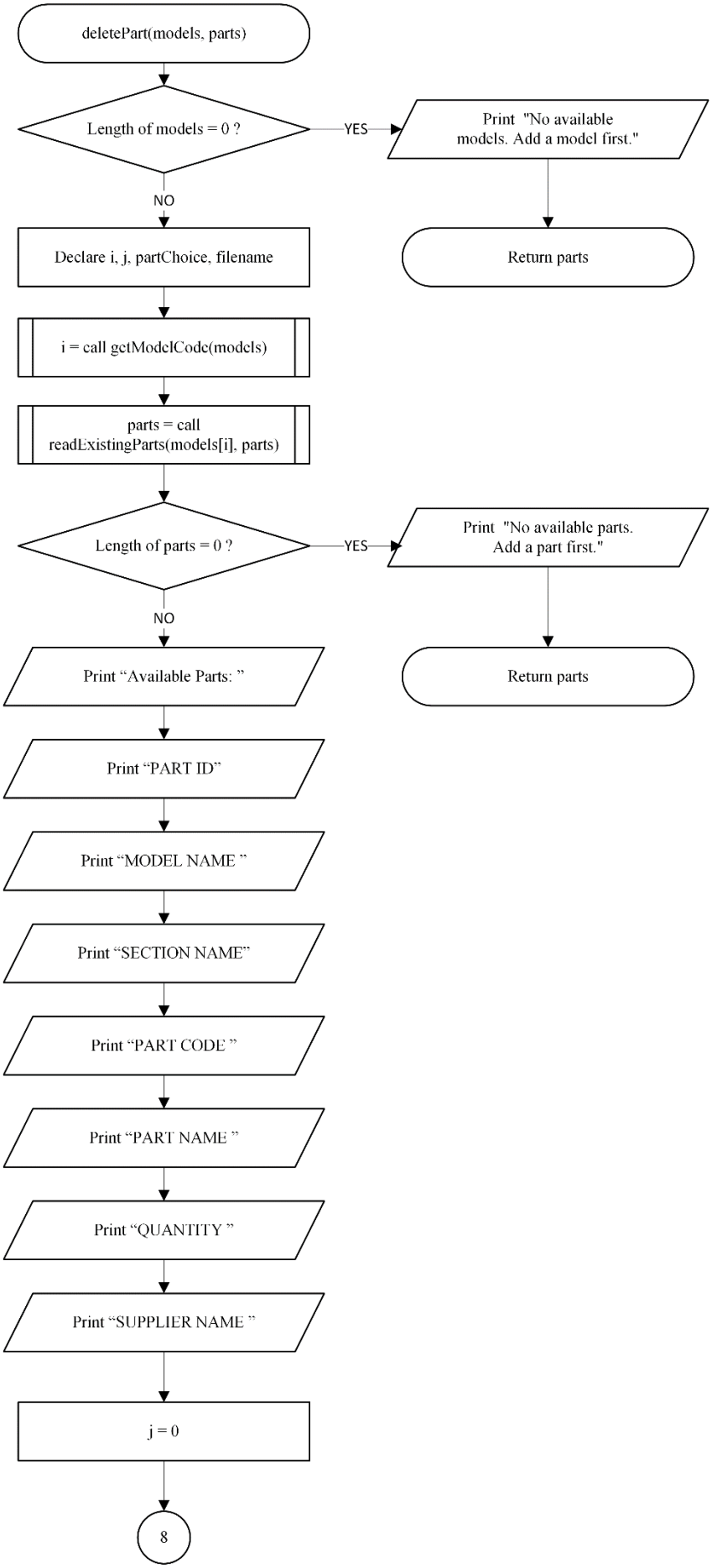

5
k = 0
l = length of suppliers
k < l ?
YES
Print suppliers[k][0]
Print suppliers[k][1]
Print newline
k = k + 1
NO
Print "Does your supplier already exist? [Y/N]"
Accept supplierExist
supplierExist = 'Y' ?
YES
Print "Choose supplier code: "
Accept parts[j][8]
k = 0
parts[j][8] = suppliers[k][0] ?
NO
YES
k = k + 1
Parts[j][9] = suppliers[k][1]
k = 0
Print "Enter supplier code: "
Accept suppliers[l][0]
Print "Enter supplier name: "
Accept suppliers[l][1]
Parts[j][8] = suppliers[l][0]
Parts[j][9] = suppliers[l][1]
Sort suppliers in alphabetical order
filename = models[i][0] + ".txt"
Open file filename for write
6

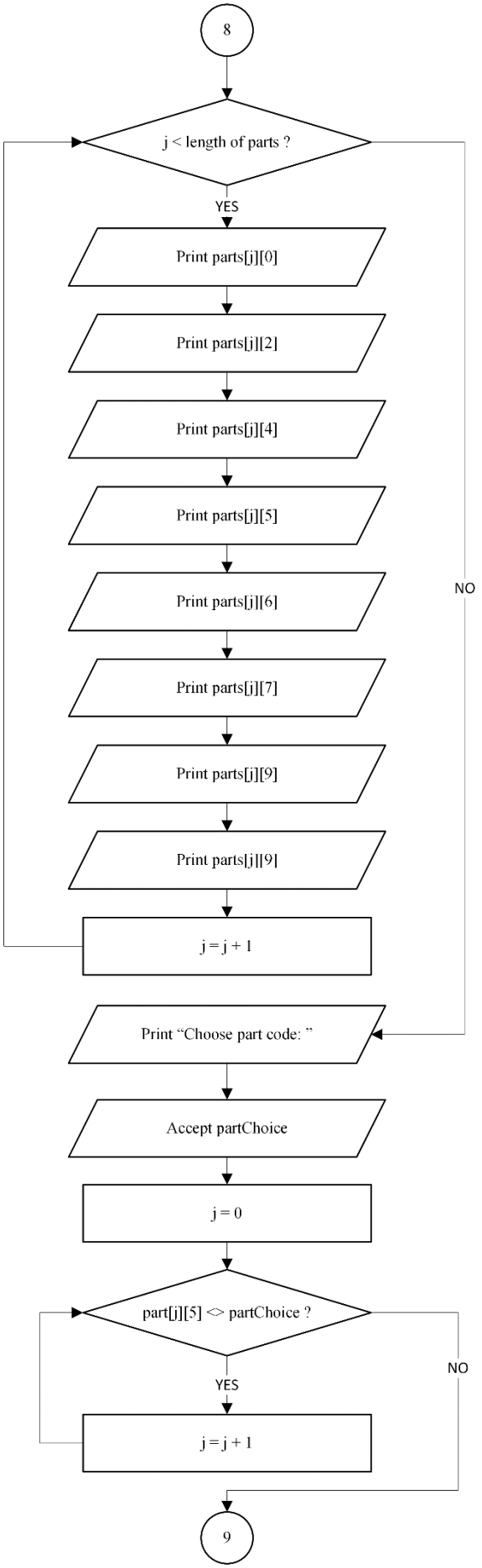

6
Write into file "PART ID"
Write into file "MODEL CODE"
Write into file "MODEL NAME"
Write into file "SECTION CODE"
Write into file "SECTION NAME"
Write into file "QUANTITY"
Write into file "SUPPLIER CODE"
Write into file "SUPPLIER NAME"
k <= j ?
NO
YES
Close file filename
7
Write into file parts[k][0]
Write into file parts[k][1]
Write into file parts[k][2]
Write into file parts[k][3]
Write into file parts[k][4]
Write into file parts[k][5]
Write into file parts[k][6]
Write into file parts[k][7]
Write into file parts[k][8]
Write into file parts[k][9]
k = k + 1

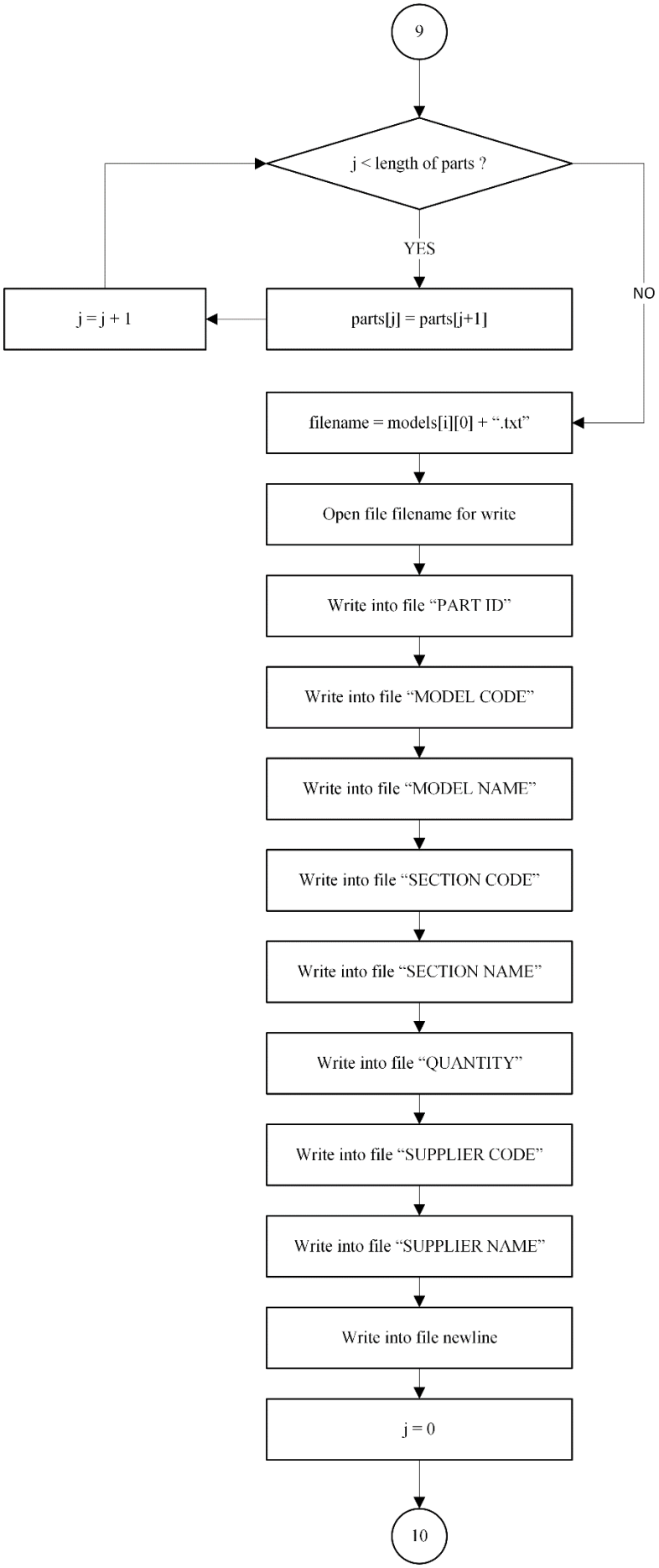

7
Open file “suppliers.txt”
Write into file “CODE”
Write into file “NAME”
k = 0
k <= l
YES
Write into file suppliers[k][0]
Write into file suppliers[k][1]
k = k + 1
Close file “suppliers.txt”
parts = call freePartList(parts)
Return suppliers, parts

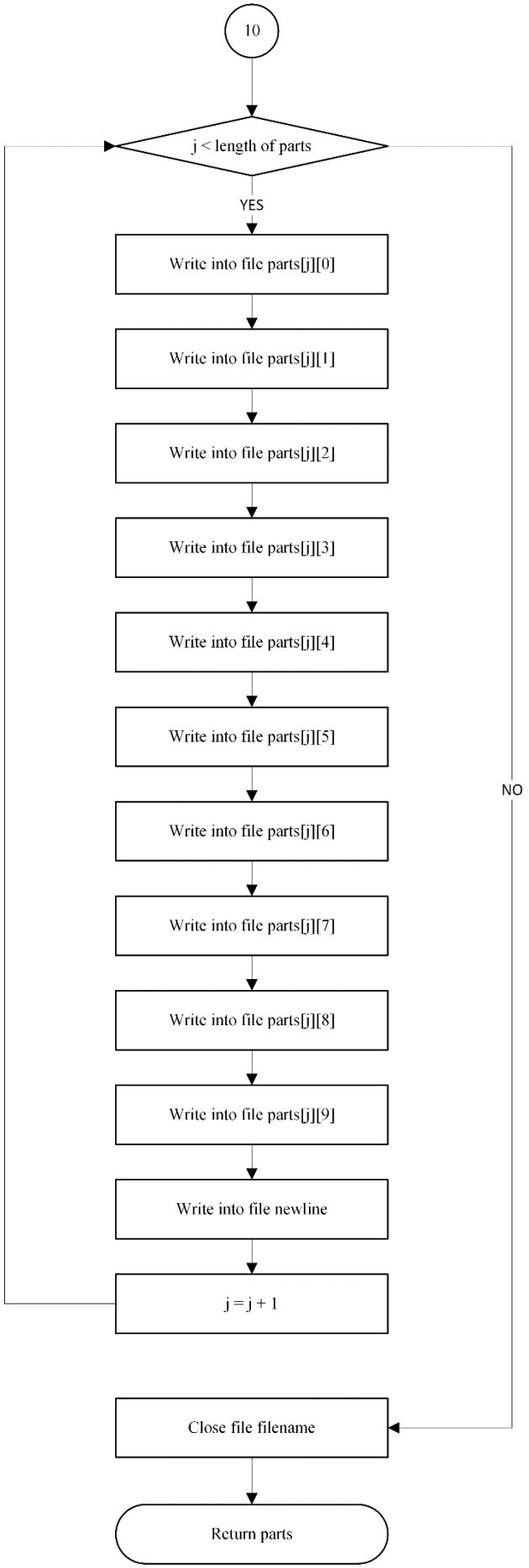

readExistingParts(model, parts)
Declare i, filename
Filename = model[0] + ".txt"
Open file filename for read
i = 0
Skip 1 line from file
End of file?
YES
Close file filename
Return parts
NO
Read from file parts[i][0]
Read from file parts[i][1]
Read from file parts[i][2]
Read from file parts[i][3]
Read from file parts[i][4]
Read from file parts[i][5]
Read from file parts[i][6]
Read from file parts[i][7]
Read from file parts[i][8]
Read from file parts[i][9]
i = i + 1

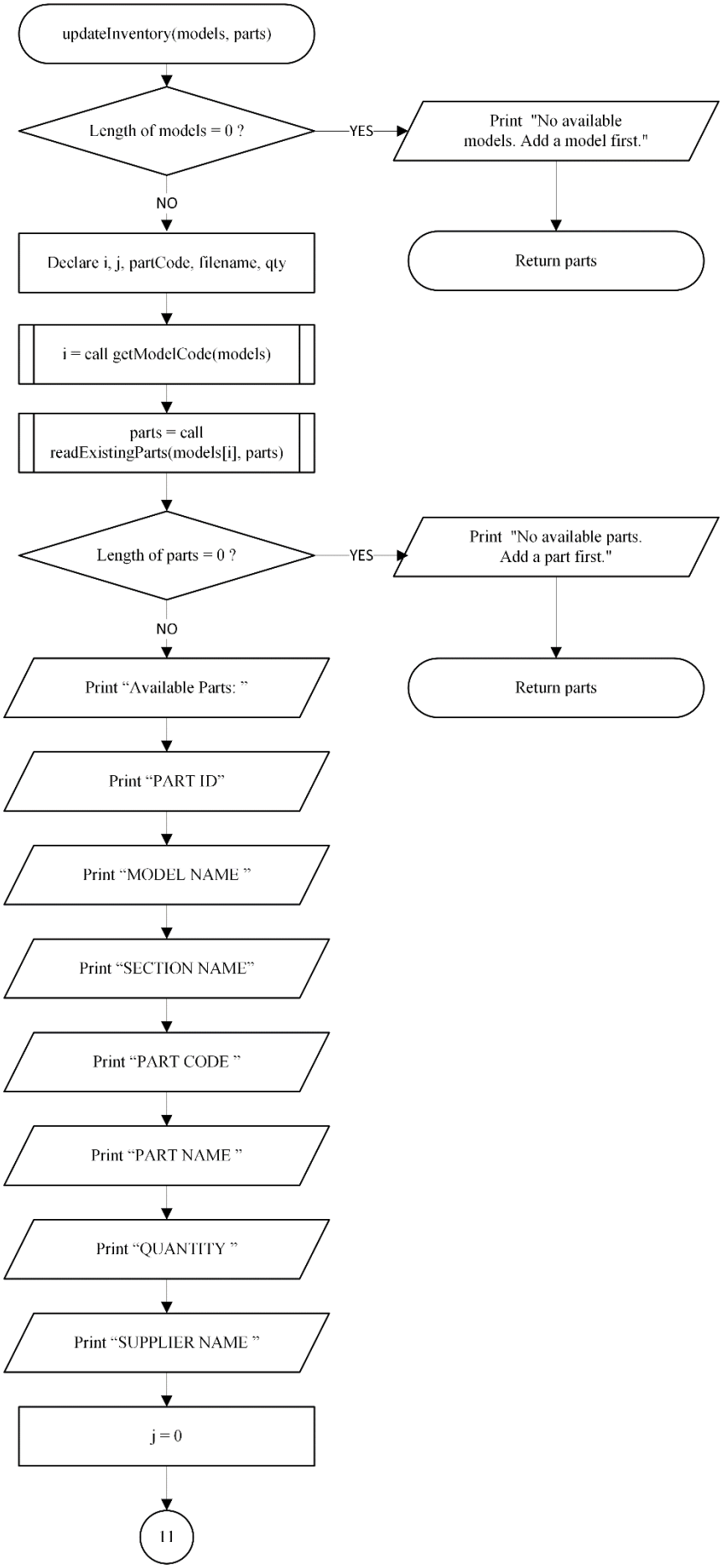

freePartList(parts)
Declare i
i = call length(parts)
i-1 >= 0 ?
YES
Parts[i] = NULL
i = i - 1
NO
Return parts

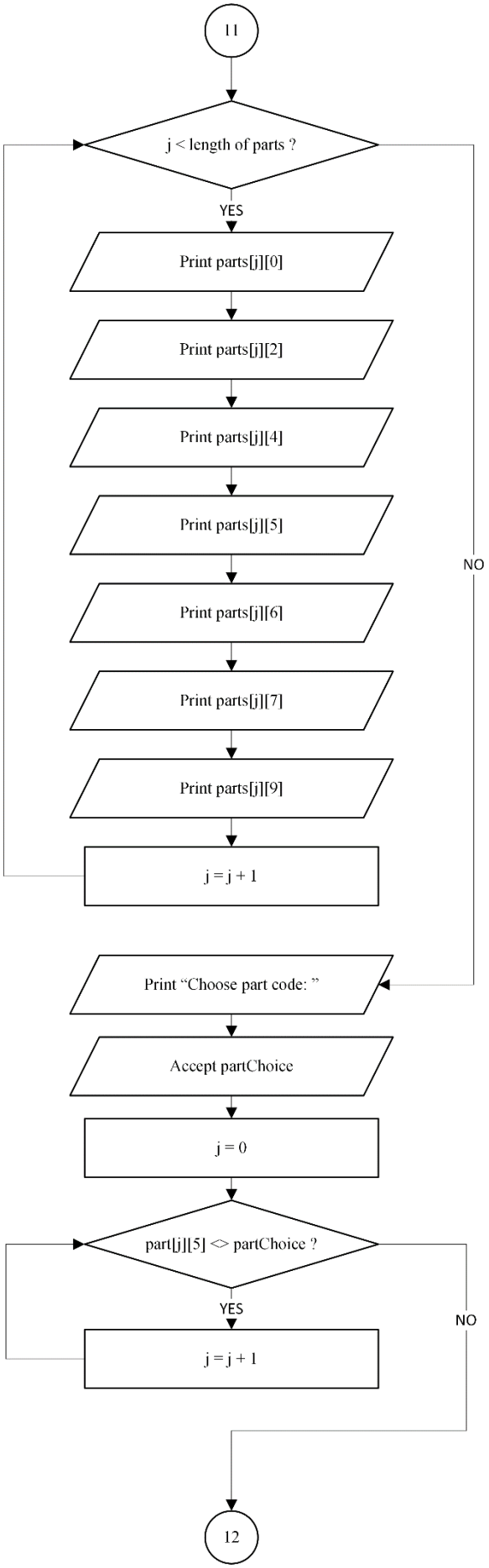

deletePart(models, parts)
Length of models = 0 ?
YES
Print "No available models. Add a model first."
NO
Return parts
Declare i, j, partChoice, filename
i = call getModelCode(models)
parts = call readExistingParts(models[i], parts)
Length of parts = 0 ?
YES
Print "No available parts. Add a part first."
NO
Return parts
Print "Available Parts: "
Print "PART ID"
Print "MODEL NAME "
Print "SECTION NAME"
Print "PART CODE "
Print "PART NAME "
Print "QUANTITY "
Print "SUPPLIER NAME "
j = 0
8

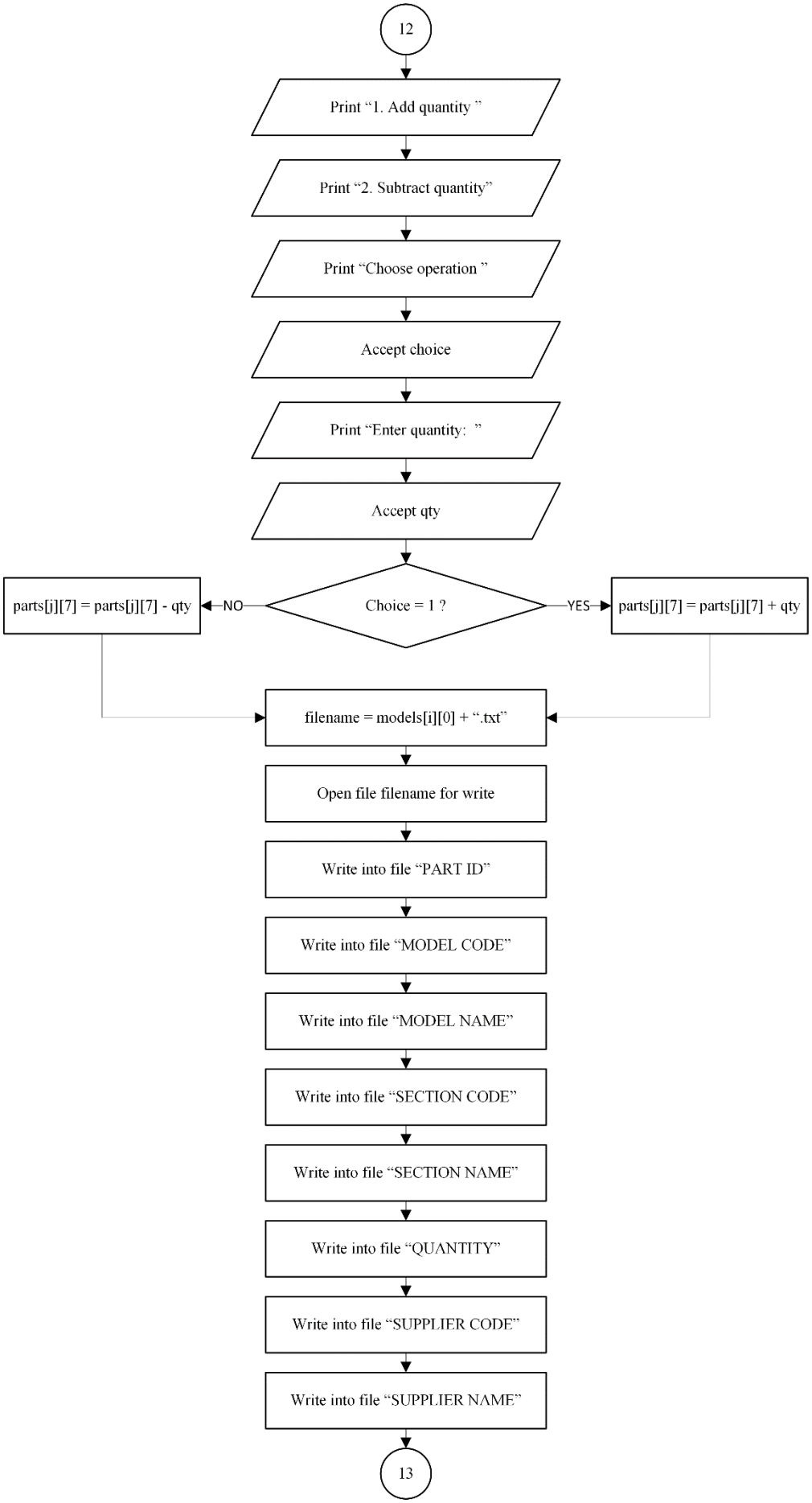

8
j < length of parts ?
YES
NO
Print parts[j][0]
Print parts[j][2]
Print parts[j][4]
Print parts[j][5]
Print parts[j][6]
Print parts[j][7]
Print parts[j][9]
Print parts[j][9]
j = j + 1
Print “Choose part code: ”
Accept partChoice
j = 0
part[j][5] <> partChoice ?
YES
NO
j = j + 1
9

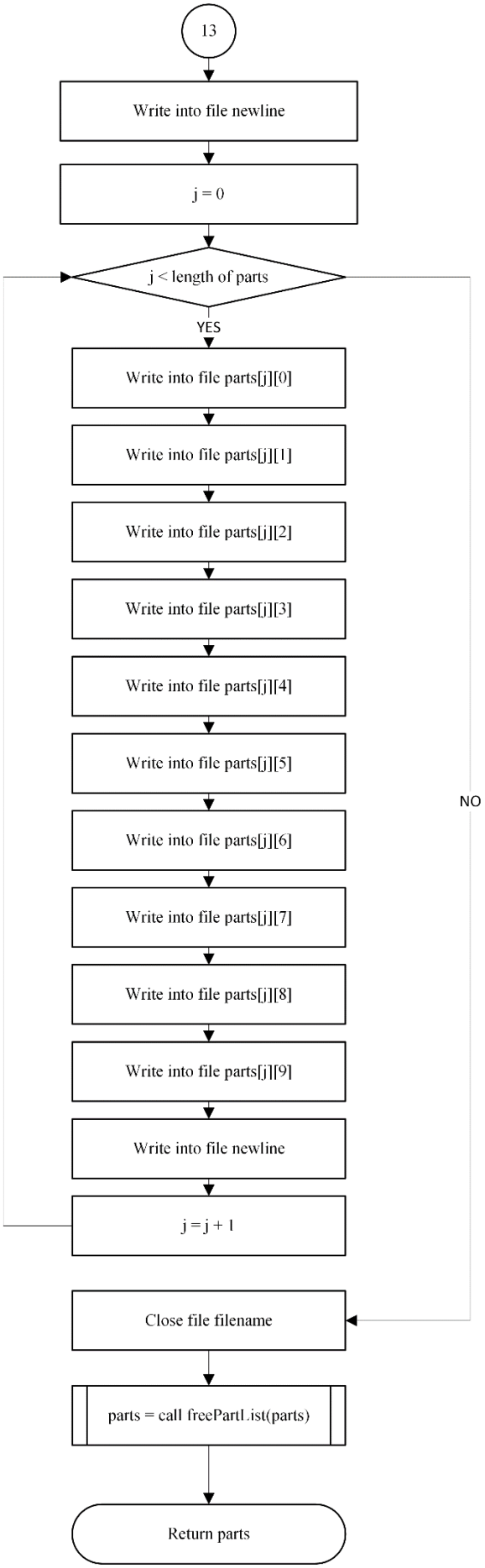

9
j < length of parts ?
YES
NO
parts[j] = parts[j+1]
j = j + 1
filename = models[i][0] + ".txt"
Open file filename for write
Write into file "PART ID"
Write into file "MODEL CODE"
Write into file "MODEL NAME"
Write into file "SECTION CODE"
Write into file "SECTION NAME"
Write into file "QUANTITY"
Write into file "SUPPLIER CODE"
Write into file "SUPPLIER NAME"
Write into file newline
j = 0
10

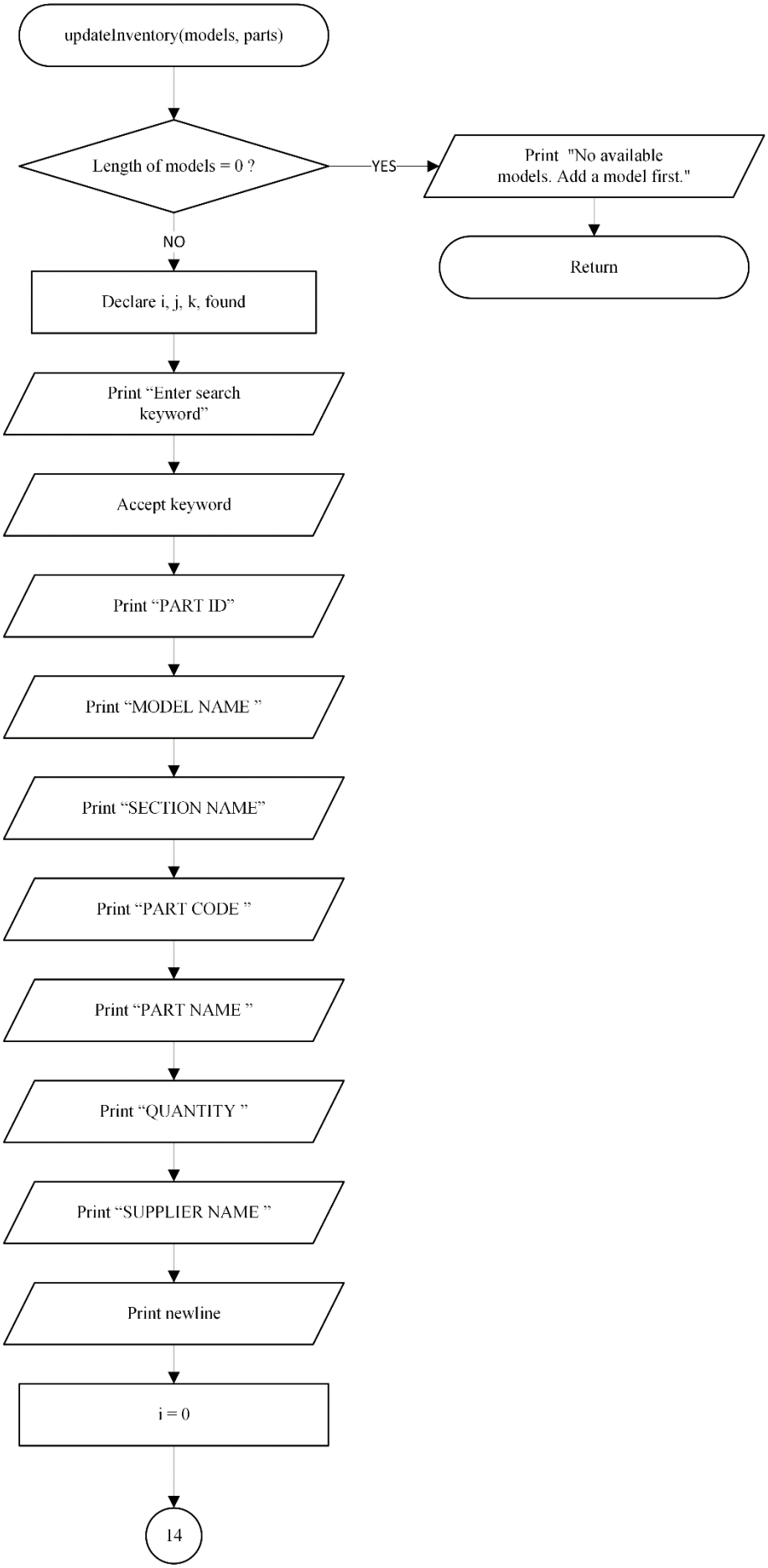

10
j < length of parts
YES
NO
Write into file parts[j][0]
Write into file parts[j][1]
Write into file parts[j][2]
Write into file parts[j][3]
Write into file parts[j][4]
Write into file parts[j][5]
Write into file parts[j][6]
Write into file parts[j][7]
Write into file parts[j][8]
Write into file parts[j][9]
Write into file newline
j = j + 1
Close file filename
Return parts

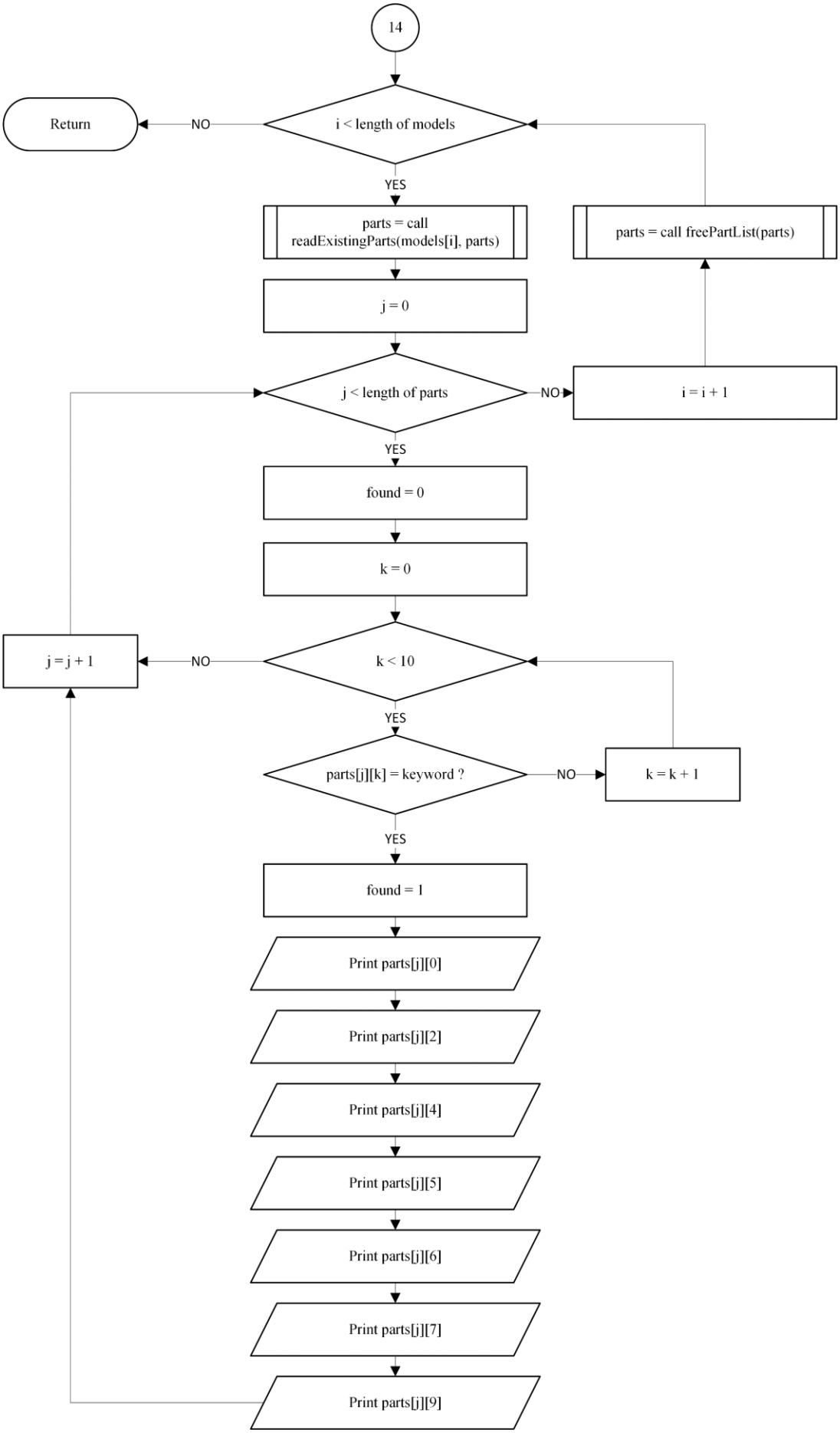

updateInventory(models, parts)
Length of models = 0 ?
YES
Print "No available models. Add a model first."
Return parts
NO
Declare i, j, partCode, filename, qty
i = call getModelCode(models)
parts = call readExistingParts(models[i], parts)
Length of parts = 0 ?
YES
Print "No available parts. Add a part first."
Return parts
NO
Print "Available Parts: "
Print "PART ID"
Print "MODEL NAME "
Print "SECTION NAME"
Print "PART CODE "
Print "PART NAME "
Print "QUANTITY "
Print "SUPPLIER NAME "
j = 0
11

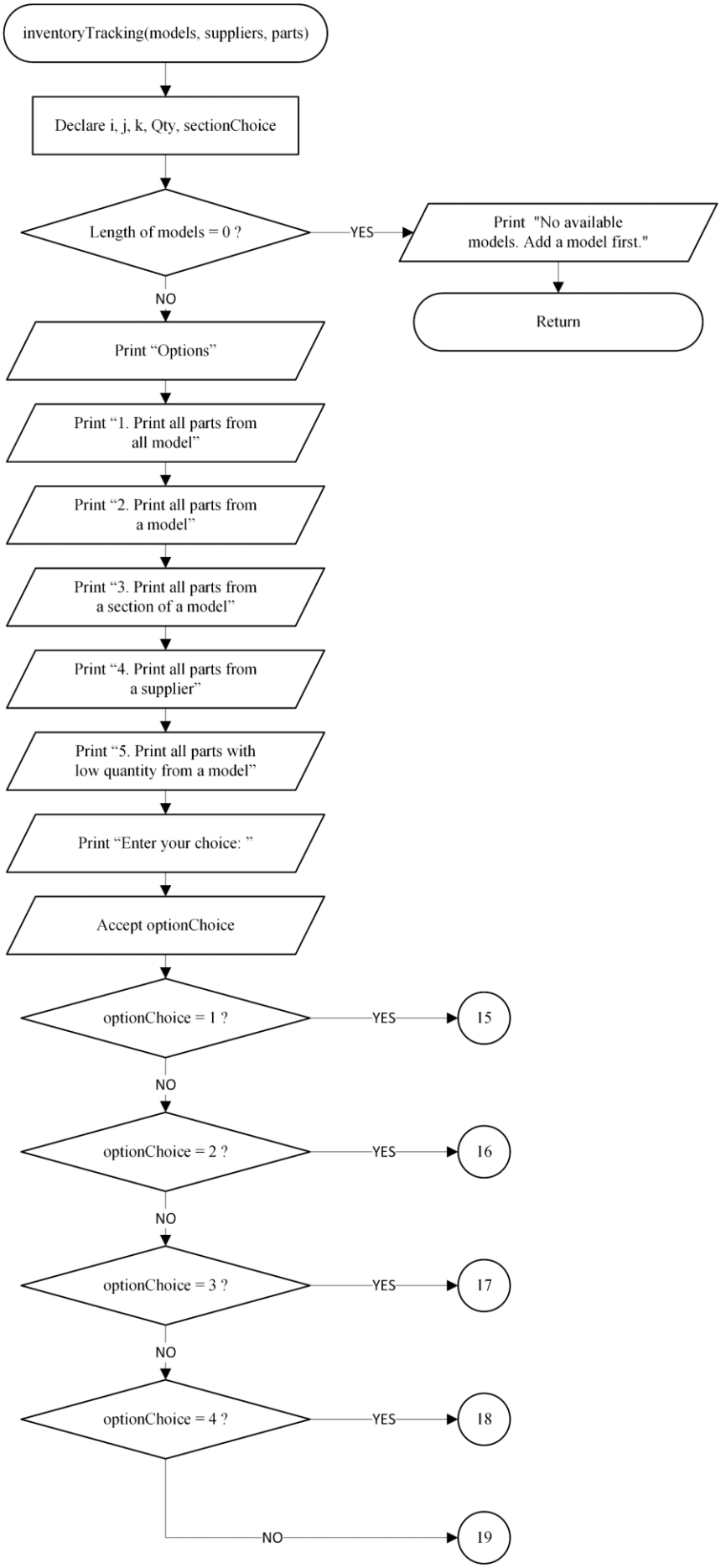

11
j < length of parts ?
YES
NO
Print parts[j][0]
Print parts[j][2]
Print parts[j][4]
Print parts[j][5]
Print parts[j][6]
Print parts[j][7]
Print parts[j][9]
j = j + 1
Print "Choose part code: "
Accept partChoice
j = 0
part[j][5] <> partChoice ?
YES
NO
j = j + 1
12

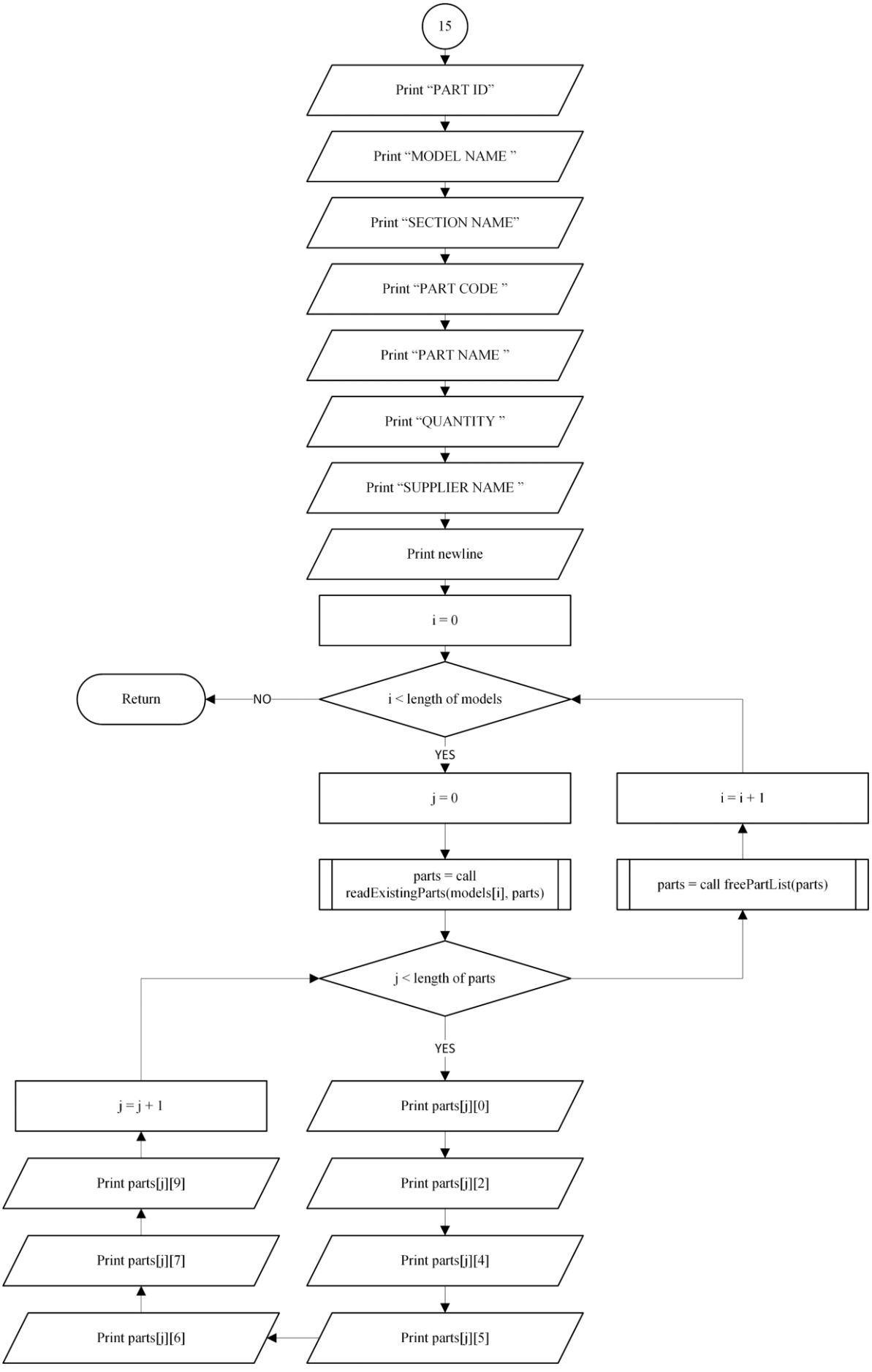

12
Print "1. Add quantity "
Print "2. Subtract quantity"
Print "Choose operation "
Accept choice
Print "Enter quantity: "
Accept qty
Choice = 1 ?
NO
parts[j][7] = parts[j][7] - qty
YES
parts[j][7] = parts[j][7] + qty
filename = models[i][0] + ".txt"
Open file filename for write
Write into file "PART ID"
Write into file "MODEL CODE"
Write into file "MODEL NAME"
Write into file "SECTION CODE"
Write into file "SECTION NAME"
Write into file "QUANTITY"
Write into file "SUPPLIER CODE"
Write into file "SUPPLIER NAME"
13

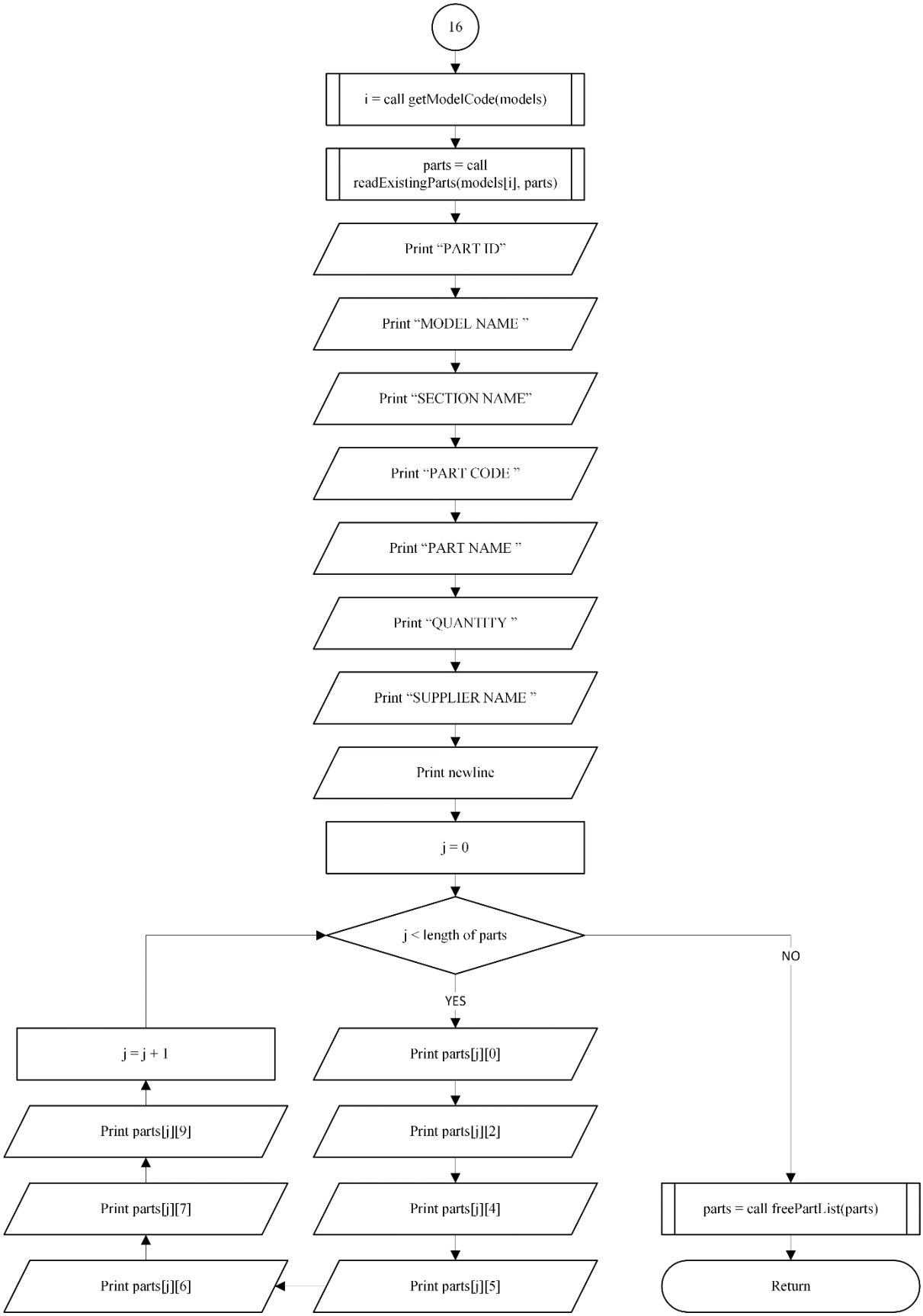

13
Write into file newline
j = 0
j < length of parts
YES
NO
Write into file parts[j][0]
Write into file parts[j][1]
Write into file parts[j][2]
Write into file parts[j][3]
Write into file parts[j][4]
Write into file parts[j][5]
Write into file parts[j][6]
Write into file parts[j][7]
Write into file parts[j][8]
Write into file parts[j][9]
Write into file newline
j = j + 1
Close file filename
parts = call freePartList(parts)
Return parts

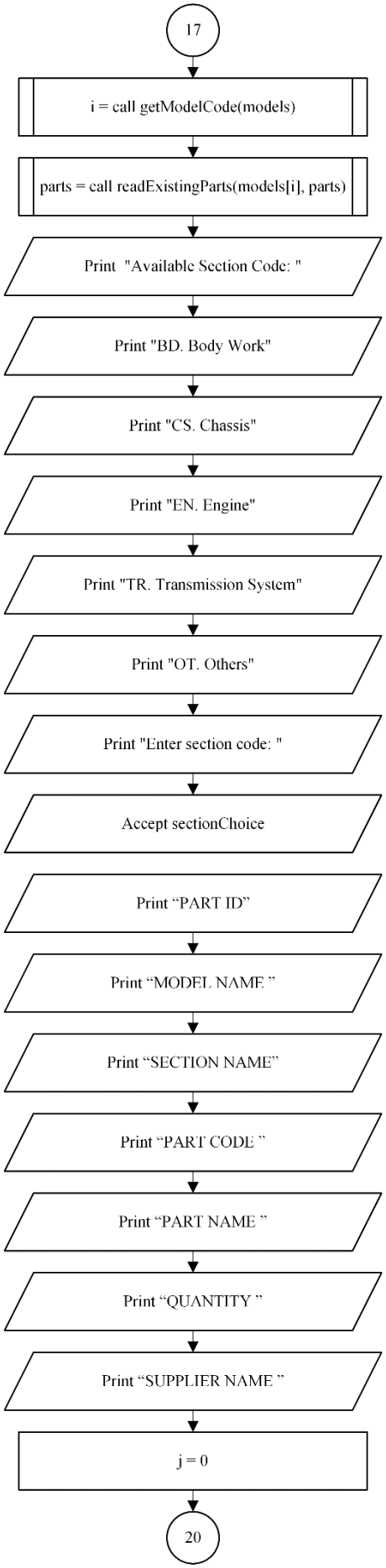

updateInventory(models, parts)
Length of models = 0 ?
YES
Print "No available models. Add a model first."
Return
NO
Declare i, j, k, found
Print “Enter search keyword”
Accept keyword
Print “PART ID”
Print “MODEL NAME ”
Print “SECTION NAME”
Print “PART CODE ”
Print “PART NAME ”
Print “QUANTITY ”
Print “SUPPLIER NAME ”
Print newline
i = 0
14

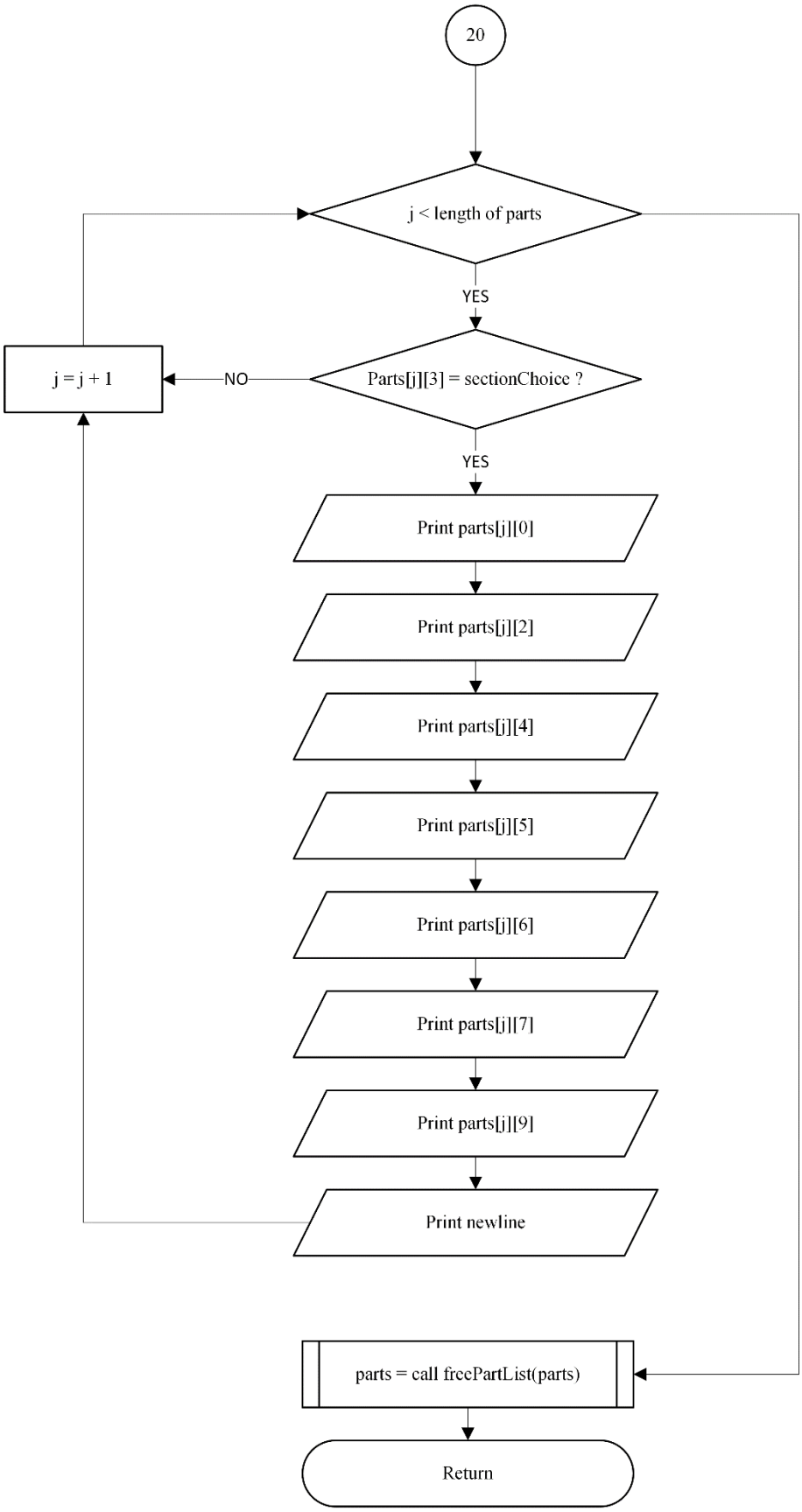

14
i < length of models
NO
Return
YES
parts = call readExistingParts(models[i], parts)
parts = call freePartList(parts)
j = 0
j < length of parts
NO
i = i + 1
YES
found = 0
k = 0
k < 10
NO
j = j + 1
YES
parts[j][k] = keyword ?
NO
k = k + 1
YES
found = 1
Print parts[j][0]
Print parts[j][2]
Print parts[j][4]
Print parts[j][5]
Print parts[j][6]
Print parts[j][7]
Print parts[j][9]

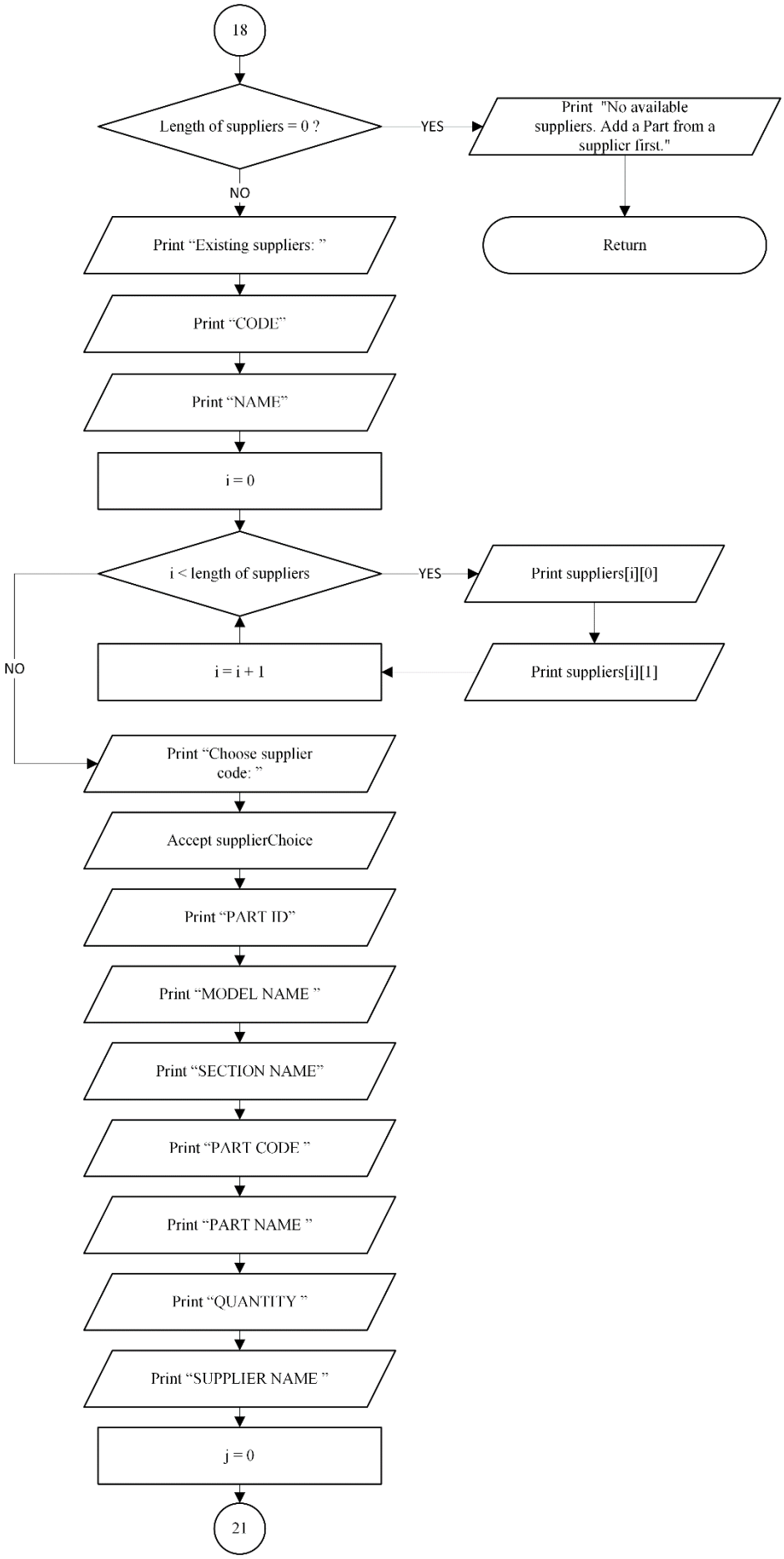

inventoryTracking(models, suppliers, parts)
Declare i, j, k, Qty, sectionChoice
Length of models = 0 ?
YES
Print "No available models. Add a model first."
Return
NO
Print "Options"
Print "1. Print all parts from all model"
Print "2. Print all parts from a model"
Print "3. Print all parts from a section of a model"
Print "4. Print all parts from a supplier"
Print "5. Print all parts with low quantity from a model"
Print "Enter your choice: "
Accept optionChoice
optionChoice = 1 ?
YES
15
NO
optionChoice = 2 ?
YES
16
NO
optionChoice = 3 ?
YES
17
NO
optionChoice = 4 ?
YES
18
NO
19

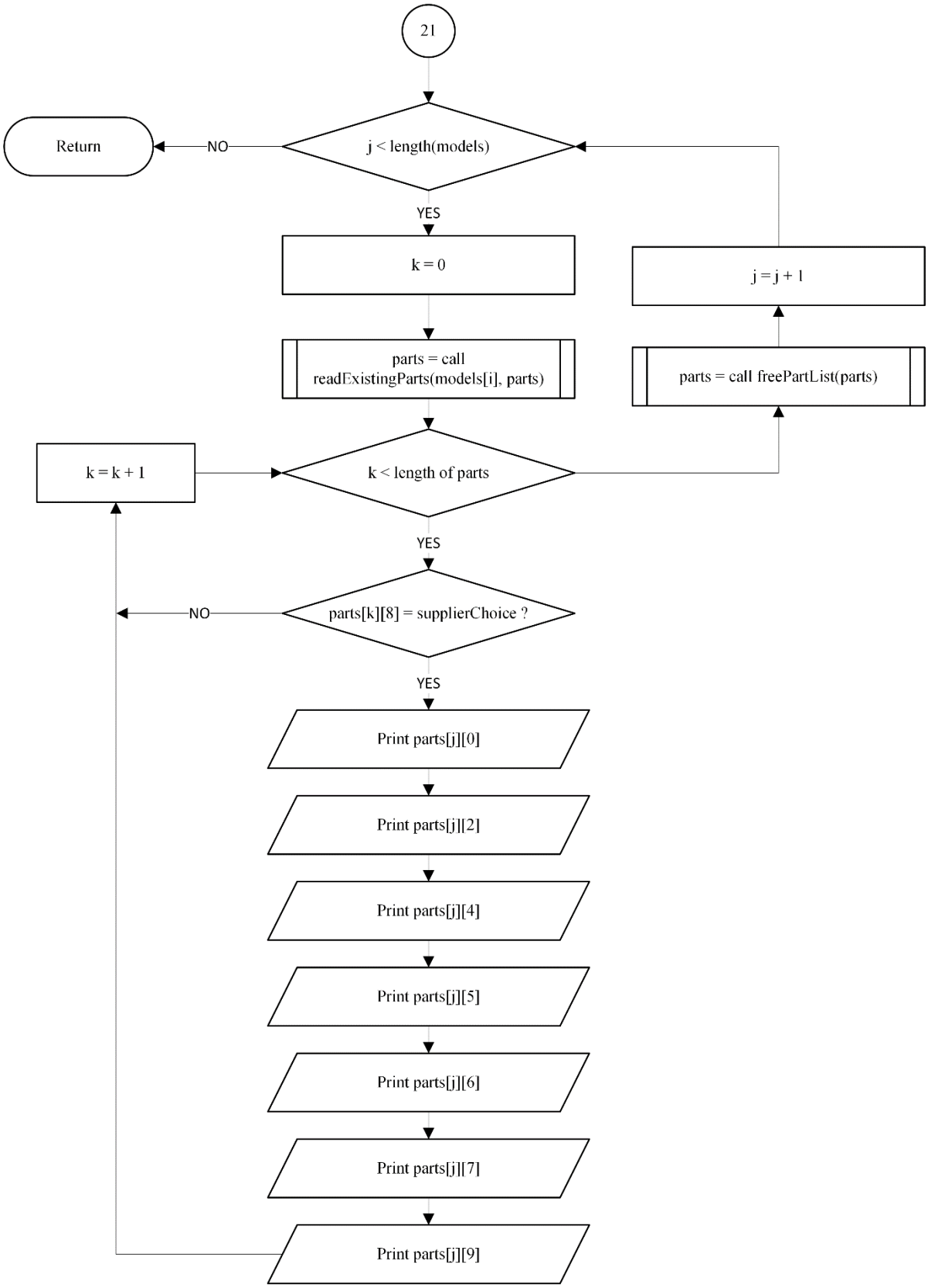

15
Print “PART ID”
Print “MODEL NAME ”
Print “SECTION NAME”
Print “PART CODE ”
Print “PART NAME ”
Print “QUANTITY ”
Print “SUPPLIER NAME ”
Print newline
i = 0
i < length of models
NO
Return
YES
j = 0
parts = call readExistingParts(models[i], parts)
j < length of parts
YES
Print parts[j][0]
Print parts[j][2]
Print parts[j][4]
Print parts[j][5]
Print parts[j][6]
Print parts[j][7]
Print parts[j][9]
j = j + 1
parts = call freePartList(parts)
i = i + 1

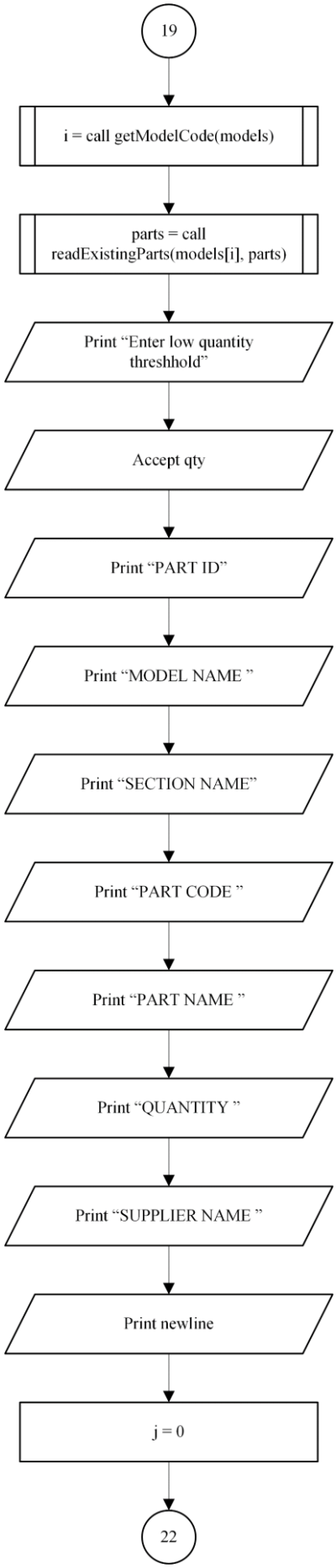

16
i = call getModelCode(models)
parts = call readExistingParts(models[i], parts)
Print "PART ID"
Print "MODEL NAME "
Print "SECTION NAME"
Print "PART CODE "
Print "PART NAME "
Print "QUANTITY "
Print "SUPPLIER NAME "
Print newline
j = 0
j < length of parts
NO
YES
Print parts[j][0]
Print parts[j][2]
Print parts[j][4]
Print parts[j][5]
Print parts[j][6]
Print parts[j][7]
Print parts[j][9]
j = j + 1
parts = call freePartList(parts)
Return

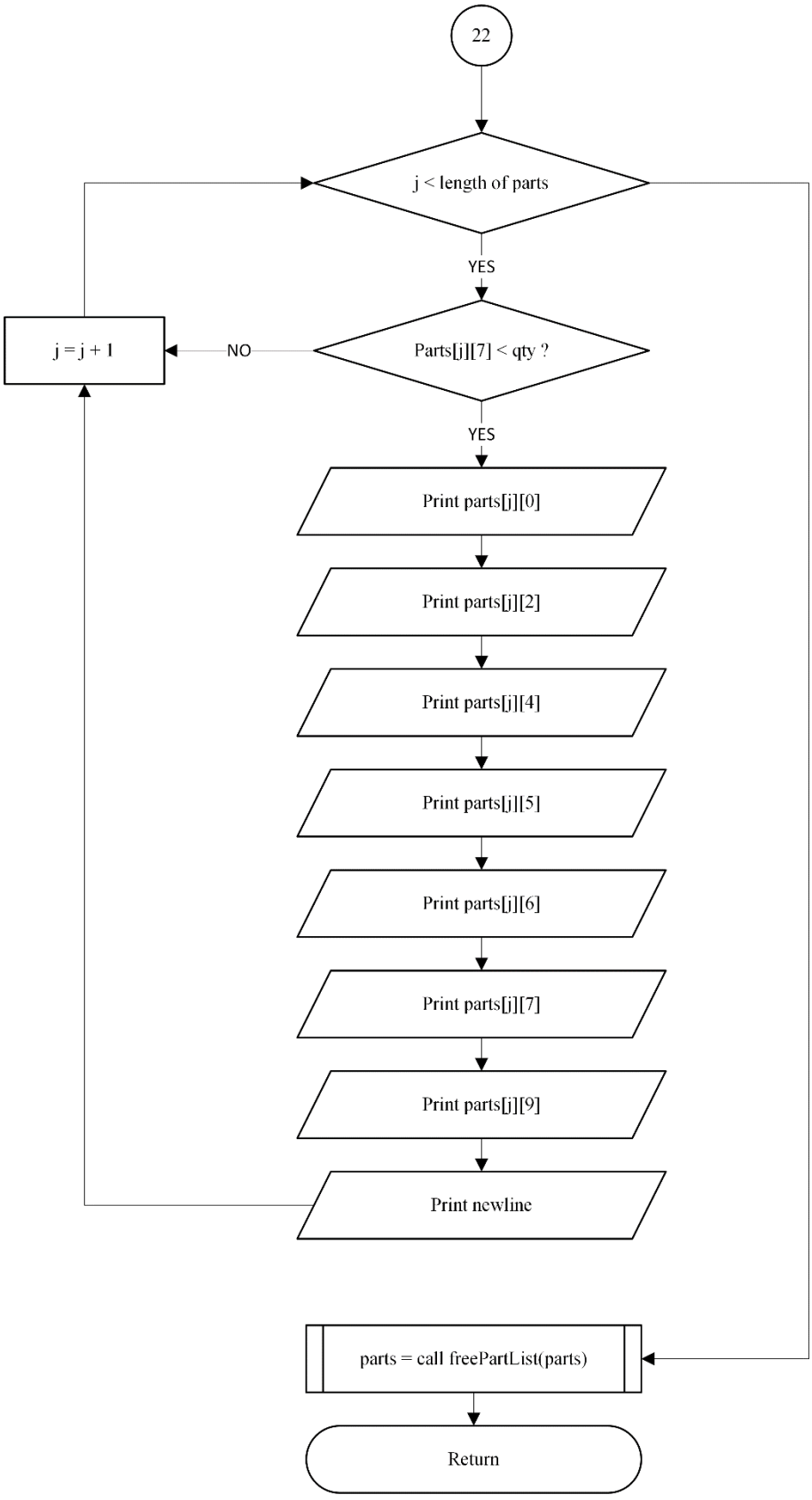

17
i = call getModelCode(models)
parts = call readExistingParts(models[i], parts)
Print "Available Section Code: "
Print "BD. Body Work"
Print "CS. Chassis"
Print "EN. Engine"
Print "TR. Transmission System"
Print "OT. Others"
Print "Enter section code: "
Accept sectionChoice
Print “PART ID”
Print “MODEL NAME ”
Print “SECTION NAME”
Print “PART CODE ”
Print “PART NAME ”
Print “QUANTITY ”
Print “SUPPLIER NAME ”
j = 0
20


20
j < length of parts
YES
j = j + 1
NO
Parts[j][3] = sectionChoice ?
YES
Print parts[j][0]
Print parts[j][2]
Print parts[j][4]
Print parts[j][5]
Print parts[j][6]
Print parts[j][7]
Print parts[j][9]
Print newline
parts = call freePartList(parts)
Return


18
Length of suppliers = 0 ?
YES
Print "No available suppliers. Add a Part from a supplier first."
Return
NO
Print “Existing suppliers: ”
Print “CODE”
Print “NAME”
i = 0
i < length of suppliers
YES
Print suppliers[i][0]
Print suppliers[i][1]
i = i + 1
NO
Print “Choose supplier code: ”
Accept supplierChoice
Print “PART ID”
Print “MODEL NAME ”
Print “SECTION NAME”
Print “PART CODE ”
Print “PART NAME ”
Print “QUANTITY ”
Print “SUPPLIER NAME ”
j = 0
21


21
j < length(models)
NO
Return
YES
k = 0
parts = call readExistingParts(models[i], parts)
k < length of parts
k = k + 1
YES
parts[k][8] = supplierChoice ?
NO
YES
Print parts[j][0]
Print parts[j][2]
Print parts[j][4]
Print parts[j][5]
Print parts[j][6]
Print parts[j][7]
Print parts[j][9]
parts = call freePartList(parts)
j = j + 1


19
i = call getModelCode(models)
parts = call readExistingParts(models[i], parts)
Print "Enter low quantity threshhold"
Accept qty
Print "PART ID"
Print "MODEL NAME "
Print "SECTION NAME"
Print "PART CODE "
Print "PART NAME "
Print "QUANTITY "
Print "SUPPLIER NAME "
Print newline
j = 0
22


22
j < length of parts
YES
Parts[j][7] < qty ?
NO
j = j + 1
YES
Print parts[j][0]
Print parts[j][2]
Print parts[j][4]
Print parts[j][5]
Print parts[j][6]
Print parts[j][7]
Print parts[j][9]
Print newline
parts = call freePartList(parts)
Return

PROGRAM SOURCE CODE

DATA TYPE

```c
struct model
{
    char code[25];
    char name[25];

    struct model* next;
}*modelStart, *modelCurr;
```

```c
struct supplier
{
    char code[25];
    char name[25];

    struct supplier* next;
}*supplierStart, * supplierCurr;
```

```c
struct part
{
    char id[25];
    char modelCode[25];
    char modelName[25];
    char sectionCode[25];
    char sectionName[25];
    char code[25];
    char name[25];
    int qty;
    char supplierCode[25];
    char supplierName[25];

    struct part* next;
}*partStart, *partCurr;
```

There are three custom data types in this program. They are *model*, *supplier* and *part*. There are two global pointers for each data type to declares three global linked list. *modelStart* and *supplierStart* linked lists are used differently compared *partStart* linked list. Because *modelStart* and *supplierStart* linked list reads from file only once, but *partStart* linked list reads from different files on every transaction and it frees up the memory once the transaction is over. It is like that because one *partStart* list gets used for all the models.

main

```c
void main()
{
    int menuChoice = NULL;

    modelStart = NULL;
    modelCurr = NULL;

    supplierStart = NULL;
    supplierCurr = NULL;
    readExistingModels();       //Read all existing models from file to linked list
    readExistingSuppliers();    //Read all existing suppliers from file to linked list

    while(1)
    {
        system("cls");
        printf("WELCOME TO AUTOMOBILE PARTS INVENTORY MANAGEMENT SYSTEM\n\n");
        printf("HOME MENU: \n");
        printf("\t1. Update inventory\n");
        printf("\t2. Inventory tracking\n");
        printf("\t3. Search inventory\n");
        printf("\t4. Add a new model\n");
        printf("\t5. Add a new part\n");
        printf("\t6. Delete a model\n");
        printf("\t7. Delete a part\n");
        printf("\t0. Exit\n");

        //MENU choice input and validation
        printf("Enter your choice of operation: ");
        if (scanf_s("%d", &menuChoice) == 0)
        {
            printf("Invalid input. Input must be an integer. Try again.\n");
            printf("Press any key to return to HOME MENU...");
            _getch();
            while (getchar() != '\n');
            continue;
        }
        while (getchar() != '\n');
```

In the *main* function, I am calling *readExistingModels* and *readExistingSuppliers* functions in the beginning. These two functions open *carModels.txt* and *suppliers.txt* to read from file into global linked list *modelStart* and *supplierStart* respectively. I am reading from these two files only once. Then in an infinite loop, I am showing all the menu options. It will keep looping forever after every transaction unless the input is zero. Then I am reading the user input in *menuChoice*. *Scanf_s* returns a non-zero value when it successfully reads from the user, otherwise it returns zero. So, if they provide a string or a character, it will be unsuccessful and return zero. In that case, it will prompt for an input again.

```
100     //Call respective functions based on input
101     if (menuChoice == 1)
102     {
103         system("cls");
104         printf("UPDATE INVENTORY\n");
105
106         updateInventory();
107     }
108     else if (menuChoice == 2)
109     {
110         system("cls");
111         printf("INVENTORY TRACKING\n");
112
113         inventoryTracking();
114     }
115     else if (menuChoice == 3)
116     {
117         system("cls");
118         printf("SEARCH INVENTORY\n");
119
120         searchInventory();
121     }
122     else if (menuChoice == 4)
123     {
124         system("cls");
125         printf("ADD A NEW MODEL\n");
126
127         addNewModel();
128     }
129     else if (menuChoice == 5)
130     {
131         system("cls");
132         printf("ADD A NEW PART\n");
133
134         addNewPart();
135     }
```

```
136     else if (menuChoice == 6)
137     {
138         system("cls");
139         printf("DELETE A MODEL\n");
140
141         deleteModel();
142     }
143     else if (menuChoice == 7)
144     {
145         system("cls");
146         printf("DELETE A PART\n");
147
148         deletePart();
149     }
150     else if (menuChoice == 0)
151     {
152         freeModelList();
153         freeSupplierList();
154         printf("Exiting program...\n");
155
156         break;
157     }
158     else
159     {
160         printf("Invalid input. Try again.\n");
161     }
162     printf("Press any key to return to HOME MENU...");
163     _getch();
```

Once I have the *menuChoice* input, I am calling the respective functions. If the input is zero, then it frees up allocated memory for *modelStart* and *supplierStart* linked lists by calling *freeModelList* and *freeSupplierList* functions. Even though the program will automatically free the memory once the execution is complete, but it is a good practice to free the memory once done it (GeeksforGeeks, 2021).

readExistingModels

```
729  void readExistingModels()
730  {
731      if (modelStart == NULL)
732      {
733          //Open File for reading
734          FILE* file;
735          if (fopen_s(&file, "carModels.txt", "r") != 0)
736              return;
737
738          fseek(file, 42, SEEK_SET); //Skip header row. 20 chars for code, 20 chars for name, 2 for newline char
739
740          //Read the file until end of file. Each loop reads one row to one node of linked list
741          //feof returns TRUE when the program tries to read a char after end of file
742          //fgetc was used for that purpose
743          while (fgetc(file) && !feof(file))
744          {
745              fseek(file, -1, SEEK_CUR); //Undo the action of fgetc inside while condition
746
747              //Allocate memory for each model
748              if (modelStart == NULL) //First node of linked list
749                  modelStart = modelCurr = (struct model*)malloc(sizeof(struct model));
750              else                    //Other nodes of linked list
751              {
752                  modelCurr->next = (struct model*)malloc(sizeof(struct model));
753                  modelCurr = modelCurr->next;
754              }
755
756              //Read from file to linked list
757              fgets(modelCurr->code, 21, file);
758              fgets(modelCurr->name, 24, file);
759
760              //Trim the empty spaces
761              rtrim(modelCurr->code);
762              rtrim(modelCurr->name);
763
764              modelCurr->next = NULL; //Setting the 'next' to NULL is very important to indicate end of list
765          }
766          fclose(file);
767      }
768      return;
769  }
```

In the *readExistingModels* function, I am reading from *carModels.txt* into *modelStart* global linked list. Right after opening the file, I am skipping header row using *fseek* function. Then I am reading from file until it is not end of file using *feof* function. There is a problem with *feof*, it does not return *true* when it reaches end of file. It returns *true* when the program tries to read another character after that. To avoid reading garbage value, I am reading one character using *fgetc* before checking end of file. This way it is properly detecting end of file. Then I am unreading the character I read right after entering the loop. I am allocating memory at the end of *modelStart* linked list using *malloc* and reading fixed length of characters from file into linked list. Then trimming the spaces at the right end of the strings using a custom function named *rtrim*.

readExistingSuppliers

```
1520  void readExistingSuppliers()
1521  {
1522      if (supplierStart == NULL)
1523      {
1524          //Open supplier file to read
1525          FILE* file;
1526          if (fopen_s(&file, "suppliers.txt", "r") != 0)
1527              return;
1528
1529          fseek(file, 42, SEEK_SET);  //Skip header row. 20 chars for code, 20 chars for name, 2 chars for newline char
1530
1531          //Read the file untile end of file. Each loop reads one row to one node of linked list
1532          //feof returns TRUE when the program tries to read a char after end of file
1533          //fgetc was used for that purpose
1534          while (fgetc(file) && !feof(file))
1535          {
1536              fseek(file, -1, SEEK_CUR); //Undo the action of fgetc inside while condition
1537
1538              //Allocate memory for each supplier
1539              if (supplierStart == NULL) //First node of linked list
1540                  supplierStart = supplierCurr = (struct supplier*)malloc(sizeof(struct supplier));
1541              else                       //Other nodes of linked list
1542              {
1543                  supplierCurr->next = (struct supplier*)malloc(sizeof(struct supplier));
1544                  supplierCurr = supplierCurr->next;
1545              }
1546
1547              //Read from file to linked list
1548              fgets(supplierCurr->code, 21, file);
1549              fgets(supplierCurr->name, 50, file);
1550
1551              //Trim the empty spaces
1552              rtrim(supplierCurr->code);
1553              rtrim(supplierCurr->name);
1554
1555              supplierCurr->next = NULL; //Setting the 'next' to NULL is very important to indicate end of list
1556          }
1557          fclose(file);
1558      }
1559      return;
1560  }
```

This function works the same way as *readExistingModels*. The only difference is it is reading from *suppliers.txt* into *supplierStart* linked list.

readExistingParts

```
986  void readExistingParts()
987  {
988      if (modelStart == NULL)
989          return;
990
991      //Create filename
992      FILE* file;
993      char filename[50];
994      strcpy_s(filename, modelCurr->code);
995      strcat_s(filename, ".txt");
996
997      //Open file
998      if (fopen_s(&file, filename, "r") != 0)
999      {
1000         printf("\nERROR. FILE NOT FOUND!.\n");
1001         return;
1002     }
1003
1004
1005     fseek(file, 202, SEEK_SET); //Skip header row. 200 chars for 10 columns, 2 chars for newline char
1006
1007     //Read the file until end of file. Each loop reads one row to one node of linked list
1008     //feof returns TRUE when the program tries to read a char after end of file
1009     //fgetc was used for that purpose
1010     while (fgetc(file) && !feof(file))
1011     {
1012         fseek(file, -1, SEEK_CUR); //Undo the action of fgetc inside while condition
1013
1014         //Allocate memory for each part
1015         if (partStart == NULL) //First node of linked list
1016             partStart = partCurr = (struct part*)malloc(sizeof(struct part));
1017         else                   //Other nodes of linked list
1018         {
1019             partCurr->next = (struct part*)malloc(sizeof(struct part));
1020             partCurr = partCurr->next;
1021         }
1022
1023         //Read from file to linked list
1024         fgets(partCurr->id, 21, file);
1025         fgets(partCurr->modelCode, 21, file);
1026         fgets(partCurr->modelName, 21, file);
1027         fgets(partCurr->sectionCode, 21, file);
1028         fgets(partCurr->sectionName, 21, file);
1029         fgets(partCurr->code, 21, file);
1030         fgets(partCurr->name, 21, file);
1031         char qty_str[25];
1032         fgets(qty_str, 21, file);
1033         fgets(partCurr->supplierCode, 21, file);
1034         fgets(partCurr->supplierName, 25, file);
1035
1036         //Trim the empty spaces
1037         rtrim(partCurr->id);
1038         rtrim(partCurr->modelCode);
1039         rtrim(partCurr->modelName);
1040         rtrim(partCurr->sectionCode);
1041         rtrim(partCurr->sectionName);
1042         rtrim(partCurr->code);
1043         rtrim(partCurr->name);
1044         rtrim(qty_str);
1045         rtrim(partCurr->supplierCode);
1046         rtrim(partCurr->supplierName);
1047
1048         partCurr->qty = atoi(qty_str); //Convert string to int for quantity
1049
1050         partCurr->next = NULL; //Setting the 'next' to NULL is very important to indicate end of list
1051     }
1052     fclose(file);
1053     return;
1054 }
```

This function has the same logic as *readExistingModels* function, but there is a bit of change. First of all, filename is not fixed. I am creating the filename in line 994 using model code from *modelCurr* global pointer which is pointer to the model to be read. I set that before calling this function. That means it is reading from the *filename* file into *partStart* linked list. One of the component, *qty* in *part* data type, is integer but *fgets* returns string. So to overcome that, I am storing the quantity into a temporary string and converting it to integer later.

getModelCode

```
947  void getModelCode()
948  {
949      if (modelStart == NULL)
950          return;
951
952      //Loop through model linked list and show all models
953      printf("Available Models: \n");
954      modelCurr = modelStart;
955      while (modelCurr != NULL)
956      {
957          printf("\t%s. %s\n", modelCurr->code, modelCurr->name);
958          modelCurr = modelCurr->next;
959      }
960
961      //Model code choice input and validation
962      int matchFound = 0;
963      char choice[25];
964      do
965      {
966          printf("Choose Model Code: ");
967          gets_s(choice);
968          strcpy_s(choice, upperCase(choice));
969
970          modelCurr = modelStart;
971          while (modelCurr != NULL)
972          {
973              if (strcmp(modelCurr->code, choice) == 0)
974              {
975                  matchFound = 1;
976                  break;
977              }
978              modelCurr = modelCurr->next;
979          }
980          if (matchFound == 0)
981              printf("Invalid input. Try again.\n");
982
983      } while (matchFound == 0);
984  }
```

This function shows all the available models to the user and prompts for a model choice input. In the beginning of this function, I am checking if the *modelStart* is empty or not. If it is empty, it will return because we have no models available to show. Then I am using a *while* loop to show all the available models and prompting them to choose one of the models using model code. Then I am running another *while* loop which is validating to make sure provided input exists in the *modelStart* linked list and at the same time, pointing *modelCurr* to the chosen model, which is necessary as this function is not returning anything. This function gets used when I need user to choose one of the available models.

addNewModel

```
771  void addNewModel()
772  {
773      //Allocate memory for new model
774      if (modelStart == NULL) //For the first node
775          modelStart = modelCurr = (struct model*)malloc(sizeof(struct model));
776      else                    //For other nodes
777      {
778          modelCurr = modelStart;
779          while (modelCurr->next != NULL)
780              modelCurr = modelCurr->next;
781          modelCurr->next = (struct model*)malloc(sizeof(struct model));
782          modelCurr = modelCurr->next;
783      }
784      modelCurr->next = NULL; //Setting the 'next' to NULL is very important to indicate end of list
785
786      //Model Code input and validation
787      //Validations: Not empty, length >= 2, length <= 5, Unique, alpha
788      char modelCode[25];
789      int modelCodeValid = 1;
790      do
791      {
792          modelCodeValid = 1;
793          printf("Enter model code: ");
794          gets_s(modelCode);
795          strcpy_s(modelCode, upperCase(rtrim(modelCode)));
796          if (strcmp(modelCode, "") == 0)
797          {
798              printf("Please provide a model code. Try again\n");
799              modelCodeValid = 0;
800              continue;
801          }
802          if (strlen(modelCode) > 5)
803          {
804              printf("Model code cannot be more than 5 characters long. Try again\n");
805              modelCodeValid = 0;
806              continue;
807          }
808          if (strlen(modelCode) < 2)
809          {
810              printf("Model code must be at least 2 characters long. Try again\n");
811              modelCodeValid = 0;
812              continue;
813          }
814          for (int i = strlen(modelCode) - 1; i >=  0 && modelCodeValid == 1; i--)
815          {
816              if (isalpha(modelCode[i]) == 0)
817              {
818                  printf("Model code can only contain letters. Try again\n");
819                  modelCodeValid = 0;
820                  break;
821              }
822          }
823          //Looking for duplicate
824          modelCurr = modelStart;
825          while (modelCurr->next != NULL && modelCodeValid == 1)
826          {
827              if (strcmp(modelCurr->code, modelCode) == 0)
828              {
829                  printf("Invalid Input. Model code already exists. It must be unique. Try again\n");
830                  modelCodeValid = 0;
831                  break;
832              }
833              modelCurr = modelCurr->next;
834          }
835      } while (modelCodeValid == 0);
836      strcpy_s(modelCurr->code, modelCode);
```

In *addNewModel* function, I am allocating memory for the new model using *malloc.* Then I am using *modelCodeValid* as a switch to validate the input. There are 5 different validations in place for model code input which is explained later in the report.

```
838     //Model name input and validation
839     //Validation: Not empty, must contain at least one letter
840     int modelNameValid = 1;
841     do
842     {
843         modelNameValid = 1;
844         printf("Enter model name: ");
845         gets_s(modelCurr->name);
846         strcpy_s(modelCurr->name, rtrim(modelCurr->name));
847
848         if (strcmp(modelCurr->name, "") == 0)
849         {
850             printf("Please provide a model name. Try again\n");
851             modelNameValid = 0;
852             continue;
853         }
854         for (int i = strlen(modelCurr->name) - 1; i >= 0; i--)
855         {
856             if (isalpha(modelCurr->name[i]) != 0)
857             {
858                 modelNameValid = 1;
859                 break;
860             }
861             modelNameValid = 0;
862         }
863         if (modelNameValid == 0)
864             printf("Invalid input. Try again\n");
865
866     } while (modelNameValid == 0);
867
868     //Create filename
869     char filename[50];
870     strcpy_s(filename, modelCurr->code);
871     strcat_s(filename, ".txt");
```

Then I am taking model name input and validating in two stages. After that, I am creating the *filename* using model code of the new model as it will be needed later in this function to create a file to store all parts data of this newly added model.

```
873     //Sort the last node
874     struct model* modelTemp = modelCurr, * modelPrev = NULL;
875
876     modelCurr = modelStart;
877     while (modelCurr != NULL)
878     {
879         if (strcmp(modelTemp->code, modelCurr->code) < 0) //If proper place of the last node is found
880         {
881             //Swap nodes
882             modelTemp->next = modelCurr;
883             if (modelCurr != modelStart)
884                 modelPrev->next = modelTemp;
885             else
886                 modelStart = modelTemp;
887
888             //Make the new last node 'next' to NULL
889             modelCurr = modelStart;
890             while (modelCurr != NULL)
891             {
892                 if (modelCurr->next == modelTemp && modelCurr != modelPrev)
893                 {
894                     modelCurr->next = NULL;
895                     break;
896                 }
897                 modelCurr = modelCurr->next;
898             }
899             break;
900         }
901         modelPrev = modelCurr;
902         modelCurr = modelCurr->next;
903     }
```

Once I am done with the input, it is time to sort the new model. Since I sort every node right after taking input, that means my data is always sorted. That means I only need to sort the newly added node. I am using two temporary pointers named *modelTemp* and *modelPrev* to assist. *modelTemp* is pointing to the node that need to be sorted while *modelPrev* will keep track of the previous node of *modelTemp. modelCurr* is looping through the linked list and finding its proper position using *strcmp* function. Once the proper position of the *modelTemp* is found, I am putting *modelTemp* at its proper place. Then I am going to the end of new last node and setting pointer *next* of that node to *NULL*. With that, *modelStart* linked list is sorted in alphabetical order by model code.

```
905     //Open carmodels.txt file
906     FILE* file;
907     if (fopen_s(&file, "carModels.txt", "w") != 0)
908         return;
909
910     //Write header into file
911     fprintf(file, "%-20s", "CODE");
912     fprintf(file, "%-20s", "NAME");
913
914     //Write rows of model data into file
915     modelCurr = modelStart;
916     while (modelCurr != NULL)
917     {
918         fprintf(file, "%s", "\n");
919
920         fprintf(file, "%-20s", modelCurr->code);
921         fprintf(file, "%-20s", modelCurr->name);
922         modelCurr = modelCurr->next; //Next model
923     }
924     fclose(file);
925
926     //Create model file
927     if (fopen_s(&file, filename, "w") == 0)
928     {
929         //Print header into file
930         fprintf(file, "%-20s", "PART ID");
931         fprintf(file, "%-20s", "MODEL CODE");
932         fprintf(file, "%-20s", "MODEL NAME");
933         fprintf(file, "%-20s", "SECTION CODE");
934         fprintf(file, "%-20s", "SECTION NAME");
935         fprintf(file, "%-20s", "PART CODE");
936         fprintf(file, "%-20s", "PART NAME");
937         fprintf(file, "%-20s", "QUANTITY");
938         fprintf(file, "%-20s", "SUPPLIER CODE");
939         fprintf(file, "%-20s", "SUPPLIER NAME");
940         fclose(file);
941
942         printf("%s created successfully.\n", filename);
943     }
944     return;
```

After sorting the linked list, I am opening the master file for models *carModels.txt* in *write* mode. I am opening it in *write* mode instead of *append* mode because if we append the new model, it will not be sorted. In that case I must sort it on every transaction. So, I am overwriting the file and writing all the model information into the file along with header row. After that, I am creating a file for this model using *filename* I created earlier by opening it in *write* mode and writing the header row.

addNewSupplier

```
1417  void addNewSupplier()
1418  {
1419      //Allocate memory for new supplier
1420      if (supplierStart == NULL) //First node
1421          supplierCurr = supplierStart = (struct supplier*)malloc(sizeof(struct supplier));
1422      else                       //Other nodes
1423      {
1424          supplierCurr = supplierStart;
1425          while (supplierCurr->next != NULL)
1426              supplierCurr = supplierCurr->next;
1427          supplierCurr->next = (struct supplier*)malloc(sizeof(struct supplier));
1428          supplierCurr = supplierCurr->next;
1429      }
1430      supplierCurr->next = NULL; //Setting the 'next' to NULL is very important to indicate e
1431
1432      //Supplier code input and validation
1433      //Validations: Not empty, length >= 2, length <= 5, Unique, only alpha
1434      int supplierCodeValid = 1;
1435      char supplierCode[25];
1436      do
1437      {
1438          supplierCodeValid = 1;
1439
1440          printf("Enter supplier code: ");
1441          gets_s(supplierCode);
1442          strcpy_s(supplierCode, upperCase(supplierCode));
1443          if (strcmp(supplierCode, "") == 0)
1444          {
1445              printf("Please provide a supplier code. Try again\n");
1446              supplierCodeValid = 0;
1447              continue;
1448          }
1449          if (strlen(supplierCode) > 5)
1450          {
1451              printf("Supplier code cannot be more than 5 characters long. Try again\n");
1452              supplierCodeValid = 0;
1453              continue;
1454          }
1455          if (strlen(supplierCode) < 2)
1456          {
1457              printf("Supplier code must be at least 2 characters long. Try again\n");
1458              supplierCodeValid = 0;
1459              continue;
1461          for (int i = strlen(supplierCode) - 1; i >= 0 && supplierCodeValid == 1; i--)
1462          {
1463              if (isalpha(supplierCode[i]) == 0)
1464              {
1465                  printf("Supplier code can only contain letters. Try again\n");
1466                  supplierCodeValid = 0;
1467                  break;
1468              }
1469          }
1470          //Looking for duplicate
1471          supplierCurr = supplierStart;
1472          while (supplierCurr->next != NULL && supplierCodeValid == 1)
1473          {
1474              if (strcmp(supplierCurr->code, supplierCode) == 0)
1475              {
1476                  printf("Invalid Input. Supplier code already exists. It must be unique. Try again\n");
1477                  supplierCodeValid = 0;
1478                  break;
1479              }
1480
1481              supplierCurr = supplierCurr->next;
1482          }
1483      } while (supplierCodeValid == 0);
1484      strcpy_s(supplierCurr->code, supplierCode);
```

```
    //Supplier name input and validation
    //Validations: not empty, must contain at least one letter
    int supplierNameValid = 1;
    do
    {
        supplierNameValid = 1;
        printf("Enter Supplier Name: ");
        gets_s(supplierCurr->name);
        strcpy_s(supplierCurr->name, rtrim(supplierCurr->name));

        if (strcmp(supplierCurr->name, "") == 0)
        {
            printf("Please provide a supplier name. Try again\n");
            supplierNameValid = 0;
            continue;
        }
        for (int i = strlen(supplierCurr->name) - 1; i >= 0; i--)
        {
            if (isalpha(supplierCurr->name[i]) != 0)
            {
                supplierNameValid = 1;
                break;
            }
            supplierNameValid = 0;
        }
        if (supplierNameValid == 0)
            printf("Invalid input. Try again\n");
    } while (supplierNameValid == 0);

    //Sort last node of the supplier linked list
    struct supplier* supplierTemp = supplierCurr, * supplierPrev = NULL;
    supplierCurr = supplierStart;
    while (supplierCurr != NULL)
    {
        if (strcmp(supplierTemp->code, supplierCurr->code) < 0) //If proper place of the last node is found
        {
            //Swap nodes
            supplierTemp->next = supplierCurr;
            if (supplierCurr != supplierStart)
                supplierPrev->next = supplierTemp;
            else
                supplierStart = supplierTemp;

            //Make the new last node 'next' to NULL
            supplierCurr = supplierStart;
            while (supplierCurr != NULL)
            {
                if (supplierCurr->next == supplierTemp && supplierCurr != supplierPrev)
                {
                    supplierCurr->next = NULL;
                    break;
                }
                supplierCurr = supplierCurr->next;
            }
            break;
        }
        supplierPrev = supplierCurr;
        supplierCurr = supplierCurr->next;
    }

    //Open supplier file to write updated, sorted supplier linked list
    FILE* file;
    if (fopen_s(&file, "suppliers.txt", "w") != 0)
    {
        printf("ERROR!");
        return;
    }

    //Write header into file
    fprintf(file, "%-20s", "SUPPLIER CODE");
    fprintf(file, "%-20s", "SUPPLIER NAME");

    //Write rows of supplier data into file
    supplierCurr = supplierStart;
    while (supplierCurr != NULL)
    {
        fprintf(file, "\n%-20s", supplierCurr->code);
        fprintf(file, "%-20s", supplierCurr->name);
        supplierCurr = supplierCurr->next;
    }
    fclose(file);

    supplierCurr = supplierTemp;
}
```

addNewSupplier works the same way as *addNewModel*. The only difference is this function reads from *suppliers.txt* instead of *carModels.txt* and writes into *modelStart* linked list instead of *supplierStart* linked list.

addNewPart

```
1056  void addNewPart()
1057  {
1058      getModelCode(); //Ask user for model choice input and modelCurr points to input model
1059      readExistingParts(); //Read all data from a file to linked list partStart
1060
1061      if (modelCurr == NULL)
1062          return;
1063
1064      //Allocate memory for the new part
1065      if (partStart == NULL) //For the first node
1066          partCurr = partStart = (struct part*)malloc(sizeof(struct part));
1067      else                    //Other nodes
1068      {
1069          while (partCurr->next != NULL)
1070              partCurr = partCurr->next;
1071          partCurr->next = (struct part*)malloc(sizeof(struct part));
1072          partCurr = partCurr->next;
1073      }
1074      partCurr->next = NULL; //Setting the 'next' to NULL is very important to indicate end of list
1075
1076      //Copy model info of the new part
1077      strcpy_s(partCurr->modelCode, modelCurr->code);
1078      strcpy_s(partCurr->modelName, modelCurr->name);
```

This function has a lot in common with *addNewModel* function. At first, I am calling *getModelCode* function to read model code from the user. It will set the *modelCurr* pointer to the chosen model. Then I am calling *readExistingParts* function which reads all the parts information into the *partStart* linked list. After that I am allocating memory using *malloc* at the end of the *partStart* list. Then copying model information into the allocated memory.

```
1080        //Show available sections
1081        printf("Available Section Code: \n");
1082        printf("\tBD. Body Work\n");
1083        printf("\tCS. Chassis\n");
1084        printf("\tEN. Engine\n");
1085        printf("\tTR. Transmission System\n");
1086        printf("\tOT. Others\n");
1087
1088        //Section Code input and validation
1089        while(1)
1090        {
1091            printf("Choose Section Code: ");
1092
1093            gets_s(partCurr->sectionCode);
1094            strcpy_s(partCurr->sectionCode, upperCase(rtrim(partCurr->sectionCode)));
1095            if (strcmp(partCurr->sectionCode, "BD") == 0)
1096            {
1097                strcpy_s(partCurr->sectionName, "Body Work");
1098                break;
1099            }
1100            if (strcmp(partCurr->sectionCode, "CS") == 0)
1101            {
1102                strcpy_s(partCurr->sectionName, "Chassis");
1103                break;
1104            }
1105            if (strcmp(partCurr->sectionCode, "EN") == 0)
1106            {
1107                strcpy_s(partCurr->sectionName, "Engine");
1108                break;
1109            }
1110            if (strcmp(partCurr->sectionCode, "TR") == 0)
1111            {
1112                strcpy_s(partCurr->sectionName, "Transmission System");
1113                break;
1114            }
1115            if (strcmp(partCurr->sectionCode, "OT") == 0)
1116            {
1117                strcpy_s(partCurr->sectionName, "Others");
1118                break;
1119            }
1120
1121            printf("Invalid input. Try again.\n");
1122        }
```

Next is the section input. I am showing available sections and taking input from the user and setting section name based on input.

```c
//Part code input and validation
//Validations: Not empty, length >= 2, length <= 5, Unique, only alpha
char partCode[25];
int partCodeValid = 1;
do
{
    partCodeValid = 1;
    partCurr = partStart;
    printf("Enter part code: ");
    gets_s(partCode);
    strcpy_s(partCode, upperCase(partCode));

    if (strcmp(partCode, "") == 0)
    {
        printf("Please provide a part code. Try again\n");
        partCodeValid = 0;
        continue;
    }
    if (strlen(partCode) > 5)
    {
        printf("Part code cannot be more than 5 characters long. Try again\n");
        partCodeValid = 0;
        continue;
    }
    else if (strlen(partCode) < 2)
    {
        printf("Part code must be at least 2 characters long. Try again\n");
        partCodeValid = 0;
        continue;
    }
    for (int i = strlen(partCode) - 1; i >= 0 && partCodeValid == 1; i--)
    {
        if (isalpha(partCode[i]) == 0)
        {
            printf("Part code can only contain letters. Try again\n");
            partCodeValid = 0;
            break;
        }
    }
    //Looking for duplicate
    while (partCurr->next != NULL && partCodeValid == 1)
    {
        if (strcmp(partCurr->code, partCode) == 0)
        {
            printf("Invalid Input. Part code already exists. It must be unique. Try again\n");
            partCodeValid = 0;
            break;
        }

        partCurr = partCurr->next;
    }
} while (partCodeValid == 0);
strcpy_s(partCurr->code, partCode);

//Generate ID using model code, section code, part code
//Format: modelCode-sectionCode-partCode
strcpy_s(partCurr->id, modelCurr->code);
strcat_s(partCurr->id, "-");
strcat_s(partCurr->id, partCurr->sectionCode);
strcat_s(partCurr->id, "-");
strcat_s(partCurr->id, partCurr->code);
```

Then I am taking part code input and validating it. Validation is discussed later in the report in the additional features section. Once I have the part code, I am generating part ID by joining model code, section code and part code using hyphen “-”. Since every code is unique, generated ID will also be unique. For example, a part in model Blaze (BZ), section Body Work (BD) with part code EH will have ID BZ-BD-EH.

```c
//part name input and validation
//Validations: not empty, must contain at least one letter
int partNameValid = 1;
do
{
    partNameValid = 1;
    printf("Enter Part Name: ");
    gets_s(partCurr->name);
    strcpy_s(partCurr->name, rtrim(partCurr->name));

    if (strcmp(partCurr->name, "") == 0)
    {
        printf("Please provide a part name. Try again\n");
        partNameValid = 0;
        continue;
    }

    for (int i = strlen(partCurr->name) - 1; i >= 0; i--)
    {
        if (isalpha(partCurr->name[i]) != 0)
        {
            partNameValid = 1;
            break;
        }
        partNameValid = 0;
    }
    if(partNameValid == 0)
        printf("Invalid input. Try again\n");
} while (partNameValid == 0);

//Part quantity input and validation
while (1)
{
    printf("Enter Part Quantity: ");
    scanf_s("%d", &partCurr->qty); while (getchar() != '\n');

    if (partCurr->qty < 1)
        printf("Invalid quantity. Try again.\n");
    else
        break;
}
```

Then I am taking part name input and part quantity input. Both inputs go through multiple stages of validations.

```c
//Supplier details
char supplierExist, choiceSupplier[25];
if (supplierStart == NULL)
    supplierExist = 'N';
else
{
    //Show all existing suppliers
    printf("Existing Suppliers:\n");

    supplierCurr = supplierStart;
    while (supplierCurr != NULL)
    {
        printf("\t%s. %s\n", supplierCurr->code, supplierCurr->name);
        supplierCurr = supplierCurr->next;
    }

    //Ask if supplier new or already exist
    while(1)
    {
        printf("Does your supplier already exist? [Y/N]: ");
        scanf_s("%c", &supplierExist); while (getchar() != '\n');
        if (supplierExist == 'y' || supplierExist == 'Y' || supplierExist == 'n' || supplierExist == 'N')
            break;
        printf("Invalid input. try again\n");
    }
}
```

Supplier details input is a bit different. At first, I am showing all the existing suppliers using *supplierStart* linked list. Then asking the user if they want to add or choose from the list.

```
1256        //Option 1: Add a new supplier
1257        if (supplierExist == 'N' || supplierExist == 'n')
1258        {
1259            //Add new supplier and supplierCurr points to added supplier
1260            addNewSupplier();
1261
1262            //Copy supplier details of the new part
1263            strcpy_s(partCurr->supplierCode, supplierCurr->code);
1264            strcpy_s(partCurr->supplierName, supplierCurr->name);
1265
1266        }
1267        //Option 2: Choose existing supplier
1268        else if (supplierExist == 'Y' || supplierExist == 'y')
1269        {
1270            //Supplier code choice input and validation
1271            int matchFound = 0;
1272            do
1273            {
1274                printf("Choose supplier code: ");
1275                gets_s(choiceSupplier);
1276                strcpy_s(choiceSupplier, upperCase(choiceSupplier));
1277                supplierCurr = supplierStart;
1278                while (supplierCurr != NULL)
1279                {
1280                    if (strcmp(supplierCurr->code, choiceSupplier) == 0)
1281                    {
1282                        matchFound = 1;
1283                        strcpy_s(partCurr->supplierCode, supplierCurr->code);
1284                        strcpy_s(partCurr->supplierName, supplierCurr->name);
1285                        break;
1286                    }
1287                    supplierCurr = supplierCurr->next;
1288                }
1289                if (matchFound == 0)
1290                    printf("Invalid input. Try again.\n");
1291            } while (matchFound == 0);
1292        }
```

If the user says the supplier of the part does not exist, then it will call *addNewSupplier* function. This function adds a supplier to the *supplierStart* linked list and *supplierCurr* points to the new supplier. Then I am copying the supplier details from *supplierCurr* to part details. On the other hand, if the supplier already exists, user can easily choose one of the existing one. This way there is no chance of having duplicate in *suppliers.txt* file.

```
1294        //Sort the last node of the part linked list
1295        struct part* partTemp = partCurr, * partPrev = NULL;
1296
1297        partCurr = partStart;
1298        while (partCurr != NULL)
1299        {
1300            if (strcmp(partTemp->id, partCurr->id) < 0) //If proper place of the last node is found
1301            {
1302                //Swap nodes
1303                partTemp->next = partCurr;
1304                if (partCurr != partStart)
1305                    partPrev->next = partTemp;
1306                else
1307                    partStart = partTemp;
1308
1309                //Make the new last node 'next' to NULL
1310                partCurr = partStart;
1311                while (partCurr != NULL)
1312                {
1313                    if (partCurr->next == partTemp && partCurr != partPrev)
1314                    {
1315                        partCurr->next = NULL;
1316                        break;
1317                    }
1318                    partCurr = partCurr->next;
1319                }
1320                break;
1321            }
1322            partPrev = partCurr;
1323            partCurr = partCurr->next;
1324        }
```

Then I am sorting the *partStart* linked list just like I sorted *modelStart* linked list in *addNewModel* function.

```
1331    //Open model file to write the updated, sorted part list
1332    FILE* file;
1333    if (fopen_s(&file, filename, "w") != 0)
1334    {
1335        printf("ERROR!");
1336        return;
1337    }
1338
1339    //Write header into file
1340    fprintf(file, "%-20s", "PART ID");
1341    fprintf(file, "%-20s", "MODEL CODE");
1342    fprintf(file, "%-20s", "MODEL NAME");
1343    fprintf(file, "%-20s", "SECTION CODE");
1344    fprintf(file, "%-20s", "SECTION NAME");
1345    fprintf(file, "%-20s", "PART CODE");
1346    fprintf(file, "%-20s", "PART NAME");
1347    fprintf(file, "%-20s", "QUANTITY");
1348    fprintf(file, "%-20s", "SUPPLIER CODE");
1349    fprintf(file, "%-20s", "SUPPLIER NAME");
1350
1351    //Write rows of parts data
1352    partCurr = partStart;
1353    while (partCurr != NULL)
1354    {
1355        fprintf(file, "\n");
1356        fprintf(file, "%-20s", partCurr->id);
1357        fprintf(file, "%-20s", partCurr->modelCode);
1358        fprintf(file, "%-20s", partCurr->modelName);
1359        fprintf(file, "%-20s", partCurr->sectionCode);
1360        fprintf(file, "%-20s", partCurr->sectionName);
1361        fprintf(file, "%-20s", partCurr->code);
1362        fprintf(file, "%-20s", partCurr->name);
1363        fprintf(file, "%-20d", partCurr->qty);
1364        fprintf(file, "%-20s", partCurr->supplierCode);
1365        fprintf(file, "%-20s", partCurr->supplierName);
1366
1367        partCurr = partCurr->next;
1368    }
1369    fclose(file);
1370
1371    printf("\n%s added successfully.\n", partTemp->name);
1372    freePartList(); //Free part linked list
1373    return;
```

After that, I am opening the file using *filename* and overwriting new sorted linked list into the file over the existing data. With that, a new part is successfully added to the system.

deleteModel

Just like I added the feature to add a model, I also added a feature to delete one. This will be useful when the company decides to stop manufacturing a model.

```
1572  void deleteModel()
1573  {
1574      if (modelStart == NULL)
1575      {
1576          printf("No available models. Add a model first.\n");
1577          return;
1578      }
1579
1580      getModelCode(); //Ask user for model choice input and modelCurr points to input model
1581
1582      //Delete confirmation input
1583      printf("All the data related to %s (%s) will be deleted.\n", modelCurr->name, modelCurr->code);
1584      printf("Are you sure you want to continue? [Y/N]: ");
1585
1586      char confirmDelete = NULL;
1587      scanf_s("%c", &confirmDelete);
```

In *deleteModel* function, I am making sure there is model available to be deleted. Then I am calling *getModelCode* function to get model user wants to delete. *modelCurr* will point to the chosen node in *modelStart* linked list. Then I am asking for confirmation.

```
1589      //Delete the model
1590      if (confirmDelete == 'y' || confirmDelete == 'Y')
1591      {
1592          //Create filename using model code
1593          char filename[50];
1594          strcpy_s(filename, modelCurr->code);
1595          strcat_s(filename, ".txt");
1596
1597          //delete node from linked list
1598          if (strcmp(modelStart->code, modelCurr->code) == 0) //If selected node is first node
1599          {
1600              modelStart = modelStart->next;
1601              free(modelCurr);
1602              modelCurr = modelStart;
1603          }
1604          else
1605          {
1606              struct model* modelPrev = modelStart;
1607              while (strcmp(modelPrev->next->code, modelCurr->code) != 0)
1608                  modelPrev = modelPrev->next;
1609              modelPrev->next = modelCurr->next;
1610              free(modelCurr);
1611              modelCurr = modelStart;
1612          }
```

Once I get the confirmation to delete, I am saving the *filename* of this model, which we will need later in the function. There are two scenarios when it comes to delete a node from a linked list. The node to be deleted can be the first node. In that case, I am just freeing up the memory in that node and setting the next node as the head node of this list. For the rest of the nodes, I am using a pointer *modelPrev* to delete. *modelPrev* is pointing to the previous node of the node to be deleted, *modelCurr*. Then *modelPrev* passing over to the next node of *modelCurr* to disconnect *modelCurr* from the list. Once *modelCurr* is disconnected from list, I am freeing it up.

```
1614        //Open file to write updated model linked list into file
1615        FILE* file;
1616        if (fopen_s(&file, "carModels.txt", "w") != 0)
1617            return;
1618
1619        //Write header into file
1620        fprintf(file, "%-20s", "CODE");
1621        fprintf(file, "%-20s", "NAME");
1622
1623        //Write rows of model data into file
1624        while (modelCurr != NULL)
1625        {
1626            fprintf(file, "\n");
1627            fprintf(file, "%-20s", modelCurr->code);
1628            fprintf(file, "%-20s", modelCurr->name);
1629            modelCurr = modelCurr->next;
1630        }
1631        fclose(file);
1632
1633        //Delete model file
1634        if (remove(filename) == 0)
1635            printf("%s has been deleted successfully.\n", filename);
1636    }
1637    else
1638        printf("Delete process cancelled.\n");
```

Now we have deleted the chosen model from our linked list. It is time to update in the file. So, I am just overwriting the linked list over existing data. At the end, I am deleting the model file by calling *remove* function using *filename* I created earlier in the function. In case the user decides not to delete then it will abort the process.

deletePart

```
1642  void deletePart()
1643  {
1644      if (modelStart == NULL)
1645      {
1646          printf("No available models. Add a model first.\n");
1647          return;
1648      }
1649      getModelCode(); //Ask user for model choice input and modelCurr points to input model
1650      readExistingParts(); //Read all data from a file to linked list partStart
1651
1652      if (modelCurr == NULL)
1653          return;
1654
1655      if (partStart == NULL)
1656      {
1657          printf("No available parts for this model. Add a part first.\n");
1658          return;
1659      }
1660
1661      printf("Available Parts:\n");
1662
1663      //Print header rows to console
1664      printf("\t%-20s", "PART ID");
1665      printf("\t%-20s", "MODEL NAME");
1666      printf("\t%-20s", "SECTION NAME");
1667      printf("\t%-20s", "PART CODE");
1668      printf("\t%-20s", "PART NAME");
1669      printf("\t%-20s", "QUANTITY");
1670      printf("\t%-20s\n", "SUPPLIER NAME");
1671
1672      //Print rows of parts data
1673      partCurr = partStart;
1674      while (partCurr != NULL)
1675      {
1676          printf("\t%-20s", partCurr->id);
1677          printf("\t%-20s", partCurr->modelName);
1678          printf("\t%-20s", partCurr->sectionName);
1679          printf("\t%-20s", partCurr->code);
1680          printf("\t%-20s", partCurr->name);
1681          printf("\t%-20d", partCurr->qty);
1682          printf("\t%-20s\n", partCurr->supplierName);
1683          partCurr = partCurr->next;
1684      }
```

Deleting a part is a lot like deleting a model. In *deletePart* function I am calling *getModelCode* to prompt user to choose one of the available models. Then I am reading all the parts details from the chosen model into *partStart* linked list. Using the linked list, I am printing all the available parts so that user can choose a part to delete. In case there is no model available or no part available under this mode, I am showing an appropriate message and returning to home menu.

```
1686    //Part code choice input and validation
1687    int matchFound = 0;
1688    char choicePart[25];
1689    do
1690    {
1691        printf("Choose Part Code: ");
1692
1693        gets_s(choicePart);
1694        strcpy_s(choicePart, upperCase(choicePart));
1695
1696        partCurr = partStart;
1697        while (partCurr != NULL)
1698        {
1699            if (strcmp(partCurr->code, choicePart) == 0)
1700            {
1701                matchFound = 1;
1702                break;
1703            }
1704            partCurr = partCurr->next;
1705        }
1706        if (matchFound == 0)
1707            printf("Invalid input. Try again.\n");
1708
1709    } while (matchFound == 0);
1710
1711    //Delete confirmation input
1712    printf("All the data related to %s (%s) of %s (%s) will be deleted.\n", partCurr->name, partCurr->code, modelCurr->name, modelCurr->code);
1713    printf("Are you sure you want to continue? [Y/N]: ");
1714    char confirmDelete = NULL;
1715    scanf_s("%c", &confirmDelete);
```

User then chooses one of the available parts and provides confirmation to delete the part.

```
1717        //Delete part
1718        if (confirmDelete == 'Y' || confirmDelete == 'y')
1719        {
1720            //Delete the node
1721            if (strcmp(partStart->code, partCurr->code) == 0) // selected part is the first member of linked list
1722            {
1723                partStart = partStart->next;
1724                free(partCurr);
1725                partCurr = partStart;
1726            }
1727            else
1728            {
1729                struct part* partPrev = partStart;
1730                while (strcmp(partPrev->next->code, partCurr->code) != 0)
1731                    partPrev = partPrev->next;
1732                partPrev->next = partCurr->next;
1733                free(partCurr);
1734                partCurr = partStart;
1735            }
```

Then I am deleting the node with the chosen part from the *partStart* linked list just like I deleted a node from *modelStart* linked list in *deleteModel* function.

```
1742        //Open file to write updated part linked list into file
1743        FILE* file;
1744        if (fopen_s(&file, filename, "w") != 0)
1745        {
1746            printf("ERROR. File not found.\n");
1747            return;
1748        }
1749
1750        //Write header into file
1751        fprintf(file, "%-20s", "PART ID");
1752        fprintf(file, "%-20s", "MODEL CODE");
1753        fprintf(file, "%-20s", "MODEL NAME");
1754        fprintf(file, "%-20s", "SECTION CODE");
1755        fprintf(file, "%-20s", "SECTION NAME");
1756        fprintf(file, "%-20s", "PART CODE");
1757        fprintf(file, "%-20s", "PART NAME");
1758        fprintf(file, "%-20s", "QUANTITY");
1759        fprintf(file, "%-20s", "SUPPLIER CODE");
1760        fprintf(file, "%-20s", "SUPPLIER NAME");
1761
1762        //Write rows of parts data into file
1763        partCurr = partStart;
1764        while (partCurr != NULL)
1765        {
1766            fprintf(file, "\n");
1767            fprintf(file, "%-20s", partCurr->id);
1768            fprintf(file, "%-20s", partCurr->modelCode);
1769            fprintf(file, "%-20s", partCurr->modelName);
1770            fprintf(file, "%-20s", partCurr->sectionCode);
1771            fprintf(file, "%-20s", partCurr->sectionName);
1772            fprintf(file, "%-20s", partCurr->code);
1773            fprintf(file, "%-20s", partCurr->name);
1774            fprintf(file, "%-20d", partCurr->qty);
1775            fprintf(file, "%-20s", partCurr->supplierCode);
1776            fprintf(file, "%-20s", partCurr->supplierName);
1777
1778            partCurr = partCurr->next;
1779        }
1780        fclose(file);
1781        printf("Delete Process Successful\n");
1782    }
1783    else
1784        printf("Delete Process Cancelled.\n");
1785
1786    freePartList(); //Free parts linked list
1787 }
```

Then I am opening the file of the model of this part in *write* mode. Then overwriting all existing data with updated *partStart* linked list. At the end, *freePartList* function gets called which frees up all the memory allocated by the *partStart* linked list.

searchInventory

```
339  void searchInventory()
340  {
341      if (modelStart == NULL)
342      {
343          printf("No available models. Add a model first.\n");
344          return;
345      }
346
347      int matchFound = 0;
348
349      //Keyword input
350      char keyword[25];
351      printf("Enter Search Keyword: ");
352      gets_s(keyword);
353      strcpy_s(keyword, upperCase(keyword));
354
355      //Print header in console
356      printf("\t%-20s", "PART ID");
357      printf("\t%-20s", "MODEL NAME");
358      printf("\t%-20s", "SECTION NAME");
359      printf("\t%-20s", "PART CODE");
360      printf("\t%-20s", "PART NAME");
361      printf("\t%-20s", "QUANTITY");
362      printf("\t%-20s\n", "SUPPLIER NAME");
```

In the search function, I am prompting the user for a search *keyword* input. When the provide it, I am printing the header row.

```
364      //Loop through the models
365      modelCurr = modelStart;
366      while (modelCurr != NULL)
367      {
368          readExistingParts(); //Read all data from the file to linked list partStart
369
370          //Loop through the partStart linked list to find match
371          int found = 0;
372          partCurr = partStart;
373          while (partCurr != NULL)
374          {
375              char modelName[25];
376              char sectionName[25];
377              char partName[25];
378              char supplierName[25];
379
380              strcpy_s(modelName, partCurr->modelName);
381              strcpy_s(sectionName, partCurr->sectionName);
382              strcpy_s(partName, partCurr->name);
383              strcpy_s(supplierName, partCurr->supplierName);
384
385              //Look for match
386              if (strstr(partCurr->id, keyword))
387                  found = 1;
388              else if (strstr(partCurr->modelCode, keyword))
389                  found = 1;
390              else if (strstr(upperCase(modelName), keyword))
391                  found = 1;
392              else if (strstr(partCurr->sectionCode, keyword))
393                  found = 1;
394              else if (strstr(upperCase(sectionName), keyword))
395                  found = 1;
396              else if (strstr(partCurr->code, keyword))
397                  found = 1;
398              else if (strstr(upperCase(partName), keyword))
399                  found = 1;
400              else if (strstr(partCurr->supplierCode, keyword))
401                  found = 1;
402              else if (strstr(upperCase(supplierName), keyword))
403                  found = 1;
```

Once I have the search keyword, I am looping through each model and reading all the parts details into *partStart* linked list using *readExistingParts* function. Then I am looping through each part from that list and searching for a match using *strstr* function. If the keyword is a substring of any of the information from a part, it will be considered as a match.

```
404
405             //If found, print rows of parts data
406             if (found)
407             {
408                 printf("\t%-20s", partCurr->id);
409                 printf("\t%-20s", partCurr->modelName);
410                 printf("\t%-20s", partCurr->sectionName);
411                 printf("\t%-20s", partCurr->code);
412                 printf("\t%-20s", partCurr->name);
413                 printf("\t%-20d", partCurr->qty);
414                 printf("\t%-20s\n", partCurr->supplierName);
415                 matchFound++;
416             }
417             found = 0;
418             partCurr = partCurr->next;
419         }
420         freePartList(); //Free part Linked list
421 
422         modelCurr = modelCurr->next; //Next model
423     }
424     printf("Match Found: %d\n", matchFound);
425 }
```

Once I find a match, I am printing the information about the part. At the end of each model, I am freeing up the allocated memory for *partStart* linked list using *freePartList* function before going into the next model. This is necessary because if I do not free up this memory, the program will not be able to write from the new file into the list because this list will not be empty. To end it all, I am printing the total number of match found for this keyword.

updateInventory

```
169  void updateInventory()
170  {
171      if (modelStart == NULL)
172      {
173          printf("No available models. Add a model first.\n");
174          return;
175      }
176
177      getModelCode(); //Ask user for model choice input and modelCurr points to input model
178      readExistingParts(); //Read all data from a file to linked list partStart
179
180      if (modelCurr == NULL)
181          return;
182
183      if (partStart == NULL)
184      {
185          printf("No available parts for this model. Add a part first.\n");
186          return;
187      }
188
189      //Print the header in console
190      printf("Available Parts:\n");
191      printf("\t%-20s", "PART ID");
192      printf("\t%-20s", "MODEL NAME");
193      printf("\t%-20s", "SECTION NAME");
194      printf("\t%-20s", "PART CODE");
195      printf("\t%-20s", "PART NAME");
196      printf("\t%-20s", "QUANTITY");
197      printf("\t%-20s\n", "SUPPLIER NAME");
198
199      //Print rows of parts data
200      partCurr = partStart;
201      while (partCurr != NULL)
202      {
203          printf("\t%-20s", partCurr->id);
204          printf("\t%-20s", partCurr->modelName);
205          printf("\t%-20s", partCurr->sectionName);
206          printf("\t%-20s", partCurr->code);
207          printf("\t%-20s", partCurr->name);
208          printf("\t%-20d", partCurr->qty);
209          printf("\t%-20s\n", partCurr->supplierName);
210
211          partCurr = partCurr->next;
212      }
```

In the beginning of *updateInventory* function, I am calling *getModelCode* function to prompt user to choose one of the available models. This function sets *modelCurr* to point to the chosen model. Then *readExistingParts* function opens the file of that model and reads all the data into the *partStart* linked list. After that I am printing all the available parts to the screen. In case *modelStart* or *partStart* is empty, the function will return to main menu.

```c
//Part code input and validation
int matchFound = 0;
char partChoice[25];
do
{
    printf("Choose Part Code: ");

    gets_s(partChoice);
    strcpy_s(partChoice, upperCase(partChoice));

    partCurr = partStart;
    while (partCurr != NULL)
    {
        if (strcmp(partCurr->code, partChoice) == 0)
        {
            matchFound = 1;
            break;
        }
        partCurr = partCurr->next;
    }
    if (matchFound == 0)
        printf("Invalid input. Try again.\n");

} while (matchFound == 0);

//Add or remove quantity input and validation
int operation = 0;
printf("Operations:\n");
printf("\t1. Add Quantity [Supplier to Warehouw]\n");
printf("\t2. Subtract Quantity [Warehouse to Assembly]\n");

while(1)
{
    printf("Choose Operation: ");
    scanf_s("%d", &operation); while (getchar() != '\n');

    if (operation != 1 && operation != 2)
        printf("Invalid input. Try again.\n");
    else
        break;
}
```

Then User chooses one of the available parts from the list shown. Later, the user gets presented by two options: *Add* or *subtract* quantity. If there are parts coming from supplier to the warehouse, then the user should choose to *add* and if the assembly team is requesting parts from warehouse, then *subtract* option should be chosen.

```c
//Part quantity input, validation and update in linked list
int qty = 0;
while (operation == 1) //Add quantity
{
    printf("Enter quantity to add: ");
    scanf_s("%d", &qty); while (getchar() != '\n');

    if (qty < 1)
        printf("Invalid quantity. Try again.\n");
    else
    {
        partCurr->qty += qty;
        break;
    }
}
while (operation == 2) //Subtract quantity
{
    if (partCurr->qty == 0)
    {
        printf("Sorry. Part out of stock.\n");
        return;
    }
    printf("Enter quantity to subtract: ");
    scanf_s("%d", &qty); while (getchar() != '\n');

    if (qty < 1 || qty > partCurr->qty)
        printf("Invalid quantity. Try again.\n");
    else
    {
        partCurr->qty -= qty;
        break;
    }
}
```

When the user chooses to add or remove, program prompts user for quantity. Quantity gets validated in two stages. Validation process has been discussed later in the report. Once the program makes sure the input quantity is valid, then it updates the quantity of the selected part.

```
290     //Open File
291     char filename[50];
292     strcpy_s(filename, modelCurr->code);
293     strcat_s(filename, ".txt");
294
295     FILE* file;
296     if (fopen_s(&file, filename, "w") != 0)
297     {
298         printf("ERROR. File not found.\n");
299         return;
300     }
301
302     //Write Header to file
303     fprintf(file, "%-20s", "PART ID");
304     fprintf(file, "%-20s", "MODEL CODE");
305     fprintf(file, "%-20s", "MODEL NAME");
306     fprintf(file, "%-20s", "SECTION CODE");
307     fprintf(file, "%-20s", "SECTION NAME");
308     fprintf(file, "%-20s", "PART CODE");
309     fprintf(file, "%-20s", "PART NAME");
310     fprintf(file, "%-20s", "QUANTITY");
311     fprintf(file, "%-20s", "SUPPLIER CODE");
312     fprintf(file, "%-20s", "SUPPLIER NAME");
313
314     //Write rows of parts data to file
315     partCurr = partStart;
316     while (partCurr != NULL)
317     {
318         fprintf(file, "\n");
319         fprintf(file, "%-20s", partCurr->id);
320         fprintf(file, "%-20s", partCurr->modelCode);
321         fprintf(file, "%-20s", partCurr->modelName);
322         fprintf(file, "%-20s", partCurr->sectionCode);
323         fprintf(file, "%-20s", partCurr->sectionName);
324         fprintf(file, "%-20s", partCurr->code);
325         fprintf(file, "%-20s", partCurr->name);
326         fprintf(file, "%-20d", partCurr->qty);
327         fprintf(file, "%-20s", partCurr->supplierCode);
328         fprintf(file, "%-20s", partCurr->supplierName);
329
330         partCurr = partCurr->next;
331     }
332     fclose(file);
333
334     printf("Inventory successfully updated.\n");
335     freePartList(); //Free Part Linked List
```

Then I am opening the appropriate file in *write* mode using *filename* and overwriting existing data with the updated *partStart* linked list. Even though it is taking more resources to write information of all the parts, it is easier to process and more secure. After writing, I am freeing up the memory allocated for *partStart* list by calling *freePartList* function.

inventoryTracking

```
426  void inventoryTracking()
427  {
428      if (modelStart == NULL)
429      {
430          printf("No available models. Add a model first.\n");
431          return;
432      }
433
434      int totalMatchFound = 0;
435
436      printf("Print Options: \n");
437      printf("\t1. Print all parts from all model\n");
438      printf("\t2. Print all parts from a model\n");
439      printf("\t3. Print all parts from a section of a model\n");
440      printf("\t4. Print all parts from a supplier\n");
441      printf("\t5. Print all parts with low quantity from a model\n");
442
443      //Choice of operation input and validation
444      int optionChoice = 0;
445
446      while (1)
447      {
448          printf("Enter choice of operation: ");
449          scanf_s("%d", &optionChoice); while (getchar() != '\n');
450
451          if (optionChoice < 1 && optionChoice > 5)
452              printf("Invalid input. Try again.\n");
453          else
454              break;
455      }
456
457      if (optionChoice == 1) { ... }
500      else if (optionChoice == 2) { ... }
532      else if (optionChoice == 3) { ... }
594      else if (optionChoice == 4) { ... }
678      else if (optionChoice == 5) { ... }
726      printf("Match Found: %d\n", totalMatchFound);
727  }
```

In the *inventoryTracking* function, I am showing five options and prompting for option input in the beginning. Once the user provides a valid input, the program goes into one of the five *if* blocks and prints parts data by looping through parts of a model or all models. We are going to see all these *if* blocks one by one.

```c
if (optionChoice == 1)
{
    if (modelStart == NULL)
    {
        printf("No available models. Add a model first.\n");
        return;
    }

        //Print header row in console
        printf("\t%-20s", "PART ID");
        printf("\t%-20s", "MODEL NAME");
        printf("\t%-20s", "SECTION NAME");
        printf("\t%-20s", "PART CODE");
        printf("\t%-20s", "PART NAME");
        printf("\t%-20s", "QUANTITY");
        printf("\t%-20s\n", "SUPPLIER NAME");

        //Loop through models
        modelCurr = modelStart;
        while (modelCurr != NULL)
        {
            readExistingParts(); //Read all parts data into partStart linked list

            //Print rows of parts data IF supplier code matches
            partCurr = partStart;
            while (partCurr != NULL)
            {
                printf("\t%-20s", partCurr->id);
                printf("\t%-20s", partCurr->modelName);
                printf("\t%-20s", partCurr->sectionName);
                printf("\t%-20s", partCurr->code);
                printf("\t%-20s", partCurr->name);
                printf("\t%-20d", partCurr->qty);
                printf("\t%-20s\n", partCurr->supplierName);
                totalMatchFound++;

                partCurr = partCurr->next;
            }
            freePartList(); //Free part linked list

            modelCurr = modelCurr->next; //Next model
        }
}
```

Option 1 just loops through all the models and for each model it reads all parts data from file using *readExistingParts*. Then I am printing all the parts into the screen. Before going into the next model, I am freeing up the memory allocated by *partStart* linked list by calling *freePartList* function.

```c
else if (optionChoice == 2)
{
    getModelCode(); //Ask user for model choice input and modelCurr points to input model
    readExistingParts(); //Read all data from a file to linked list partStart

    //Print Header row in console
    printf("\t%-20s", "PART ID");
    printf("\t%-20s", "MODEL NAME");
    printf("\t%-20s", "SECTION NAME");
    printf("\t%-20s", "PART CODE");
    printf("\t%-20s", "PART NAME");
    printf("\t%-20s", "QUANTITY");
    printf("\t%-20s\n", "SUPPLIER NAME");

    //Print rows of parts data
    partCurr = partStart;
    while (partCurr != NULL)
    {
        printf("\t%-20s", partCurr->id);
        printf("\t%-20s", partCurr->modelName);
        printf("\t%-20s", partCurr->sectionName);
        printf("\t%-20s", partCurr->code);
        printf("\t%-20s", partCurr->name);
        printf("\t%-20d", partCurr->qty);
        printf("\t%-20s\n", partCurr->supplierName);
        partCurr = partCurr->next;
        totalMatchFound++;
    }

    freePartList(); //Free part linked list

}
```

As for option 2, instead of showing parts from all models, it only shows from one model. For that I am calling *getModelCode* function to prompt user to chose from one of the available models. Just as before, after printing all the parts, I am freeing up the *partStart* linked list.

```
532     else if (optionChoice == 3)
533     {
534         getModelCode(); //Ask user for model choice input and modelCurr points to input model
535         readExistingParts(); //Read all data from a file to linked list partStart
536
537         //Section choice input and validation
538         char sectionChoice[25];
539         printf("Available sections: ");
540         printf("\tBD. Body\n");
541         printf("\tCS. Chassis\n");
542         printf("\tEN. Engine\n");
543         printf("\tTR. Transmission System\n");
544         printf("\tOT. Others\n");
545
546         while (1)
547         {
548             printf("Choose Section Code: ");
549             gets_s(sectionChoice);
550
551             if (strcmp(upperCase(sectionChoice), "BD") == 0)
552                 break;
553             if (strcmp(upperCase(sectionChoice), "CS") == 0)
554                 break;
555             if (strcmp(upperCase(sectionChoice), "EN") == 0)
556                 break;
557             if (strcmp(upperCase(sectionChoice), "TR") == 0)
558                 break;
559             if (strcmp(upperCase(sectionChoice), "OT") == 0)
560                 break;
561
562             printf("Invalid input. Try again.\n");
563         }
```

For option 3, It is almost as same as option 2, but it also prompts user to choose one of the 5 sections available.

```
565         //Print header row to console
566         printf("\t%-20s", "PART ID");
567         printf("\t%-20s", "MODEL NAME");
568         printf("\t%-20s", "SECTION NAME");
569         printf("\t%-20s", "PART CODE");
570         printf("\t%-20s", "PART NAME");
571         printf("\t%-20s", "QUANTITY");
572         printf("\t%-20s\n", "SUPPLIER NAME");
573
574         //Print rows of parts data IF section code matches
575         partCurr = partStart;
576         while (partCurr != NULL)
577         {
578             if (strcmp(partCurr->sectionCode, upperCase(sectionChoice)) == 0)
579             {
580                 printf("\t%-20s", partCurr->id);
581                 printf("\t%-20s", partCurr->modelName);
582                 printf("\t%-20s", partCurr->sectionName);
583                 printf("\t%-20s", partCurr->code);
584                 printf("\t%-20s", partCurr->name);
585                 printf("\t%-20d", partCurr->qty);
586                 printf("\t%-20s\n", partCurr->supplierName);
587                 totalMatchFound++;
588             }
589             partCurr = partCurr->next;
590         }
591
592         freePartList(); //Free part linked list
593     }
```

Then I am looping through *partStart* and showing all the parts from the chosen section.

```
else if (optionChoice == 4)
{
    if (modelStart == NULL)
    {
        printf("No available models. Add a model first.\n");
        return;
    }

    if (supplierStart == NULL)
    {
        printf("No available suppliers. Add a part from a supplier first.\n");
        return;
    }

    //Show available suppliers using supplier linked list
    printf("Existing Suppliers: \n");
    supplierCurr = supplierStart;
    while (supplierCurr != NULL)
    {
        printf("\t%s. %s\n", supplierCurr->code, supplierCurr->name);
        supplierCurr = supplierCurr->next;
    }

    //Supplier choice input and validation
    int matchFound = 0;
    char choiceSupplier[25];
    do
    {
        printf("Choose supplier code: ");
        gets_s(choiceSupplier);
        strcpy_s(choiceSupplier, upperCase(choiceSupplier));
        supplierCurr = supplierStart;
        while (supplierCurr != NULL)
        {
            if (strcmp(supplierCurr->code, choiceSupplier) == 0)
            {
                matchFound = 1;
                break;
            }
            supplierCurr = supplierCurr->next;
        }
        if (matchFound == 0)
            printf("Invalid input. Try again.\n");
    } while (matchFound == 0);
```

In section 4, I am prompting user to choose one of the existing supplier because this block will print all the parts from a specific supplier.

```
    //Print header row in console
    printf("\t%-20s", "PART ID");
    printf("\t%-20s", "MODEL NAME");
    printf("\t%-20s", "SECTION NAME");
    printf("\t%-20s", "PART CODE");
    printf("\t%-20s", "PART NAME");
    printf("\t%-20s", "QUANTITY");
    printf("\t%-20s\n", "SUPPLIER NAME");

    //Loop through models
    modelCurr = modelStart;
    while (modelCurr != NULL)
    {
        readExistingParts(); //Read all parts data into partStart linked list

        //Print rows of parts data IF supplier code matches
        partCurr = partStart;
        while (partCurr != NULL)
        {
            if (strcmp(partCurr->supplierCode, choiceSupplier) == 0)
            {
                printf("\t%-20s", partCurr->id);
                printf("\t%-20s", partCurr->modelName);
                printf("\t%-20s", partCurr->sectionName);
                printf("\t%-20s", partCurr->code);
                printf("\t%-20s", partCurr->name);
                printf("\t%-20d", partCurr->qty);
                printf("\t%-20s\n", partCurr->supplierName);
                totalMatchFound++;
            }

            partCurr = partCurr->next;
        }
        freePartList(); //Free part linked list

        modelCurr = modelCurr->next; //Next model
    }
```

Then just like option 1, I am looping through all the parts of all the models and printing information of a part that is from the chosen supplier. At the end, I am freeing up the *partStart* linked list by calling *freePartList* function.

```
678     else if (optionChoice == 5)
679     {
680         getModelCode(); //Ask user for model choice input and modelCurr points to input model
681         readExistingParts(); //Read all data from a file to linked list partStart
682 
683         //Low quantity threshhold input and validation
684         int qty = 0;
685         while (1)
686         {
687             printf("Enter Low Quantity threshhold: ");
688             scanf_s("%d", &qty); while (getchar() != '\n');
689             if (qty <= 0)
690                 printf("Invalid input. Try again.\n");
691             else
692                 break;
693         }
694 
695         //Print header in console
696         printf("\t%-20s", "PART ID");
697         printf("\t%-20s", "MODEL NAME");
698         printf("\t%-20s", "SECTION NAME");
699         printf("\t%-20s", "PART CODE");
700         printf("\t%-20s", "PART NAME");
701         printf("\t%-20s", "QUANTITY");
702         printf("\t%-20s\n", "SUPPLIER NAME");
703 
704         //Print rows of parts data IF quantity < low quantity threshhold
705         partCurr = partStart;
706         while (partCurr != NULL)
707         {
708             if(partCurr->qty < qty)
709             {
710                 printf("\t%-20s", partCurr->id);
711                 printf("\t%-20s", partCurr->modelName);
712                 printf("\t%-20s", partCurr->sectionName);
713                 printf("\t%-20s", partCurr->code);
714                 printf("\t%-20s", partCurr->name);
715                 printf("\t%-20d", partCurr->qty);
716                 printf("\t%-20s\n", partCurr->supplierName);
717 
718                 totalMatchFound++;
719             }
720             partCurr = partCurr->next;
721         }
722 
723         freePartList(); //Free part linked list
```

Last option, option 5, must prompt user for model choice as per question requirement. Once the user chooses a model then the program calls *readExistingParts* function to read all the data from selected model into *partStart* linked list. After that, the program prompts the user for low quantity threshold. It means any part with available quantity less than this will be considered low stock. Then I am looping through all the parts and checking if the available quantity is less than the low quantity threshold. If yes, then I am printing the part details. At the end I am calling *freePartList* to free up the memory allocated by *partStart* linked list.

freeModelList

```
1802  void freeModelList()
1803  {
1804      if (modelStart == NULL) //IF the list is empty already
1805          modelCurr = NULL;
1806
1807      else if (modelStart->next == NULL) //If there there is only one node
1808      {
1809          free(modelStart);
1810          modelStart = modelCurr = NULL;
1811      }
1812      else //If list has two or more nodes
1813      {
1814          //Delete nodes from beginning to end
1815          while (modelStart != NULL)
1816          {
1817              modelCurr = modelStart;
1818              modelStart = modelStart->next;
1819              free(modelCurr);
1820          }
1821          modelCurr = modelStart = NULL;
1822      }
1823  }
```

In this function, I am freeing up the memory allocated by *modelStart* linked list. In here, I am clearing the list from top to bottom. To do that, I am setting *modelCurr* to the head node of the list. Then changing the head node *modelStart* to the next one. Now that the previous head node which *modelCurr* was pointing to is disconnected from the list, I am freeing it up. Then the process keeps going on until it reaches end of the list.

freeSupplierList

```
1825  void freeSupplierList()
1826  {
1827      if (supplierStart == NULL) //IF the list is empty already
1828          supplierCurr = NULL;
1829      else if (supplierStart->next == NULL) //If there there is only one node
1830      {
1831          free(supplierStart);
1832          supplierStart = supplierCurr = NULL;
1833      }
1834      else //If list has two or more nodes
1835      {
1836          //Delete nodes from beginning to end
1837          while (supplierStart != NULL)
1838          {
1839              supplierCurr = supplierStart;
1840              supplierStart = supplierStart->next;
1841              free(supplierCurr);
1842          }
1843          supplierCurr = supplierStart = NULL;
1844      }
1845  }
```

This function frees up the memory allocated by *supplierStart* linked list. It works the same way as *freeModelList* function.

freePartList

```
1847    void freePartList()
1848    {
1849        if (partStart == NULL) //IF the list is empty already
1850            partCurr = NULL;
1851        else if (partStart->next == NULL) //If there there is only one node
1852        {
1853            free(partStart);
1854            partStart = partCurr = NULL;
1855        }
1856        else //If list has two or more nodes
1857        {
1858            //Delete nodes from beginning to end
1859            while(partStart != NULL)
1860            {
1861                partCurr = partStart;
1862                partStart = partStart->next;
1863                free(partCurr);
1864            }
1865            partCurr = partStart = NULL;
1866        }
1867    }
```

This function frees up the memory allocated by *partStart* linked list. It works the same way as *freeModelList* function. This function gets called in every major function that calls *readExistingParts*.

rtrim

```
1788    char* rtrim(char* str)
1789    {
1790        //Loop through the string character by character from end to beginning
1791        for (int i = strlen(str) - 1; i >= 0; i--)
1792        {
1793            if (str[i] != ' ' && str[i] != '\n')    //If end of empty spaces or new line char found
1794            {
1795                str[i + 1] = '\0'; //End the string by putting null char
1796                break;
1797            }
1798        }
1799        return str;
1800    }
```

rtrim function gets used when reading input from user or reading from file to clear out the empty spaces and new lines at the end of the string. This function loops from right to left of a string character by character. Once it finds a character other than *newline* character or space, it stops and puts *null* character to state end of string.

upperCase

```
1869    char* upperCase(char* str)
1870    {
1871        //Loop through characters of the string from end to beginning
1872        for (int i = strlen(str) - 1; i >= 0; i--)
1873            str[i] = toupper(str[i]);
1874
1875        return str;
1876    }
```

upperCase function loops through a string character by character and converts every character to an upper-case character using *toupper* function. It gets used when taking code input from user.

lowerCase

```
1878    char* lowerCase(char* str)
1879    {
1880        //Loop through characters of the string from end to beginning
1881        for (int i = strlen(str) - 1; i >= 0; i--)
1882            str[i] = tolower(str[i]);
1883
1884        return str;
1885    }
```

lowerCase function works just like *upperCase* function. Instead of using *toupper* function, it uses *tolower* function. This function gets used to make the search case insensitive.

ADDITIONAL FEATURES

Additional features are the features that were not required by the question but designed to provide a better user experience and to improve the integrity of the program. There are quite a few additional features that have been implemented in this program.

ABILITY TO ADD AND DELETE A MODEL

An automobile manufacturing company starts with a fixed number of models with plans to manufacture new models in the future. With this scenario in mind, I added a feature to add a new model instead of working with fixed number of models. Benefit of having this feature is that the source code does not need to be edited every time the company decides to add a new model to the system. They can do it from the program itself. All the model information is stored in a master file called *carModels.txt*.

When a user wants to add a new model, *addNewModel* function gets called. It allocates memory for the new model information. Then it prompts user for new model code and model name to update the newly allocated memory. Every input gets validated in various stages which is described in detail later in the report. Once the input has been validated, the *modelStart* linked list gets sorted in alphabetical order by model code. For this, I only need to sort the newly added model. After sorting, it is time to update into file. At first, I update the *carModels.txt* with the sorted and updated linked list using write mode. And then I create a new txt file with model code as its name and write the header row in it.

There could be a scenario where the company decides to shut down an old model. For that, I have added a feature to delete the model. When they decide to it, *deleteModel* function gets called which shows all the available models to the user. User chooses one of those models and gives confirmation to delete. After getting the confirmation, the program deletes the node with this model from the list and writes the updated linked list into the *carModels.txt.* Then the program deletes the file for this model.

ABILITY TO ADD AND DELETE A PART

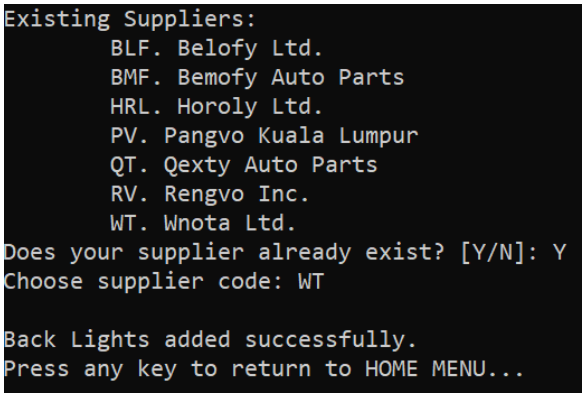
Just like models, I am not hardcoding any parts of a model. As I am not hardcoding it, the user must input all the parts one by one. That is where *addNewPart* function comes in. This gives the user the ability to add as many parts as they want.

When a user wants to add a part, program shows them available models and sections and prompts for model and section input. Once they choose it, it asks for part code, part name and quantity. After getting the information, the program validates each input. Using model code, section code and part code a unique ID is generated for the new part by joining each code with a hyphen "-". For example, A part with a part code EH from Blaze (BZ) model and Engine (EN) section will have an ID BZ-EN-EH. As for the supplier details, there are two cases: either the supplier already exists in the master file *supplier.txt* or it does not. So, for this reason, the program shows all the existing supplier and prompts user if the supplier of this part already exists. If yes it prompts for supplier code choice. Otherwise, it asks for supplier code and supplier name. Once the program gets all the information, it sorts the linked list and writes the sorted linked list into appropriate model file. If there is a new supplier, it also sorts *supplierStart* linked list and writes into *supplier.txt.*

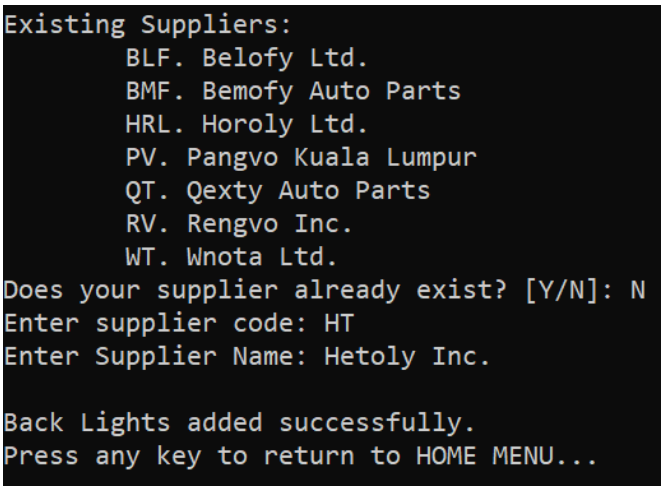
Just like adding functionality, there is also a functionality to delete a part. *deletePart* function prompts user for model choice, section choice and part choice. Once it gets all these information, it deletes the node with the chosen part from the *partStart* linked list and writes the updated linked list into the chosen model file.

ABILITY TO ADD OR CHOOSE SUPPLIER

According to the question, a supplier can provide more than one part. When a supplier provides more than one part, and if we take supplier details from the user there is a big chance of having duplicates. Instead of doing that, I have two options. As soon as they provide all the details about a part, the program shows all the existing suppliers, if there is any and then asks whether the supplier of the new part already exists, or it is a new one.



If the supplier already exists, then they can input Y and choose the supplier from the list, as shown in the figure above. Otherwise, program will prompt for new supplier code and name as shown in the figure below and validates before adding the part. For the new supplier, it gets added to the *supplierStart* linked list and gets sorted. Then the program writes the linked list into *suppliers.txt.*



INPUT VALIDATION

There are quite a few types of input validations in the program. We'll go through one by one.

CODE INPUT:

```
787     //Model Code input and validation
788     //Validations: Not empty, length >= 2, length <= 5, Unique, alpha
789     char modelCode[25];
790     int modelCodeValid = 1;
791     do
792     {
793         modelCodeValid = 1;
794         printf("Enter model code: ");
795         gets_s(modelCode);
796         strcpy_s(modelCode, upperCase(rtrim(modelCode)));
797         if (strcmp(modelCode, "") == 0)
798         {
799             printf("Please provide a model code. Try again\n");
800             modelCodeValid = 0;
801             continue;
802         }
803         if (strlen(modelCode) > 5)
804         {
805             printf("Model code cannot be more than 5 characters long. Try again\n");
806             modelCodeValid = 0;
807             continue;
808         }
809         if (strlen(modelCode) < 2)
810         {
811             printf("Model code must be at least 2 characters long. Try again\n");
812             modelCodeValid = 0;
813             continue;
814         }
```

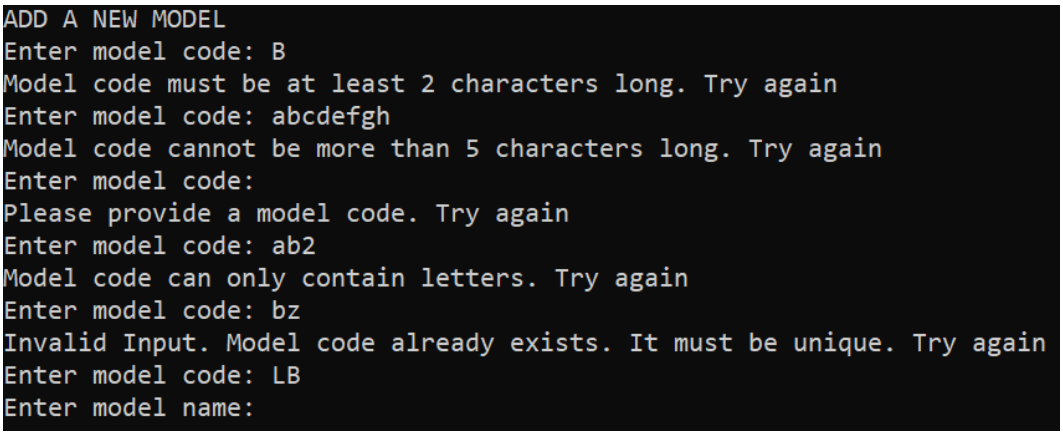
```
815         for (int i = strlen(modelCode) - 1; i >=  0 && modelCodeValid == 1; i--)
816         {
817             if (isalpha(modelCode[i]) == 0)
818             {
819                 printf("Model code can only contain letters. Try again\n");
820                 modelCodeValid = 0;
821                 break;
822             }
823         }
824         //Looking for duplicate
825         modelCurr = modelStart;
826         while (modelCurr->next != NULL && modelCodeValid == 1)
827         {
828             if (strcmp(modelCurr->code, modelCode) == 0)
829             {
830                 printf("Invalid Input. Model code already exists. It must be unique. Try again\n");
831                 modelCodeValid = 0;
832                 break;
833             }
834             modelCurr = modelCurr->next;
835         }
836     } while (modelCodeValid == 0);
837     strcpy_s(modelCurr->code, modelCode);
```

Model code, part code and supplier code have the same input validation. But before going into validation, I convert all the letters of the code to uppercase using a custom function named *upperCase*, which loops through all the characters of a string and converts any lowercase character to uppercase. I also used another custom function called *rtrim*, which gets

rid of the extra spaces from the right of the string, if there is any. These two things help validating the input. Then it goes through five validations. It keeps prompting for input until the input is valid.

- Code cannot be empty
- Minimum length of the code is 2
- Maximum length of the code is 5
- Code can only contain letters
- Code must be unique in a model

First four validations are quite simple. I used *strcmp* function to check if it's empty, *strlen* function to find the length and lastly *isalpha* function to check if all the characters are letters. For the uniqueness test, I looped through the *partStart* global linked list which holds all the information about existing parts under this model and checked whether there was a match using *strcmp*. Since the code has been converted into uppercase, "bz" and "BZ" will be considered as duplicate. Because of the trimming "BZ " and "BZ" will also be considered duplicate.



NAME INPUT:

```
839     //Model name input and validation
840     //Validation: Not empty, must contain at least one letter
841     int modelNameValid = 1;
842     do
843     {
844         modelNameValid = 1;
845         printf("Enter model name: ");
846         gets_s(modelCurr->name);
847         strcpy_s(modelCurr->name, rtrim(modelCurr->name));
848
849         if (strcmp(modelCurr->name, "") == 0)
850         {
851             printf("Please provide a model name. Try again\n");
852             modelNameValid = 0;
853             continue;
854         }
855         for (int i = strlen(modelCurr->name) - 1; i >= 0; i--)
856         {
857             if (isalpha(modelCurr->name[i]) != 0)
858             {
859                 modelNameValid = 1;
860                 break;
861             }
862             modelNameValid = 0;
863         }
864         if (modelNameValid == 0)
865             printf("Invalid input. Try again\n");
866
867     } while (modelNameValid == 0);
```

This validation is used for model name, part name and supplier name input. Right after taking the input, I trim them using *rtrim* custom function. Then it goes through two validations. It keeps prompting for input until the input is valid. Validations are:

- Name cannot be empty
- Name must contain at least one letter

Checking whether the name is empty is very easy. I used *strcmp* function to do that. As for the second one, I am looping through the characters of the name until I find a letter using *strlen* and *isalpha* functions.

```
ADD A NEW MODEL
Enter model code: LB
Enter model name:
Please provide a model name. Try again
Enter model name: 231
Invalid input. Try again
Enter model name: Lambo Red
LB.txt created successfully.
```

STRING CHOICE INPUT:

```
953     //Loop through model linked list and show all models
954     printf("Available Models: \n");
955     modelCurr = modelStart;
956     while (modelCurr != NULL)
957     {
958         printf("\t%s. %s\n", modelCurr->code, modelCurr->name);
959         modelCurr = modelCurr->next;
960     }
961
962     //Model code choice input and validation
963     int matchFound = 0;
964     char choice[25];
965     do
966     {
967         printf("Choose Model Code: ");
968         gets_s(choice);
969         strcpy_s(choice, upperCase(choice));
970
971         modelCurr = modelStart;
972         while (modelCurr != NULL)
973         {
974             if (strcmp(modelCurr->code, choice) == 0)
975             {
976                 matchFound = 1;
977                 break;
978             }
979             modelCurr = modelCurr->next;
980         }
981         if (matchFound == 0)
982             printf("Invalid input. Try again.\n");
983
984     } while (matchFound == 0);
```

When the user is given to choose from one of the available options and the input is asking for a string, then this validation gets used. For example, user is very often prompted to choose from one of the available models. As we can see in the code from *getModelCode* function. I used a while loop to show all the available models from *modelStart* linked list in line 956. Then I used the same linked list to validate the input *choice* is one of the model codes and not some random values. If the user provides invalid input, then program will keep prompting for input.

```
DELETE A MODEL
Available Models:
        AR. Armer
        BZ. Blaze
        GDZ. Godzilla
        GZ. Godzilla
        LB. Lambo Red
        RP. Raptor
Choose Model Code: ZZz
Invalid input. Try again.
Choose Model Code:
Invalid input. Try again.
Choose Model Code: 1
Invalid input. Try again.
```

INTEGER INPUT:

```
270     while (operation == 2) //Subtract quantity
271     {
272         if (partCurr->qty == 0)
273         {
274             printf("Sorry. Part out of stock.\n");
275             return;
276         }
277         printf("Enter quantity to subtract: ");
278         scanf_s("%d", &qty); while (getchar() != '\n');
279 
280         if (qty < 1 || qty > partCurr->qty)
281             printf("Invalid quantity. Try again.\n");
282         else
283         {
284             partCurr->qty -= qty;
285             break;
286         }
287     }
```

When the program prompts user for an integer input, first thing we must make sure the program does not throw error when the user provides a string or a character instead of an integer. To do that, I set the variable that is taking integer input, in this case *qty*, to zero. On top of that, I am clearing the buffer right after scanning for input. After that I checked if the provided input is required range. For example, requested quantity of a part must be less than or equal to the available quantity. In such scenario, program will keep prompting user for input.

```
Operations:
        1. Add Quantity [Supplier to Warehouw]
        2. Subtract Quantity [Warehouse to Assembly]
Choose Operation: 2
Enter quantity to subtract: a
Invalid quantity. Try again.
Enter quantity to subtract: 0
Invalid quantity. Try again.
Enter quantity to subtract: 100000
Invalid quantity. Try again.
```

NON-CASE SENSITIVE SEARCH BY KEYWORD

```
384             //Look for match
385             if (strstr(partCurr->id, keyword))
386                 found = 1;
387             else if (strstr(partCurr->modelCode, keyword))
388                 found = 1;
389             else if (strstr(upperCase(modelName), keyword))
390                 found = 1;
391             else if (strstr(partCurr->sectionCode, keyword))
392                 found = 1;
393             else if (strstr(upperCase(sectionName), keyword))
394                 found = 1;
395             else if (strstr(partCurr->code, keyword))
396                 found = 1;
397             else if (strstr(upperCase(partName), keyword))
398                 found = 1;
399             else if (strstr(partCurr->supplierCode, keyword))
400                 found = 1;
401             else if (strstr(upperCase(supplierName), keyword))
402                 found = 1;
403
404             //If found, print rows of parts data
405             if (found)
406             {
407                 printf("\t%-20s", partCurr->id);
408                 printf("\t%-20s", partCurr->modelName);
409                 printf("\t%-20s", partCurr->sectionName);
410                 printf("\t%-20s", partCurr->code);
411                 printf("\t%-20s", partCurr->name);
412                 printf("\t%-20d", partCurr->qty);
413                 printf("\t%-20s\n", partCurr->supplierName);
414                 matchFound++;
415             }
416             found = 0;
```

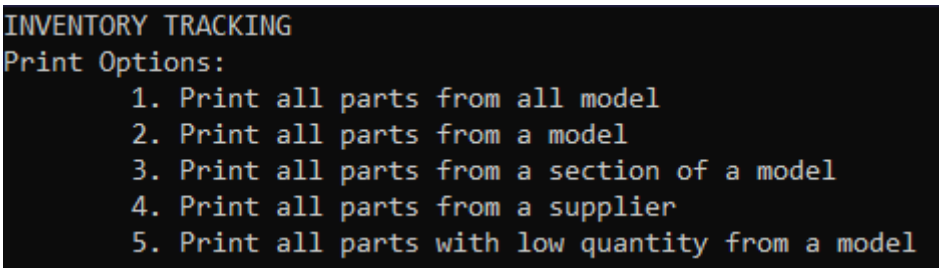
Requirement of the question was to be able to search using part ID or supplier. Instead of doing that, I made the search function to be able to search using keyword. And this keyword is NOT case sensitive. The program goes through every part of every model using nested loops. It looks for a match in 9 elements of a part. These are, part ID, model code, model name, section code, section name, part code, part name, supplier code, supplier name. If it finds a match, it shows that part and moves on to the next one. Main reason of having all

the possible information in the *part* data type is to get better search experience. At the end of the search results, it also shows the total number of matches found.

```
SEARCH INVENTORY
Enter Search Keyword: e
        PART ID                 MODEL NAME              SECTION NAME            PART CODE               PART NAME
        AR-BD-BN                Armer                   Body Work               BN                      Bonnet
        AR-BD-BP                Armer                   Body Work               BP                      Bumper
        AR-BD-CC                Armer                   Body Work               CC                      Car Cover
        AR-BD-HG                Armer                   Body Work               HG                      Hinges
        AR-BD-SS                Armer                   Body Work               SS                      Support Stick
        AR-CS-CR                Armer                   Chassis                 CR                      Coupling Rod
        AR-EN-ENB               Armer                   Engine                  ENB                     Engine Block
        AR-OT-LT                Armer                   Others                  LT                      Lights
        BZ-BD-BT                Blaze                   Body Work               BT                      Bonnet
        GDZ-EN-EH               Godzilla                Engine                  EH                      Engine Holder
Match Found: 10
```

```
SEARCH INVENTORY
Enter Search Keyword: en
        PART ID                 MODEL NAME              SECTION NAME            PART CODE               PART NAME
        AR-BD-BN                Armer                   Body Work               BN                      Bonnet
        AR-EN-ENB               Armer                   Engine                  ENB                     Engine Block
        GDZ-EN-EH               Godzilla                Engine                  EH                      Engine Holder
Match Found: 3
```

ADVANCED INVENTORY TRACKING



Inventory tracking in my program is very advanced. It has five different options users can choose from where question only asked for two. The options are self-explanatory. First option loops through each part of each model using nested loops. Second option prompts user to choose one of the available models and loops through the parts of that specific model. Third options user for both model and section and only shows the parts from that section of the model. Fourth option prompts user to choose one of the suppliers and loops through every part of every model using nested loops and only shows the ones with matching supplier. As for the last one, it asks for a model as per question requirements. Then it asks for a low quantity threshold instead of a fixed number. Then shows all the parts with quantity less than the threshold from that model.

```
INVENTORY TRACKING
Print Options:
        1. Print all parts from all model
        2. Print all parts from a model
        3. Print all parts from a section of a model
        4. Print all parts from a supplier
        5. Print all parts with low quantity from a model
Enter choice of operation: 5
Available Models:
        AR. Armer
        BZ. Blaze
        GDZ. Godzilla
        GZ. Godzilla
        LB. Lambo Red
        RP. Raptor
Choose Model Code: AR
Enter Low Quantity threshhold: 50
        PART ID                 MODEL NAME              SECTION NAME            PART CODE               PART NAME               QUANTITY
        AR-BD-BN                Armer                   Body Work               BN                      Bonnet                  21
        AR-BD-BP                Armer                   Body Work               BP                      Bumper                  22
        AR-BD-CC                Armer                   Body Work               CC                      Car Cover               25
        AR-BD-SS                Armer                   Body Work               SS                      Support Stick           20
        AR-CS-CR                Armer                   Chassis                 CR                      Coupling Rod            30
        AR-OT-LT                Armer                   Others                  LT                      Lights                  22
Match Found: 6
Press any key to return to HOME MENU...
```

SAMPLE INPUT/OUTPUT

MAIN MENU

```
WELCOME TO AUTOMOBILE PARTS INVENTORY MANAGEMENT SYSTEM

HOME MENU:
        1. Update inventory
        2. Inventory tracking
        3. Search inventory
        4. Add a new model
        5. Add a new part
        6. Delete a model
        7. Delete a part
        0. Exit
Enter your choice of operation:
```

In the main menu, user has 8 options to choose from and after each operation ends it brings them back to it. If the user provides any invalid input, it will show an error message and prompt for input again.

UPDATE INVENTORY

```
UPDATE INVENTORY
No available models. Add a model first.
Press any key to return to HOME MENU...
```

First option in home menu is updating the inventory. If there is no model available meaning the user have not added any model yet, then it will show a message and bring the user back to home menu.

```
UPDATE INVENTORY
Available Models:
        AR. Armer
        BZ. Blaze
        GDZ. Godzilla
        GZ. Godzilla
        LB. Lambo Red
        RP. Raptor
Choose Model Code: LB
No available parts for this model. Add a part first.
Press any key to return to HOME MENU...
```

Same thing happens when the user chooses a model with no part in it. This extra layer of validation increases the integrity of the program and makes sure it does throw an error.

```
UPDATE INVENTORY
Available Models:
       AR. Armer
       BZ. Blaze
       GDZ. Godzilla
       GZ. Godzilla
       LB. Lambo Red
       RP. Raptor
Choose Model Code: AR
Available Parts:
       PART ID              MODEL NAME          SECTION NAME        PART CODE           PART NAME           QUANTITY
       AR-BD-BN             Armer               Body Work           BN                  Bonnet              121
       AR-BD-BP             Armer               Body Work           BP                  Bumper              22
       AR-BD-CC             Armer               Body Work           CC                  Car Cover           25
       AR-BD-HG             Armer               Body Work           HG                  Hinges              74
       AR-BD-SS             Armer               Body Work           SS                  Support Stick       20
       AR-CS-CR             Armer               Chassis             CR                  Coupling Rod        30
       AR-EN-ENB            Armer               Engine              ENB                 Engine Block        54
       AR-OT-LT             Armer               Others              LT                  Lights              22
Choose Part Code: BN
Operations:
       1. Add Quantity [Supplier to Warehouse]
       2. Subtract Quantity [Warehouse to Assembly]
Choose Operation:
```

Once the user chooses model and part properly, program will show two operations to choose from. Either add quantity of a part by importing from supplier to warehouse or subtract quantity by transferring from warehouse to assembly area.

```
Operations:
       1. Add Quantity [Supplier to Warehouse]
       2. Subtract Quantity [Warehouse to Assembly]
Choose Operation: 1
Enter quantity to add: 50
Inventory successfully updated.
Press any key to return to HOME MENU...
```

```
Operations:
       1. Add Quantity [Supplier to Warehouse]
       2. Subtract Quantity [Warehouse to Assembly]
Choose Operation: 2
Enter quantity to subtract: 9000
Invalid quantity. Try again.
Enter quantity to subtract: 100
Inventory successfully updated.
Press any key to return to HOME MENU...
```

For both options, program will prompt the user for quantity input. Once they provide a valid input, it will do the chosen operation. As for invalid input, program will keep prompting the user until a valid input is provided.

INVENTORY TRACKING

```
INVENTORY TRACKING
No available models. Add a model first.
Press any key to return to HOME MENU...
```

```
INVENTORY TRACKING
Print Options:
       1. Print all parts from all model
       2. Print all parts from a model
       3. Print all parts from a section of a model
       4. Print all parts from a supplier
       5. Print all parts with low quantity from a model
Enter choice of operation:
```

In the inventory tracking, if there is no model available, the program will show a message and return to home menu. However, if there is at least one model, it will show 5 options to choose from. Each option has been described below:

```
INVENTORY TRACKING
Print Options:
        1. Print all parts from all model
        2. Print all parts from a model
        3. Print all parts from a section of a model
        4. Print all parts from a supplier
        5. Print all parts with low quantity from a model
Enter choice of operation: 1
        PART ID                 MODEL NAME              SECTION NAME            PART CODE               PART NAME               QUANTITY
        AR-BD-BN                Armer                   Body Work               BN                      Bonnet                  21
        AR-BD-BP                Armer                   Body Work               BP                      Bumper                  22
        AR-BD-CC                Armer                   Body Work               CC                      Car Cover               25
        AR-BD-HG                Armer                   Body Work               HG                      Hinges                  74
        AR-BD-SS                Armer                   Body Work               SS                      Support Stick           20
        AR-CS-CR                Armer                   Chassis                 CR                      Coupling Rod            30
        AR-EN-ENB               Armer                   Engine                  ENB                     Engine Block            54
        AR-OT-LT                Armer                   Others                  LT                      Lights                  22
        BZ-BD-BT                Blaze                   Body Work               BT                      Bonnet                  20
        GDZ-EN-EH               Godzilla                Engine                  EH                      Engine Holder           24
        RP-BD-BN                Raptor                  Body Work               BN                      Bonnet                  22
        RP-OT-BT                Raptor                  Others                  BT                      Back Lights             20
Match Found: 12
Press any key to return to HOME MENU...
```

For the first option, the program will show all the available parts under all models. It also shows supplier name, which is not included in the screenshot.

```
INVENTORY TRACKING
Print Options:
        1. Print all parts from all model
        2. Print all parts from a model
        3. Print all parts from a section of a model
        4. Print all parts from a supplier
        5. Print all parts with low quantity from a model
Enter choice of operation: 2
Available Models:
        AR. Armer
        BZ. Blaze
        GDZ. Godzilla
        GZ. Godzilla
        LB. Lambo Red
        RP. Raptor
Choose Model Code: AR
        PART ID                 MODEL NAME              SECTION NAME            PART CODE               PART NAME               QUANTITY
        AR-BD-BN                Armer                   Body Work               BN                      Bonnet                  21
        AR-BD-BP                Armer                   Body Work               BP                      Bumper                  22
        AR-BD-CC                Armer                   Body Work               CC                      Car Cover               25
        AR-BD-HG                Armer                   Body Work               HG                      Hinges                  74
        AR-BD-SS                Armer                   Body Work               SS                      Support Stick           20
        AR-CS-CR                Armer                   Chassis                 CR                      Coupling Rod            30
        AR-EN-ENB               Armer                   Engine                  ENB                     Engine Block            54
        AR-OT-LT                Armer                   Others                  LT                      Lights                  22
Match Found: 8
Press any key to return to HOME MENU...
```

For the second option, user is shown a list of available models. User is prompted to choose one of these models. It will show all the parts from this model.

```
Print Options:
        1. Print all parts from all model
        2. Print all parts from a model
        3. Print all parts from a section of a model
        4. Print all parts from a supplier
        5. Print all parts with low quantity from a model
Enter choice of operation: 3
Available Models:
        AR. Armer
        BZ. Blaze
        GDZ. Godzilla
        GZ. Godzilla
        LB. Lambo Red
        RP. Raptor
Choose Model Code: AR
Available sections:
        BD. Body
        CS. Chassis
        EN. Engine
        TR. Transmission System
        OT. Others
Choose Section Code: BD
        PART ID                 MODEL NAME              SECTION NAME            PART CODE               PART NAME               QUANTITY
        AR-BD-BN                Armer                   Body Work               BN                      Bonnet                  21
        AR-BD-BP                Armer                   Body Work               BP                      Bumper                  22
        AR-BD-CC                Armer                   Body Work               CC                      Car Cover               25
        AR-BD-HG                Armer                   Body Work               HG                      Hinges                  74
        AR-BD-SS                Armer                   Body Work               SS                      Support Stick           20
Match Found: 5
Press any key to return to HOME MENU...
```

For the third option, it goes into another level. It prompts for model as well as section choice. Once the user provides valid inputs, it shows all the parts from the chosen section of the model.

```
INVENTORY TRACKING
Print Options:
        1. Print all parts from all model
        2. Print all parts from a model
        3. Print all parts from a section of a model
        4. Print all parts from a supplier
        5. Print all parts with low quantity from a model
Enter choice of operation: 4
Existing Suppliers:
        BLF. Belofy Ltd.
        BMF. Bemofy Auto Parts
        HRL. Horoly Ltd.
        HT. Hetoly Inc.
        LB. Lambo
        PV. Pangvo Kuala Lumpur
        QT. Qexty Auto Parts
        RV. Rengvo Inc.
        WT. Wnota Ltd.
Choose supplier code: BLF
        PART ID                 MODEL NAME              SECTION NAME            PART CODE               PART NAME               QUANTITY                SUPPLIER NAME
        AR-BD-BP                Armer                   Body Work               BP                      Bumper                  22                      Belofy Ltd.
        AR-OT-LT                Armer                   Others                  LT                      Lights                  22                      Belofy Ltd.
Match Found: 2
Press any key to return to HOME MENU...
```

For the fourth option, program shows a list of all the available models. User gets prompted to choose one of the models. Then the program shows all the parts from the selected supplier across all models.

```
INVENTORY TRACKING
Print Options:
        1. Print all parts from all model
        2. Print all parts from a model
        3. Print all parts from a section of a model
        4. Print all parts from a supplier
        5. Print all parts with low quantity from a model
Enter choice of operation: 5
Available Models:
        AR. Armer
        BZ. Blaze
        GDZ. Godzilla
        GZ. Godzilla
        LB. Lambo Red
        RP. Raptor
Choose Model Code: AR
Enter Low Quantity threshhold: 50
        PART ID                 MODEL NAME              SECTION NAME            PART CODE               PART NAME               QUANTITY
        AR-BD-BN                Armer                   Body Work               BN                      Bonnet                  21
        AR-BD-BP                Armer                   Body Work               BP                      Bumper                  22
        AR-BD-CC                Armer                   Body Work               CC                      Car Cover               25
        AR-BD-SS                Armer                   Body Work               SS                      Support Stick           20
        AR-CS-CR                Armer                   Chassis                 CR                      Coupling Rod            30
        AR-OT-LT                Armer                   Others                  LT                      Lights                  22
Match Found: 6
Press any key to return to HOME MENU...
```

For the last option, user gets prompted to choose one of the available options. Then the program asks for low quantity threshold, the quantity that is considered as low. Then it shows all the parts from the model with quantity less than the threshold.

SEARCH INVENTORY

```
SEARCH INVENTORY
No available models. Add a model first.
Press any key to return to HOME MENU...
```

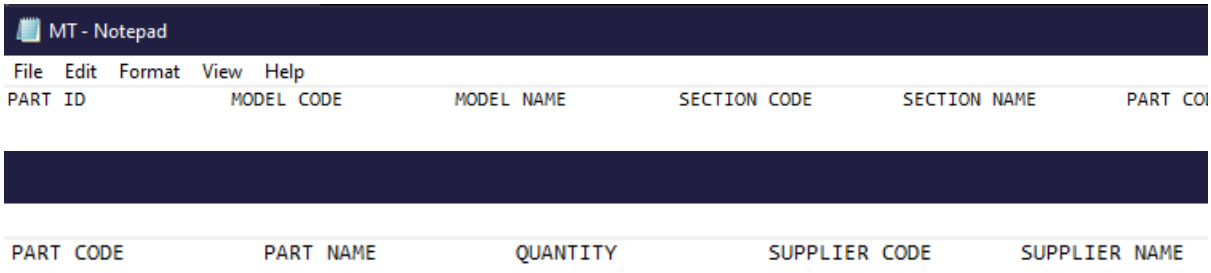
```
SEARCH INVENTORY
Enter Search Keyword: en
       PART ID              MODEL NAME           SECTION NAME         PART CODE            PART NAME            QUANTITY
       AR-BD-BN             Armer                Body Work            BN                   Bonnet               21
       AR-EN-ENB            Armer                Engine               ENB                  Engine Block         54
       GDZ-EN-EH            Godzilla             Engine               EH                   Engine Holder        24
Match Found: 3
Press any key to return to HOME MENU...
```

Search functionality of this program is so advanced. It prompts user for a keyword, and it goes through all the models and look for a match. It is case insensitive and it looks through every information of a part. When it finds a match, it prints the information of that model.

ADD A NEW MODEL

```
ADD A NEW MODEL
Enter model code: MT
Enter model name: Mountain
MT.txt created successfully.
Press any key to return to HOME MENU...
```

Adding a model is very simple. It prompts for model code and model name. But the program is very picky about the model code. Model code goes through 5 different layers of validations which has been discussed in the additional features section. Once it gets the inputs, it creates a file with model code.



It also puts the header row right after creating the file. Each model file has 10 columns of part information which have been shown in 2 screenshots above.

ADD A NEW PART

```
ADD A NEW PART
No available model. Please add a model first.
Press any key to return to HOME MENU...
```

```
Available Models:
        AR. Armer
        BZ. Blaze
        GDZ. Godzilla
        GZ. Godzilla
        LB. Lambo Red
        MT. Mountain
        RP. Raptor
Choose Model Code: MT
Available Section Code:
        BD. Body Work
        CS. Chassis
        EN. Engine
        TR. Transmission System
        OT. Others
Choose Section Code: TR
Enter part code: GB
Enter Part Name: Gear Box
Enter Part Quantity: 10
Existing Suppliers:
        BLF. Belofy Ltd.
        BMF. Bemofy Auto Parts
        HRL. Horoly Ltd.
        HT. Hetoly Inc.
        LB. Lambo
        PV. Pangvo Kuala Lumpur
        QT. Qexty Auto Parts
        RV. Rengvo Inc.
        WT. Wnota Ltd.
Does your supplier already exist? [Y/N]:
```

When adding a new part, user first need to select the model and section of that part. In case there is no model available it will show a message and return to home menu. Once the user chooses model and section, he/she gets prompted with part code, part name and part quantity. After that, the program shows all the available suppliers and asked if they want to add a new supplier or choose from existing ones.

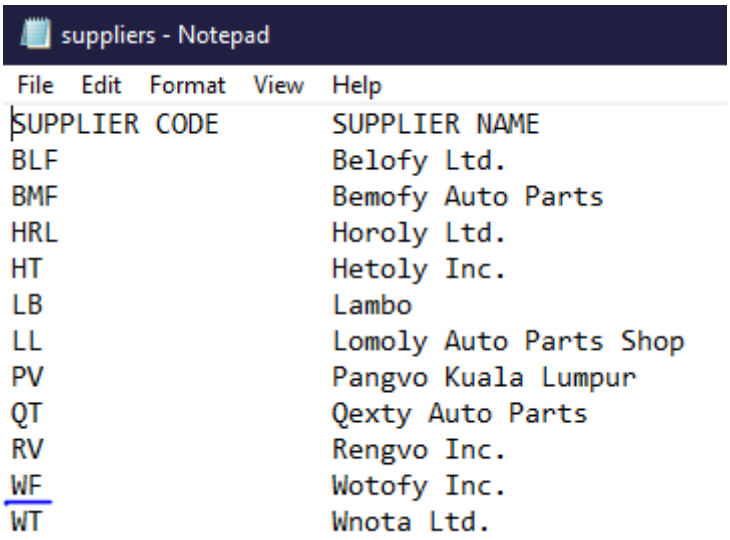
```
Does your supplier already exist? [Y/N]: y
Choose supplier code: PV

Gear Box added successfully.
Press any key to return to HOME MENU...
```

```
Does your supplier already exist? [Y/N]: n
Enter supplier code: WF
Enter Supplier Name: Wotofy Inc.

Gear Box added successfully.
Press any key to return to HOME MENU...
```

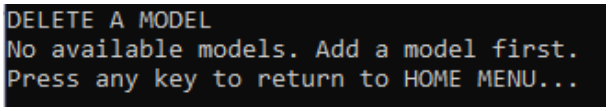
If the user selects to choose from existing one, he gets prompted for supplier code. Otherwise, program prompts for supplier code and supplier name.

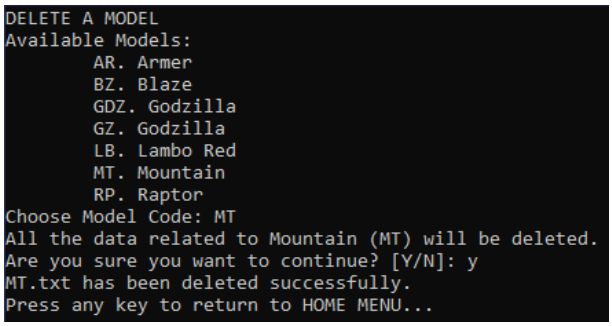
```
MT - Notepad
File  Edit  Format  View  Help
PART ID         MODEL CODE      MODEL NAME      SECTION CODE    SECTION NAME
MT-TR-GB        MT              Mountain        TR              Transmission System

                                                                                -

PART CODE       PART NAME       QUANTITY        SUPPLIER CODE   SUPPLIER NAME
GB              Gear Box        10              WF              Wotofy Inc.
```

The information gets updated into the model file which we can see in two screenshots above. Abd the new supplier details also gets updated in the supplier master file *suppliers.txt.*



DELETE A MODEL





The program makes sure there are available models before deleting one. If there is no model available, then the program shows a helpful message and returns to home menu. However, if there are available models, it shows them and prompts user for model code choice. Once the user chooses the model, the program asks for confirmation. If the user confirms, then the program deletes the model information from the model master file *carModels*.txt and deletes the file for this model. It becomes like the model never existed in the first place.

DELETE A PART

```
DELETE A PART
No available models. Add a model first.
Press any key to return to HOME MENU...
```

```
DELETE A PART
Available Models:
        AR. Armer
        BZ. Blaze
        GDZ. Godzilla
        GZ. Godzilla
        LB. Lambo Red
        MT. Mountain
        RP. Raptor
Choose Model Code: MT
No available parts for this model. Add a part first.
Press any key to return to HOME MENU...
```

If there are no available models when trying to delete a part, the program will show a helpful message and return to home menu. Same thing happens when the chosen model has no part under it.

```
DELETE A PART
Available Models:
        AR. Armer
        BZ. Blaze
        GDZ. Godzilla
        GZ. Godzilla
        LB. Lambo Red
        MT. Mountain
        RP. Raptor
Choose Model Code: MT
Available Parts:
        PART ID                 MODEL NAME              SECTION NAME            PART CODE               PART NAME
        MT-TR-GB                Mountain                Transmission System     GB                      Gear Box
Choose Part Code: GB
All the data related to Gear Box (GB) of Mountain (MT) will be deleted.
Are you sure you want to continue? [Y/N]: y
Delete Process Successful
Press any key to return to HOME MENU...
```

Once the supplier chooses one of the parts and gives confirmation to delete, that part gets deleted from the database.

EXIT

```
WELCOME TO AUTOMOBILE PARTS INVENTORY MANAGEMENT SYSTEM

HOME MENU:
        1. Update inventory
        2. Inventory tracking
        3. Search inventory
        4. Add a new model
        5. Add a new part
        6. Delete a model
        7. Delete a part
        0. Exit
Enter your choice of operation: 0
Exiting program...
```

LIMITATIONS AND IMPROVEMENT

Even though I am very proud of how program turned out, there are a few limitations I would like to work on and improve the program. One of the big limitations of this program is it is using global pointers for three linked list. It is a big security risk because every function of this program can access these three linked lists, alter the data and even delete the data in them (Reddy, 2018). I would like to use local linked list and give necessary functions read or write access based on need. But it was very hard to work with linked lists and pointers as it is the first big project that I used linked list in.

In basic database, there are usually two types of files: master file and transaction file (Sharma, n.d.). In this program, I have only used master file like *carModels.txt, suppliers.txt, BZ.txt* and so on. I would like to have a transaction file. A *log.txt* where I could record every transaction with date, time and a short description of the transaction. With this, we could see the history of all the transaction over a period.

Length of every element of every linked list is set to 25 characters to make it easy to work with. It is taking some unnecessary space. I would have liked to use as little storage as possible. I did the similar thing when storing data into file to make it easy to work with. I used the length of every column as 20 even though in some places it is not necessary. I would like to fix this issue. There is another problem that rises from these two. If the user provides a name that is 24 characters long, linked list will take it properly. But when writing to file, last 4 characters will be chopped off. It is a big problem. We can easily solve it by using same length in both places. But that needs a lot more testing.

I have provided the ability to remove a *model* or *part* from the system. But there is no way to delete a supplier. Because when the user tries to remove a supplier, there are complications. Because there are parts that has this supplier. I would have liked to have an option to delete a supplier from the system with advanced logic.

There is no option to edit a model information or part information in the program. If there is a mistake found in the program in the later stage, the user would either have to edit the files to fix it or delete the model or part and add it again.

CONCLUSION

Working on this assignment was a great learning opportunity. I will be forever grateful to the lecturer for giving me the opportunity to work on such amazing assignment. I learned a lot by struggling and researching for this assignment.

First challenge in this assignment was to work with linked list. As I have never worked with linked list in such a big scale, it took some time to fully understand how it is working. By that time, I had gone through lots and lots of bugs and errors. I have learned how to add a node to linked list, how to remove one, how to sort the list and how to free up the memory.

Other big challenge was reading from and writing into file. Working with file is complicated thing. On top of that, I had read from file into the linked list and write from linked list into the file. There were a lot of errors on every stage of the program. Luckily, I managed to come up with solution over time.

REFERENCES

Appdynamics. (n.d.). *What is Database Management Systems (DBMS)?* https://www.appdynamics.com/topics/database-management-systems

GeeksforGeeks. (2021). *what happens when you don't free memory after using malloc()* https://www.geeksforgeeks.org/what-happens-when-you-dont-free-memory-after-using-malloc/

Reddy, A. (2018). *Why should we avoid using global variables in C/C++?* https://www.tutorialspoint.com/Why-should-we-avoid-using-global-variables-in-C-Cplusplus

Sharma, P. (n.d.). *Top 6 Types of Data Files Used in any Information System | MIS* https://www.yourarticlelibrary.com/management/information-system/top-6-types-of-data-files-used-in-any-information-system-mis/70373