A blue and white logo

Description automatically generated

**MINISTRY OF EDUCATION, CULTURE AND RESEARCH**

**OF THE REPUBLIC OF MOLDOVA**

**Technical University of Moldova**

**Faculty of Computers, Informatics and Microelectronics Department of Software and Automation Engineering**

**Iamandii Ion student**

**Group: FAF-233**

**Report**

**Laboratory Work No.6**

***of the "Data Structures and Algorithms" course***

Checked:

teacher name, surname, academic level

Department of Software and Automation Engineering,

FCIM Faculty, UTM

Verificat:

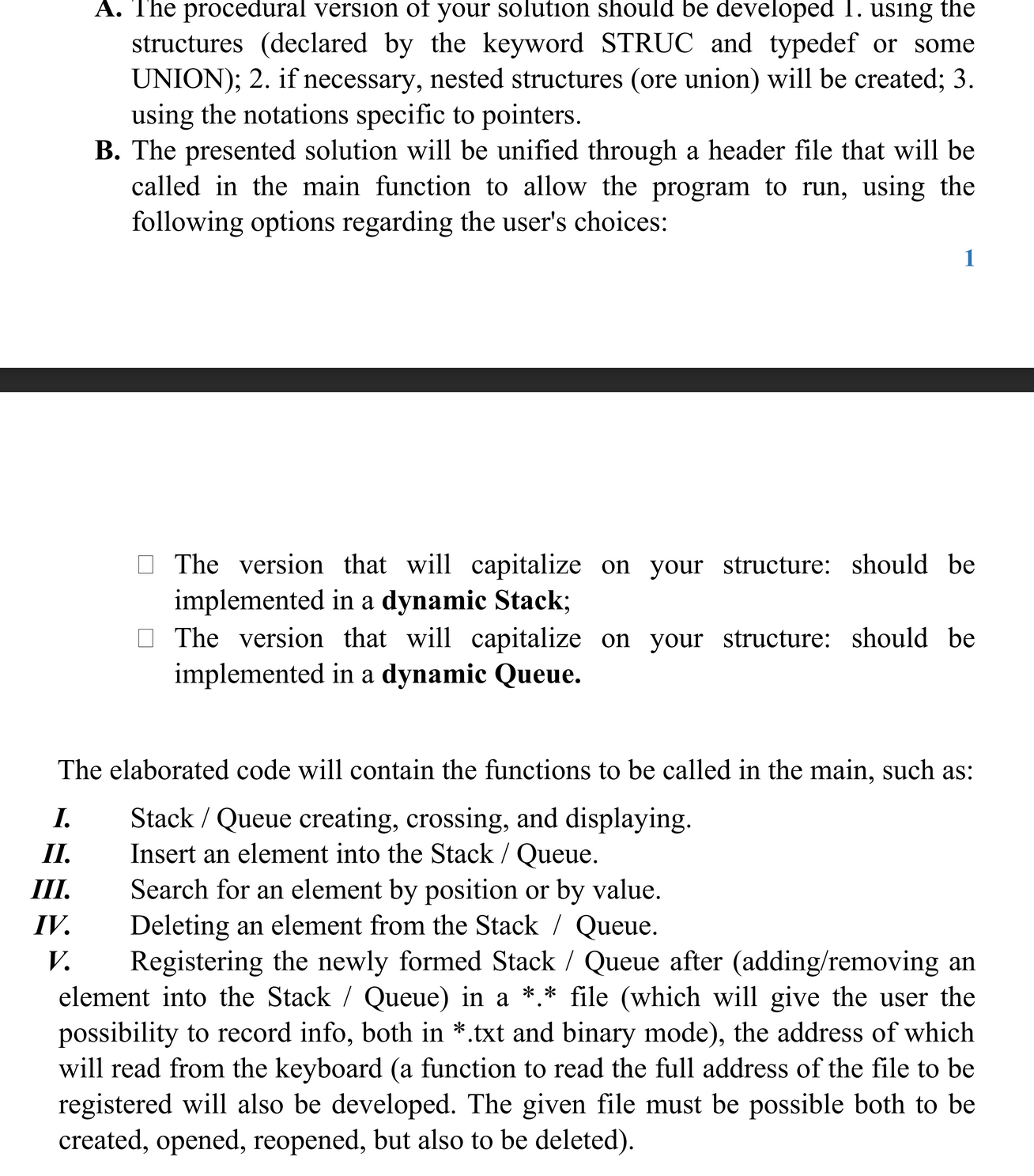
**Burlacu Natalia,** *PhD, associate professor*

Department of Software and Automation Engineering,

Facultatea FCIM, UTM

**Chisinau – 2024**

**Task 1**



1. **The code of the program, with relevant comments in it, and the Block diagram; Code:**

Part 1:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

struct WarehouseRecord {

    char owner\_name[50];

    char owner\_surname[50];

    char product\_name[50];

    char manufacturer[50];

    int contract\_date\_day;

    int contract\_date\_month;

    int contract\_date\_year;

    float wholesale\_price;

    float unit\_price;

    int quantity;

    int remaining\_years;

    int remaining\_months;

    int remaining\_days;

};

struct Node {

    struct WarehouseRecord data;

    struct Node\* next;

};

struct Stack {

    struct Node\* top;

};

struct Stack\* createStack();

void push(struct Stack\* stack, struct WarehouseRecord data);

struct WarehouseRecord pop(struct Stack\* stack);

void displayStack(struct Stack\* stack);

void insertRecord(struct Stack\* stack);

int searchRecord(struct Stack\* stack, char searchKey[]);

void deleteRecord(struct Stack\* stack, int position);

void saveStackToFile(struct Stack\* stack, char filename[]);

void readFilename(char filename[]);

void stackProgram() {

    struct Stack\* stack = createStack();

    int choice;

    char filename[100];

    do {

        printf("\n\nStack Operations Menu:\n");

        printf("1. Display Stack\n");

        printf("2. Insert Record\n");

        printf("3. Search Record\n");

        printf("4. Delete Record\n");

        printf("5. Save Stack to File\n");

        printf("6. Read Filename\n");

        printf("0. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                displayStack(stack);

                break;

            case 2:

                insertRecord(stack);

                break;

            case 3: {

                char searchKey[50];

                printf("Enter search key: ");

                scanf("%s", searchKey);

                int position = searchRecord(stack, searchKey);

                if (position == -1) {

                    printf("Record not found.\n");

                } else {

                    printf("Record found at position %d.\n", position);

                }

                break;

            }

            case 4: {

                int position;

                printf("Enter position to delete: ");

                scanf("%d", &position);

                deleteRecord(stack, position);

                break;

            }

            case 5:

                if (strcmp(filename, "") == 0) {

                    printf("Please enter a filename first.\n");

                } else {

                    saveStackToFile(stack, filename);

                    printf("Stack saved to file '%s'.\n", filename);

                }

                break;

            case 6:

                readFilename(filename);

                break;

            case 0:

                printf("Exiting...\n");

                break;

            default:

                printf("Invalid choice! Please enter a valid option.\n");

        }

    } while (choice != 0);

}

struct Stack\* createStack() {

    struct Stack\* stack = (struct Stack\*)malloc(sizeof(struct Stack));

    stack->top = NULL;

    return stack;

}

void push(struct Stack\* stack, struct WarehouseRecord data) {

    struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

    newNode->data = data;

    newNode->next = stack->top;

    stack->top = newNode;

}

struct WarehouseRecord pop(struct Stack\* stack) {

    if (stack->top == NULL) {

        printf("Stack underflow!\n");

        exit(1);

    }

    struct Node\* temp = stack->top;

    struct WarehouseRecord poppedData = temp->data;

    stack->top = temp->next;

    free(temp);

    return poppedData;

}

void displayStack(struct Stack\* stack) {

    if (stack->top == NULL) {

        printf("Stack is empty.\n");

        return;

    }

    struct Node\* current = stack->top;

    printf("Warehouse Records (top to bottom):\n");

    while (current != NULL) {

        printf("Owner: %s %s\n", current->data.owner\_name, current->data.owner\_surname);

        printf("Product Name: %s\n", current->data.product\_name);

        printf("Manufacturer: %s\n", current->data.manufacturer);

        printf("Contract Date: %d-%02d-%02d\n", current->data.contract\_date\_year, current->data.contract\_date\_month, current->data.contract\_date\_day);

        printf("Wholesale Price: %.2f\n", current->data.wholesale\_price);

        printf("Unit Price: %.2f\n", current->data.unit\_price);

        printf("Quantity: %d\n", current->data.quantity);

        printf("\n");

        current = current->next;

    }

}

void insertRecord(struct Stack\* stack) {

    struct WarehouseRecord newRecord;

    printf("Enter owner's name: ");

    scanf("%s", newRecord.owner\_name);

    printf("Enter owner's surname: ");

    scanf("%s", newRecord.owner\_surname);

    printf("Enter product name: ");

    scanf("%s", newRecord.product\_name);

    printf("Enter manufacturer: ");

    scanf("%s", newRecord.manufacturer);

    printf("Enter contract conclusion date (day month year): ");

    scanf("%d %d %d", &newRecord.contract\_date\_day, &newRecord.contract\_date\_month, &newRecord.contract\_date\_year);

    printf("Enter wholesale price: ");

    scanf("%f", &newRecord.wholesale\_price);

    printf("Enter unit price: ");

    scanf("%f", &newRecord.unit\_price);

    printf("Enter quantity (units): ");

    scanf("%d", &newRecord.quantity);

    push(stack, newRecord);

}

int searchRecord(struct Stack\* stack, char searchKey[]) {

    int position = 0;

    struct Node\* current = stack->top;

    while (current != NULL) {

        if (strcmp(current->data.product\_name, searchKey) == 0) {

            return position;

        }

        current = current->next;

        position++;

    }

    return -1;

}

void deleteRecord(struct Stack\* stack, int position) {

    if (stack->top == NULL) {

        printf("Stack is empty.\n");

        return;

    }

    if (position < 1) {

        printf("Invalid position.\n");

        return;

    }

    struct Node\* current = stack->top;

    struct Node\* prev = NULL;

    int count = 1;

    while (current != NULL && count != position) {

        prev = current;

        current = current->next;

        count++;

    }

    if (current == NULL) {

        printf("Position exceeds the size of the stack.\n");

        return;

    }

    if (prev == NULL) {

        stack->top = current->next;

    } else {

        prev->next = current->next;

    }

    free(current);

}

void saveStackToFile(struct Stack\* stack, char filename[]) {

    FILE\* file = fopen(filename, "w");

    if (file == NULL) {

        printf("Error opening file.\n");

        return;

    }

    struct Node\* current = stack->top;

    while (current != NULL) {

        fprintf(file, "Owner: %s %s\n", current->data.owner\_name, current->data.owner\_surname);

        fprintf(file, "Product Name: %s\n", current->data.product\_name);

        fprintf(file, "Manufacturer: %s\n", current->data.manufacturer);

        fprintf(file, "Contract Date: %d-%02d-%02d\n", current->data.contract\_date\_year, current->data.contract\_date\_month, current->data.contract\_date\_day);

        fprintf(file, "Wholesale Price: %.2f\n", current->data.wholesale\_price);

        fprintf(file, "Unit Price: %.2f\n", current->data.unit\_price);

        fprintf(file, "Quantity: %d\n", current->data.quantity);

        fprintf(file, "\n");

        current = current->next;

    }

    fclose(file);

}

void readFilename(char filename[]) {

    printf("Enter filename: ");

    scanf("%s", filename);

}

Part 2:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

struct Queue\_WarehouseRecord {

    char owner\_name[50];

    char owner\_surname[50];

    char product\_name[50];

    char manufacturer[50];

    int contract\_date\_day;

    int contract\_date\_month;

    int contract\_date\_year;

    float wholesale\_price;

    float unit\_price;

    int quantity;

    int remaining\_years;

    int remaining\_months;

    int remaining\_days;

};

struct Queue\_Node {

    struct Queue\_WarehouseRecord data;

    struct Queue\_Node \*next;

};

struct Queue {

    struct Queue\_Node \*front;

    struct Queue\_Node \*rear;

};

struct Queue\* Queue\_createQueue();

void Queue\_enqueue(struct Queue \*q, struct Queue\_WarehouseRecord record);

struct Queue\_WarehouseRecord Queue\_dequeue(struct Queue \*q);

void Queue\_displayQueue(struct Queue \*q);

void Queue\_insertRecord(struct Queue \*q);

int Queue\_searchRecordByValue(struct Queue \*q, struct Queue\_WarehouseRecord record);

int Queue\_searchRecordByPosition(struct Queue \*q, int position);

void Queue\_deleteRecord(struct Queue \*q, struct Queue\_WarehouseRecord record);

void Queue\_registerQueueToFile(struct Queue \*q, char \*filename);

char\* Queue\_readFilenameFromKeyboard();

struct Queue\* Queue\_createQueue() {

    struct Queue \*q = (struct Queue\*)malloc(sizeof(struct Queue));

    if (!q) {

        printf("Memory allocation error!\n");

        exit(EXIT\_FAILURE);

    }

    q->front = q->rear = NULL;

    return q;

}

void Queue\_enqueue(struct Queue \*q, struct Queue\_WarehouseRecord record) {

    struct Queue\_Node \*newNode = (struct Queue\_Node\*)malloc(sizeof(struct Queue\_Node));

    if (!newNode) {

        printf("Memory allocation error!\n");

        exit(EXIT\_FAILURE);

    }

    newNode->data = record;

    newNode->next = NULL;

    if (q->rear == NULL) {

        q->front = q->rear = newNode;

    } else {

        q->rear->next = newNode;

        q->rear = newNode;

    }

}

struct Queue\_WarehouseRecord Queue\_dequeue(struct Queue \*q) {

    if (q->front == NULL) {

        printf("Queue is empty!\n");

        exit(EXIT\_FAILURE);

    }

    struct Queue\_Node \*temp = q->front;

    struct Queue\_WarehouseRecord record = temp->data;

    q->front = q->front->next;

    if (q->front == NULL) {

        q->rear = NULL;

    }

    free(temp);

    return record;

}

void Queue\_displayQueue(struct Queue \*q) {

    if (q->front == NULL) {

        printf("Queue is empty!\n");

        return;

    }

    struct Queue\_Node \*current = q->front;

    printf("Warehouse Records:\n");

    while (current != NULL) {

        printf("Owner: %s %s, Product Name: %s, Manufacturer: %s, Contract Date: %d-%02d-%02d\n",

               current->data.owner\_name, current->data.owner\_surname, current->data.product\_name,

               current->data.manufacturer, current->data.contract\_date\_year, current->data.contract\_date\_month,

               current->data.contract\_date\_day);

        current = current->next;

    }

}

void Queue\_insertRecord(struct Queue \*q) {

    struct Queue\_WarehouseRecord record;

    printf("Enter owner's name: ");

    scanf("%s", record.owner\_name);

    printf("Enter owner's surname: ");

    scanf("%s", record.owner\_surname);

    printf("Enter product name: ");

    scanf("%s", record.product\_name);

    printf("Enter manufacturer: ");

    scanf("%s", record.manufacturer);

    printf("Enter contract conclusion date (day month year): ");

    scanf("%d %d %d", &record.contract\_date\_day, &record.contract\_date\_month, &record.contract\_date\_year);

    printf("Enter wholesale price: ");

    scanf("%f", &record.wholesale\_price);

    printf("Enter unit price: ");

    scanf("%f", &record.unit\_price);

    printf("Enter quantity (units): ");

    scanf("%d", &record.quantity);

    Queue\_enqueue(q, record);

}

int Queue\_searchRecordByValue(struct Queue \*q, struct Queue\_WarehouseRecord record) {

    struct Queue\_Node \*current = q->front;

    int position = 0;

    while (current != NULL) {

        if (strcmp(current->data.owner\_name, record.owner\_name) == 0 &&

            strcmp(current->data.owner\_surname, record.owner\_surname) == 0 &&

            strcmp(current->data.product\_name, record.product\_name) == 0 &&

            strcmp(current->data.manufacturer, record.manufacturer) == 0 &&

            current->data.contract\_date\_day == record.contract\_date\_day &&

            current->data.contract\_date\_month == record.contract\_date\_month &&

            current->data.contract\_date\_year == record.contract\_date\_year &&

            current->data.wholesale\_price == record.wholesale\_price &&

            current->data.unit\_price == record.unit\_price &&

            current->data.quantity == record.quantity) {

            return position;

        }

        position++;

        current = current->next;

    }

    return -1;

}

int Queue\_searchRecordByPosition(struct Queue \*q, int position) {

    struct Queue\_Node \*current = q->front;

    int pos = 0;

    while (current != NULL) {

        if (pos == position) {

            return pos;

        }

        pos++;

        current = current->next;

    }

    return -1;

}

void Queue\_deleteRecord(struct Queue \*q, struct Queue\_WarehouseRecord record) {

    int position = Queue\_searchRecordByValue(q, record);

    if (position == -1) {

        printf("Record not found!\n");

        return;

    }

    struct Queue\_Node \*current = q->front;

    struct Queue\_Node \*prev = NULL;

    int pos = 0;

    while (current != NULL) {

        if (pos == position) {

            if (prev == NULL) {

                q->front = current->next;

            } else {

                prev->next = current->next;

            }

            if (q->rear == current) {

                q->rear = prev;

            }

            free(current);

            printf("Record deleted successfully!\n");

            return;

        }

        prev = current;

        current = current->next;

        pos++;

    }

}

void Queue\_registerQueueToFile(struct Queue \*q, char \*filename) {

    FILE \*file = fopen(filename, "w");

    if (file == NULL) {

        printf("Error opening file!\n");

        return;

    }

    struct Queue\_Node \*current = q->front;

    while (current != NULL) {

        fprintf(file, "Owner: %s %s, Product Name: %s, Manufacturer: %s, Contract Date: %d-%02d-%02d, Wholesale Price: %.2f, Unit Price: %.2f, Quantity: %d\n",

                current->data.owner\_name, current->data.owner\_surname, current->data.product\_name,

                current->data.manufacturer, current->data.contract\_date\_year, current->data.contract\_date\_month,

                current->data.contract\_date\_day, current->data.wholesale\_price, current->data.unit\_price,

                current->data.quantity);

        current = current->next;

    }

    fclose(file);

    printf("Queue registered to file successfully!\n");

}

char\* Queue\_readFilenameFromKeyboard() {

    char \*filename = (char\*)malloc(sizeof(char) \* 100);

    if (!filename) {

        printf("Memory allocation error!\n");

        exit(EXIT\_FAILURE);

    }

    printf("Enter filename: ");

    scanf("%s", filename);

    return filename;

}

void Queue\_queueProgram() {

    struct Queue \*warehouseQueue = Queue\_createQueue();

    char \*filename;

    while (1) {

        int choice;

        printf("\n\n1. Insert Record\n");

        printf("2. Search Record\n");

        printf("3. Delete Record\n");

        printf("4. Display Warehouse Records\n");

        printf("5. Register Queue to File\n");

        printf("6. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                Queue\_insertRecord(warehouseQueue);

                break;

            case 2:

                printf("Not implemented yet!\n");

                break;

            case 3:

                printf("Not implemented yet!\n");

                break;

            case 4:

                Queue\_displayQueue(warehouseQueue);

                break;

            case 5:

                filename = Queue\_readFilenameFromKeyboard();

                Queue\_registerQueueToFile(warehouseQueue, filename);

                free(filename);

                break;

            case 6:

                printf("Exiting...\n");

                exit(EXIT\_SUCCESS);

            default:

                printf("Invalid choice!\n");

        }

    }

}

Main program, unifying both versions:

#include <stdio.h>

#include "lab6a\_1.h"

#include "lab6a\_2.h"

int main() {

    int choice;

    // Display menu

    printf("Choose the version of program to run:\n");

    printf("1. stackProgram 1\n");

    printf("2. stackProgram 2\n");

    printf("Enter your choice: ");

    scanf("%d", &choice);

    // Execute the selected program based on the choice

    switch(choice) {

        case 1:

            stackProgram();

            break;

        case 2:

            Queue\_queueProgram();

            break;

        default:

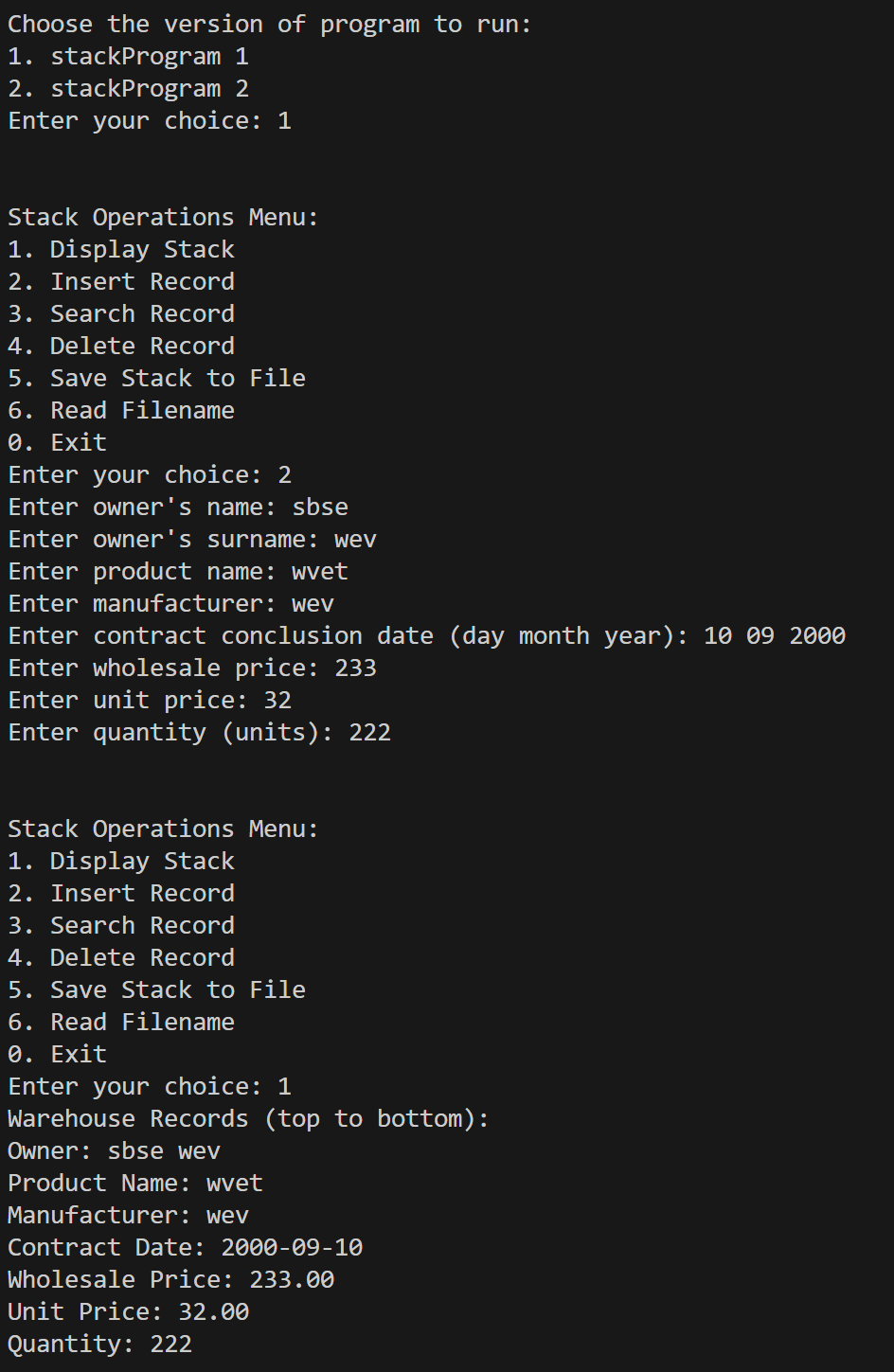
            printf("Invalid choice.\n");

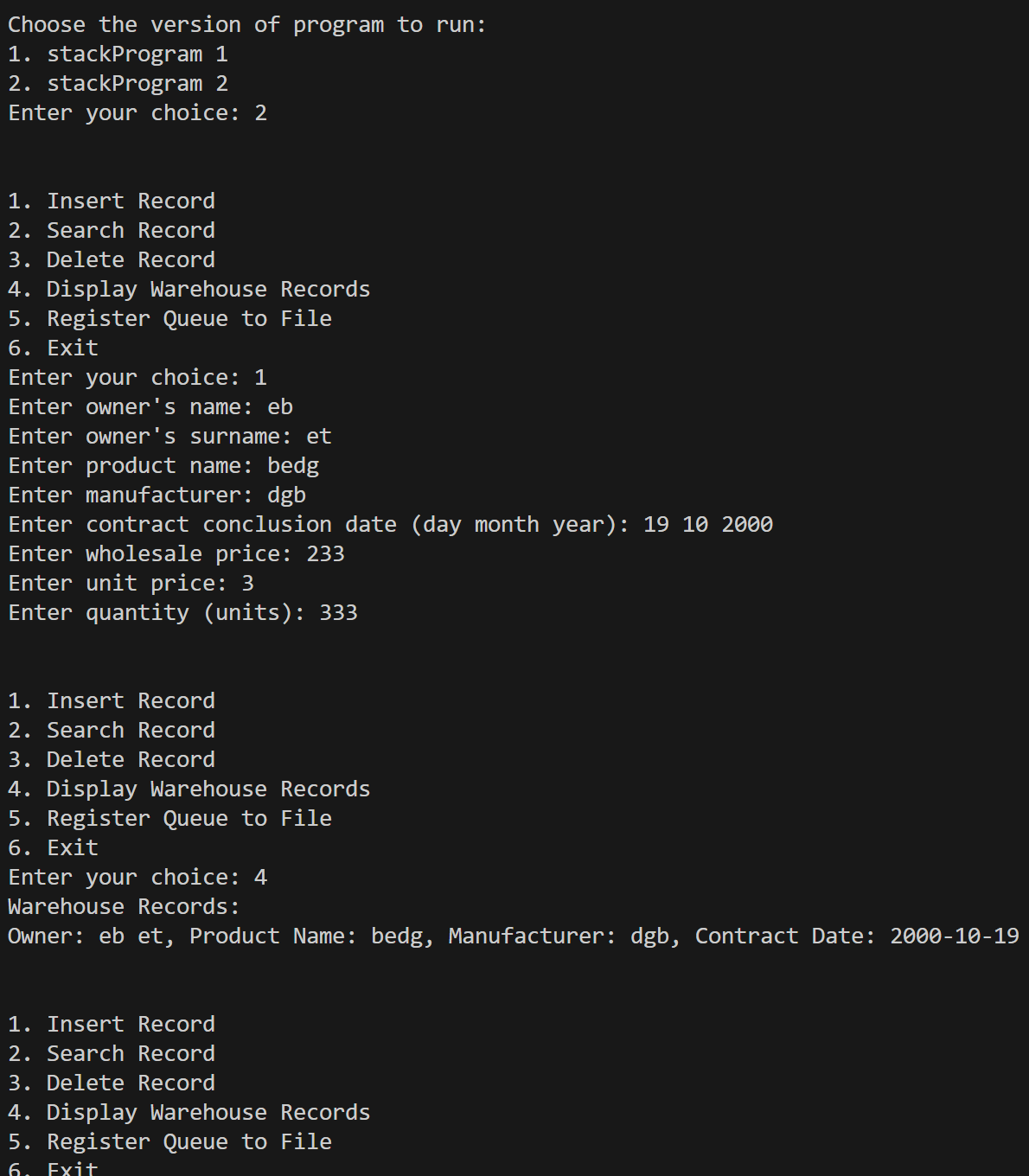
    }

    return 0;

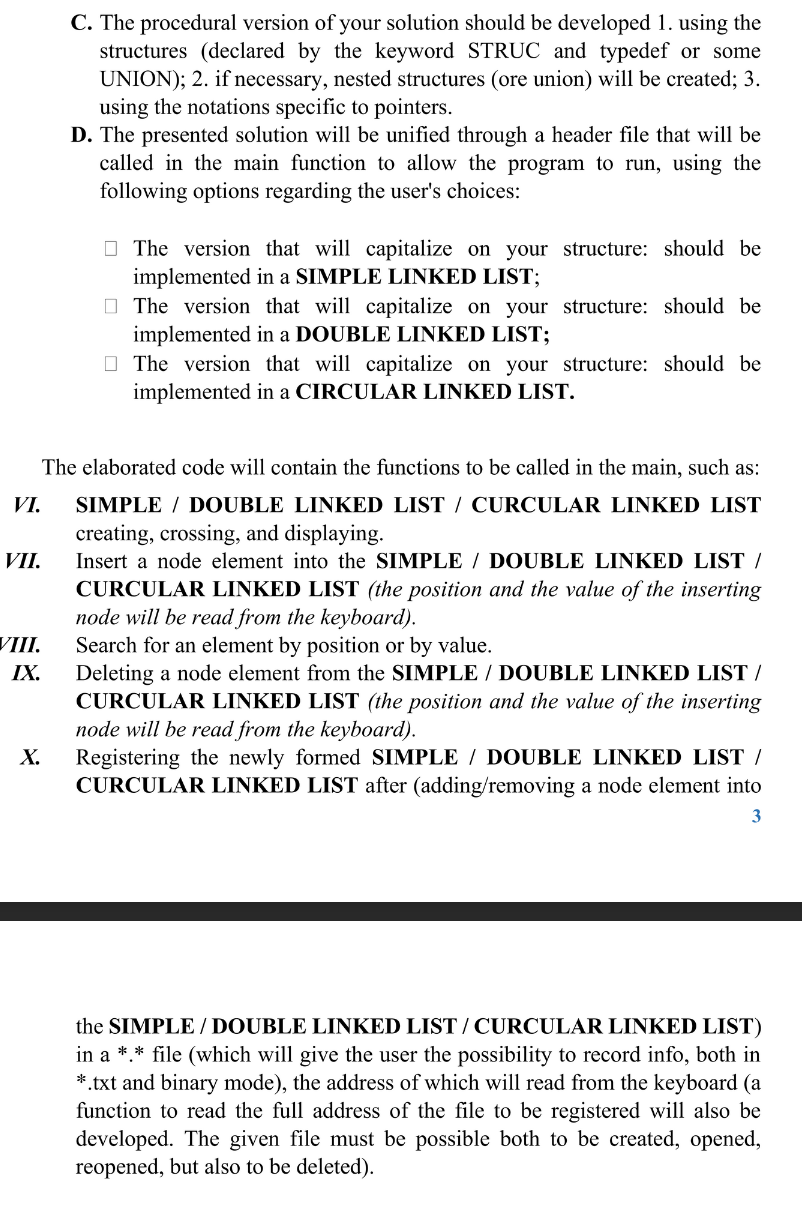
}

**Ouput:**



****

**Task 2**



**1. The code of the program, with relevant comments in it, and the Block diagram; Code:**

Part 1:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

struct SimpleWarehouseRecord {

    char owner\_name[50];

    char owner\_surname[50];

    char product\_name[50];

    char manufacturer[50];

    int contract\_date\_day;

    int contract\_date\_month;

    int contract\_date\_year;

    float wholesale\_price;

    float unit\_price;

    int quantity;

    int remaining\_years;

    int remaining\_months;

    int remaining\_days;

};

// Node structure for the linked list

struct SimpleNode {

    struct SimpleWarehouseRecord data;

    struct SimpleNode \*next;

};

// Function prototypes

struct SimpleNode \*createSimpleNode(struct SimpleWarehouseRecord data);

void insertSimpleNode(struct SimpleNode \*\*head, int position, struct SimpleWarehouseRecord data);

void deleteSimpleNode(struct SimpleNode \*\*head, int position);

void displaySimpleList(struct SimpleNode \*head);

void destroySimpleList(struct SimpleNode \*\*head);

void saveSimpleListToFile(struct SimpleNode \*head, const char \*filename);

void readSimpleListFromFile(struct SimpleNode \*\*head, const char \*filename);

char \*readSimpleFilename();

void simpleLinkedListProgram() {

    struct SimpleNode \*head = NULL;

    int num\_records;

    printf("Enter the number of records to enter: ");

    scanf("%d", &num\_records);

    // Input warehouse registry data and create the linked list

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

        struct SimpleWarehouseRecord record;

        printf("Record %d:\n", i + 1);

        printf("Enter owner's name: ");

        scanf("%s", record.owner\_name);

        printf("Enter owner's surname: ");

        scanf("%s", record.owner\_surname);

        printf("Enter product name: ");

        scanf("%s", record.product\_name);

        printf("Enter manufacturer: ");

        scanf("%s", record.manufacturer);

        printf("Enter contract conclusion date (day month year): ");

        scanf("%d %d %d", &record.contract\_date\_day, &record.contract\_date\_month, &record.contract\_date\_year);

        printf("Enter wholesale price: ");

        scanf("%f", &record.wholesale\_price);

        printf("Enter unit price: ");

        scanf("%f", &record.unit\_price);

        printf("Enter quantity (units): ");

        scanf("%d", &record.quantity);

        insertSimpleNode(&head, i, record);

    }

    int choice;

    do {

        printf("\nSimple Warehouse Management System Menu:\n");

        printf("1. Display Warehouse Records\n");

        printf("2. Save Warehouse Records to File\n");

        printf("3. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                printf("\nWarehouse Records:\n");

                displaySimpleList(head);

                break;

            case 2: {

                char \*filename = readSimpleFilename();

                saveSimpleListToFile(head, filename);

                printf("Warehouse records saved to file '%s'.\n", filename);

                free(filename);

                break;

            }

            case 3:

                printf("Exiting...\n");

                break;

            default:

                printf("Invalid choice. Please enter a number between 1 and 3.\n");

        }

    } while (choice != 3);

    // Destroy the linked list

    destroySimpleList(&head);

}

// Function to create a new node

struct SimpleNode \*createSimpleNode(struct SimpleWarehouseRecord data) {

    struct SimpleNode \*newNode = (struct SimpleNode \*)malloc(sizeof(struct SimpleNode));

    if (newNode == NULL) {

        printf("Memory allocation failed.\n");

        exit(EXIT\_FAILURE);

    }

    newNode->data = data;

    newNode->next = NULL;

    return newNode;

}

// Function to insert a node at a specified position

void insertSimpleNode(struct SimpleNode \*\*head, int position, struct SimpleWarehouseRecord data) {

    if (position < 0) {

        printf("Invalid position.\n");

        return;

    }

    struct SimpleNode \*newNode = createSimpleNode(data);

    if (position == 0) {

        newNode->next = \*head;

        \*head = newNode;

    } else {

        struct SimpleNode \*temp = \*head;

        for (int i = 0; i < position - 1 && temp != NULL; i++) {

            temp = temp->next;

        }

        if (temp == NULL) {

            printf("Position out of range.\n");

            return;

        }

        newNode->next = temp->next;

        temp->next = newNode;

    }

}

// Function to delete a node at a specified position

void deleteSimpleNode(struct SimpleNode \*\*head, int position) {

    if (\*head == NULL) {

        printf("List is empty.\n");

        return;

    }

    struct SimpleNode \*temp = \*head;

    if (position == 0) {

        \*head = temp->next;

        free(temp);

    } else {

        struct SimpleNode \*prev = NULL;

        for (int i = 0; i < position && temp != NULL; i++) {

            prev = temp;

            temp = temp->next;

        }

        if (temp == NULL) {

            printf("Position out of range.\n");

            return;

        }

        prev->next = temp->next;

        free(temp);

    }

}

// Function to display the linked list

void displaySimpleList(struct SimpleNode \*head) {

    struct SimpleNode \*current = head;

    while (current != NULL) {

        printf("Owner: %s %s\n", current->data.owner\_name, current->data.owner\_surname);

        printf("Product Name: %s\n", current->data.product\_name);

        printf("Manufacturer: %s\n", current->data.manufacturer);

        printf("Contract Date: %d-%02d-%02d\n", current->data.contract\_date\_year, current->data.contract\_date\_month, current->data.contract\_date\_day);

        printf("Wholesale Price: %.2f\n", current->data.wholesale\_price);

        printf("Unit Price: %.2f\n", current->data.unit\_price);

        printf("Quantity: %d\n\n", current->data.quantity);

        current = current->next;

    }

}

// Function to destroy the linked list and free memory

void destroySimpleList(struct SimpleNode \*\*head) {

    while (\*head != NULL) {

        struct SimpleNode \*temp = \*head;

        \*head = (\*head)->next;

        free(temp);

    }

}

// Function to save the linked list to a file

void saveSimpleListToFile(struct SimpleNode \*head, const char \*filename) {

    FILE \*file = fopen(filename, "w");

    if (file == NULL) {

        printf("Error opening file for writing.\n");

        return;

    }

    struct SimpleNode \*current = head;

    while (current != NULL) {

        fprintf(file, "Owner: %s %s\n", current->data.owner\_name, current->data.owner\_surname);

        fprintf(file, "Product Name: %s\n", current->data.product\_name);

        fprintf(file, "Manufacturer: %s\n", current->data.manufacturer);

        fprintf(file, "Contract Date: %d-%02d-%02d\n", current->data.contract\_date\_year, current->data.contract\_date\_month, current->data.contract\_date\_day);

        fprintf(file, "Wholesale Price: %.2f\n", current->data.wholesale\_price);

        fprintf(file, "Unit Price: %.2f\n", current->data.unit\_price);

        fprintf(file, "Quantity: %d\n\n", current->data.quantity);

        current = current->next;

    }

    fclose(file);

}

// Function to read a filename from the user

char \*readSimpleFilename() {

    char \*filename = (char \*)malloc(256 \* sizeof(char));

    if (filename == NULL) {

        printf("Memory allocation failed.\n");

        exit(EXIT\_FAILURE);

    }

    printf("Enter filename: ");

    scanf("%s", filename);

    return filename;

}

Part 2:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

struct DoubleWarehouseRecord {

    char owner\_name[50];

    char owner\_surname[50];

    char product\_name[50];

    char manufacturer[50];

    int contract\_date\_day;

    int contract\_date\_month;

    int contract\_date\_year;

    float wholesale\_price;

    float unit\_price;

    int quantity;

    int remaining\_years;

    int remaining\_months;

    int remaining\_days;

};

// Node structure for the double linked list

struct DoubleNode {

    struct DoubleWarehouseRecord data;

    struct DoubleNode \*prev;

    struct DoubleNode \*next;

};

// Function prototypes

struct DoubleNode \*createDoubleNode(struct DoubleWarehouseRecord data);

void insertDoubleNode(struct DoubleNode \*\*head, int position, struct DoubleWarehouseRecord data);

void deleteDoubleNode(struct DoubleNode \*\*head, int position);

void displayDoubleList(struct DoubleNode \*head);

void destroyDoubleList(struct DoubleNode \*\*head);

void saveDoubleListToFile(struct DoubleNode \*head, const char \*filename);

void readDoubleListFromFile(struct DoubleNode \*\*head, const char \*filename);

char \*readDoubleFilename();

void doubleLinkedListProgram() {

    struct DoubleNode \*head = NULL;

    int num\_records;

    printf("Enter the number of records to enter: ");

    scanf("%d", &num\_records);

    // Input warehouse registry data and create the double linked list

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

        struct DoubleWarehouseRecord record;

        printf("Record %d:\n", i + 1);

        printf("Enter owner's name: ");

        scanf("%s", record.owner\_name);

        printf("Enter owner's surname: ");

        scanf("%s", record.owner\_surname);

        printf("Enter product name: ");

        scanf("%s", record.product\_name);

        printf("Enter manufacturer: ");

        scanf("%s", record.manufacturer);

        printf("Enter contract conclusion date (day month year): ");

        scanf("%d %d %d", &record.contract\_date\_day, &record.contract\_date\_month, &record.contract\_date\_year);

        printf("Enter wholesale price: ");

        scanf("%f", &record.wholesale\_price);

        printf("Enter unit price: ");

        scanf("%f", &record.unit\_price);

        printf("Enter quantity (units): ");

        scanf("%d", &record.quantity);

        insertDoubleNode(&head, i, record);

    }

    int choice;

    do {

        printf("\nDouble Warehouse Management System Menu:\n");

        printf("1. Display Warehouse Records\n");

        printf("2. Save Warehouse Records to File\n");

        printf("3. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                printf("\nWarehouse Records:\n");

                displayDoubleList(head);

                break;

            case 2: {

                char \*filename = readDoubleFilename();

                saveDoubleListToFile(head, filename);

                printf("Warehouse records saved to file '%s'.\n", filename);

                free(filename);

                break;

            }

            case 3:

                printf("Exiting...\n");

                break;

            default:

                printf("Invalid choice. Please enter a number between 1 and 3.\n");

        }

    } while (choice != 3);

    // Destroy the double linked list

    destroyDoubleList(&head);

}

// Function to create a new node

struct DoubleNode \*createDoubleNode(struct DoubleWarehouseRecord data) {

    struct DoubleNode \*newNode = (struct DoubleNode \*)malloc(sizeof(struct DoubleNode));

    if (newNode == NULL) {

        printf("Memory allocation failed.\n");

        exit(EXIT\_FAILURE);

    }

    newNode->data = data;

    newNode->prev = NULL;

    newNode->next = NULL;

    return newNode;

}

// Function to insert a node at a specified position

void insertDoubleNode(struct DoubleNode \*\*head, int position, struct DoubleWarehouseRecord data) {

    if (position < 0) {

        printf("Invalid position.\n");

        return;

    }

    struct DoubleNode \*newNode = createDoubleNode(data);

    if (\*head == NULL) {

        \*head = newNode;

        return;

    }

    if (position == 0) {

        newNode->next = \*head;

        (\*head)->prev = newNode;

        \*head = newNode;

        return;

    }

    struct DoubleNode \*current = \*head;

    for (int i = 0; i < position - 1 && current->next != NULL; i++) {

        current = current->next;

    }

    newNode->next = current->next;

    if (current->next != NULL) {

        current->next->prev = newNode;

    }

    current->next = newNode;

    newNode->prev = current;

}

// Function to delete a node at a specified position

void deleteDoubleNode(struct DoubleNode \*\*head, int position) {

    if (\*head == NULL) {

        printf("List is empty.\n");

        return;

    }

    if (position == 0) {

        struct DoubleNode \*temp = \*head;

        \*head = (\*head)->next;

        if (\*head != NULL) {

            (\*head)->prev = NULL;

        }

        free(temp);

        return;

    }

    struct DoubleNode \*current = \*head;

    for (int i = 0; i < position && current != NULL; i++) {

        current = current->next;

    }

    if (current == NULL) {

        printf("Position out of range.\n");

        return;

    }

    current->prev->next = current->next;

    if (current->next != NULL) {

        current->next->prev = current->prev;

    }

    free(current);

}

// Function to display the double linked list

void displayDoubleList(struct DoubleNode \*head) {

    struct DoubleNode \*current = head;

    while (current != NULL) {

        printf("Owner: %s %s\n", current->data.owner\_name, current->data.owner\_surname);

        printf("Product Name: %s\n", current->data.product\_name);

        printf("Manufacturer: %s\n", current->data.manufacturer);

        printf("Contract Date: %d-%02d-%02d\n", current->data.contract\_date\_year, current->data.contract\_date\_month, current->data.contract\_date\_day);

        printf("Wholesale Price: %.2f\n", current->data.wholesale\_price);

        printf("Unit Price: %.2f\n", current->data.unit\_price);

        printf("Quantity: %d\n\n", current->data.quantity);

        current = current->next;

    }

}

// Function to destroy the double linked list and free memory

void destroyDoubleList(struct DoubleNode \*\*head) {

    while (\*head != NULL) {

        struct DoubleNode \*temp = \*head;

        \*head = (\*head)->next;

        free(temp);

    }

}

// Function to save the double linked list to a file

void saveDoubleListToFile(struct DoubleNode \*head, const char \*filename) {

    FILE \*file = fopen(filename, "w");

    if (file == NULL) {

        printf("Error opening file for writing.\n");

        return;

    }

    struct DoubleNode \*current = head;

    while (current != NULL) {

        fprintf(file, "Owner: %s %s\n", current->data.owner\_name, current->data.owner\_surname);

        fprintf(file, "Product Name: %s\n", current->data.product\_name);

        fprintf(file, "Manufacturer: %s\n", current->data.manufacturer);

        fprintf(file, "Contract Date: %d-%02d-%02d\n", current->data.contract\_date\_year, current->data.contract\_date\_month, current->data.contract\_date\_day);

        fprintf(file, "Wholesale Price: %.2f\n", current->data.wholesale\_price);

        fprintf(file, "Unit Price: %.2f\n", current->data.unit\_price);

        fprintf(file, "Quantity: %d\n\n", current->data.quantity);

        current = current->next;

    }

    fclose(file);

}

// Function to read a filename from the user

char \*readDoubleFilename() {

    char \*filename = (char \*)malloc(256 \* sizeof(char));

    if (filename == NULL) {

        printf("Memory allocation failed.\n");

        exit(EXIT\_FAILURE);

    }

    printf("Enter filename: ");

    scanf("%s", filename);

    return filename;

}

Part 3:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

struct CircularWarehouseRecord {

    char owner\_name[50];

    char owner\_surname[50];

    char product\_name[50];

    char manufacturer[50];

    int contract\_date\_day;

    int contract\_date\_month;

    int contract\_date\_year;

    float wholesale\_price;

    float unit\_price;

    int quantity;

    int remaining\_years;

    int remaining\_months;

    int remaining\_days;

};

// Node structure for the circular linked list

struct CircularNode {

    struct CircularWarehouseRecord data;

    struct CircularNode \*next;

};

// Function prototypes

struct CircularNode \*createCircularNode(struct CircularWarehouseRecord data);

void insertCircularNode(struct CircularNode \*\*head, int position, struct CircularWarehouseRecord data);

void deleteCircularNode(struct CircularNode \*\*head, int position);

void displayCircularList(struct CircularNode \*head);

void destroyCircularList(struct CircularNode \*\*head);

void saveCircularListToFile(struct CircularNode \*head, const char \*filename);

void readCircularListFromFile(struct CircularNode \*\*head, const char \*filename);

char \*readCircularFilename();

void circularLinkedListProgram() {

    struct CircularNode \*head = NULL;

    int num\_records;

    printf("Enter the number of records to enter: ");

    scanf("%d", &num\_records);

    // Input warehouse registry data and create the circular linked list

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

        struct CircularWarehouseRecord record;

        printf("Record %d:\n", i + 1);

        printf("Enter owner's name: ");

        scanf("%s", record.owner\_name);

        printf("Enter owner's surname: ");

        scanf("%s", record.owner\_surname);

        printf("Enter product name: ");

        scanf("%s", record.product\_name);

        printf("Enter manufacturer: ");

        scanf("%s", record.manufacturer);

        printf("Enter contract conclusion date (day month year): ");

        scanf("%d %d %d", &record.contract\_date\_day, &record.contract\_date\_month, &record.contract\_date\_year);

        printf("Enter wholesale price: ");

        scanf("%f", &record.wholesale\_price);

        printf("Enter unit price: ");

        scanf("%f", &record.unit\_price);

        printf("Enter quantity (units): ");

        scanf("%d", &record.quantity);

        insertCircularNode(&head, i, record);

    }

    int choice;

    do {

        printf("\nCircular Warehouse Management System Menu:\n");

        printf("1. Display Warehouse Records\n");

        printf("2. Save Warehouse Records to File\n");

        printf("3. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                printf("\nWarehouse Records:\n");

                displayCircularList(head);

                break;

            case 2: {

                char \*filename = readCircularFilename();

                saveCircularListToFile(head, filename);

                printf("Warehouse records saved to file '%s'.\n", filename);

                free(filename);

                break;

            }

            case 3:

                printf("Exiting...\n");

                break;

            default:

                printf("Invalid choice. Please enter a number between 1 and 3.\n");

        }

    } while (choice != 3);

    // Destroy the circular linked list

    destroyCircularList(&head);

}

// Function to create a new node

struct CircularNode \*createCircularNode(struct CircularWarehouseRecord data) {

    struct CircularNode \*newNode = (struct CircularNode \*)malloc(sizeof(struct CircularNode));

    if (newNode == NULL) {

        printf("Memory allocation failed.\n");

        exit(EXIT\_FAILURE);

    }

    newNode->data = data;

    newNode->next = NULL;

    return newNode;

}

// Function to insert a node at a specified position

void insertCircularNode(struct CircularNode \*\*head, int position, struct CircularWarehouseRecord data) {

    if (position < 0) {

        printf("Invalid position.\n");

        return;

    }

    struct CircularNode \*newNode = createCircularNode(data);

    if (\*head == NULL) {

        \*head = newNode;

        (\*head)->next = \*head; // Circular link

        return;

    }

    struct CircularNode \*current = \*head;

    int count = 0;

    while (count < position - 1 && current->next != \*head) {

        current = current->next;

        count++;

    }

    newNode->next = current->next;

    current->next = newNode;

    if (position == 0) {

        \*head = newNode;

    }

    // Update the last node to point to the head for circularity

    struct CircularNode \*last = \*head;

    while (last->next != \*head) {

        last = last->next;

    }

    last->next = \*head;

}

// Function to delete a node at a specified position

void deleteCircularNode(struct CircularNode \*\*head, int position) {

    if (\*head == NULL) {

        printf("List is empty.\n");

        return;

    }

    struct CircularNode \*temp = \*head;

    struct CircularNode \*prev = NULL;

    int count = 0;

    // Find the node to be deleted

    while (temp->next != \*head && count != position) {

        prev = temp;

        temp = temp->next;

        count++;

    }

    // If the first node is to be deleted

    if (position == 0) {

        struct CircularNode \*last = \*head;

        while (last->next != \*head) {

            last = last->next;

        }

        last->next = temp->next;

        \*head = temp->next;

    }

    // If the last node is to be deleted

    else if (temp->next == \*head) {

        prev->next = \*head;

    }

    // If a middle node is to be deleted

    else {

        prev->next = temp->next;

    }

    free(temp);

}

// Function to display the circular linked list

void displayCircularList(struct CircularNode \*head) {

    if (head == NULL) {

        printf("List is empty.\n");

        return;

    }

    struct CircularNode \*current = head;

    do {

        printf("Owner: %s %s\n", current->data.owner\_name, current->data.owner\_surname);

        printf("Product Name: %s\n", current->data.product\_name);

        printf("Manufacturer: %s\n", current->data.manufacturer);

        printf("Contract Date: %d-%02d-%02d\n", current->data.contract\_date\_year, current->data.contract\_date\_month, current->data.contract\_date\_day);

        printf("Wholesale Price: %.2f\n", current->data.wholesale\_price);

        printf("Unit Price: %.2f\n", current->data.unit\_price);

        printf("Quantity: %d\n\n", current->data.quantity);

        current = current->next;

    } while (current != head);

}

// Function to destroy the circular linked list and free memory

void destroyCircularList(struct CircularNode \*\*head) {

    if (\*head == NULL) {

        return;

    }

    struct CircularNode \*current = \*head;

    struct CircularNode \*temp;

    do {

        temp = current->next;

        free(current);

        current = temp;

    } while (current != \*head);

    \*head = NULL;

}

// Function to save the circular linked list to a file

void saveCircularListToFile(struct CircularNode \*head, const char \*filename) {

    FILE \*file = fopen(filename, "w");

    if (file == NULL) {

        printf("Error opening file for writing.\n");

        return;

    }

    struct CircularNode \*current = head;

    do {

        fprintf(file, "Owner: %s %s\n", current->data.owner\_name, current->data.owner\_surname);

        fprintf(file, "Product Name: %s\n", current->data.product\_name);

        fprintf(file, "Manufacturer: %s\n", current->data.manufacturer);

        fprintf(file, "Contract Date: %d-%02d-%02d\n", current->data.contract\_date\_year, current->data.contract\_date\_month, current->data.contract\_date\_day);

        fprintf(file, "Wholesale Price: %.2f\n", current->data.wholesale\_price);

        fprintf(file, "Unit Price: %.2f\n", current->data.unit\_price);

        fprintf(file, "Quantity: %d\n\n", current->data.quantity);

        current = current->next;

    } while (current != head);

    fclose(file);

}

// Function to read a filename from the user

char \*readCircularFilename() {

    char \*filename = (char \*)malloc(256 \* sizeof(char));

    if (filename == NULL) {

        printf("Memory allocation failed.\n");

        exit(EXIT\_FAILURE);

    }

    printf("Enter filename: ");

    scanf("%s", filename);

    return filename;

}

Main program, unifying all 3 versions:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include "lab6b\_1.h"

#include "lab6b\_2.h"

#include "lab6b\_3.h"

int main() {

    int choice;

    printf("Choose the version of program to run:\n");

    printf("1. Simple Linked List\n");

    printf("2. Double Linked List\n");

    printf("3. Circular Linked List\n");

    printf("Enter your choice: ");

    scanf("%d", &choice);

    switch(choice) {

        case 1:

            simpleLinkedListProgram();

            break;

        case 2:

            doubleLinkedListProgram();

            break;

        case 3:

            circularLinkedListProgram();

            break;

        default:

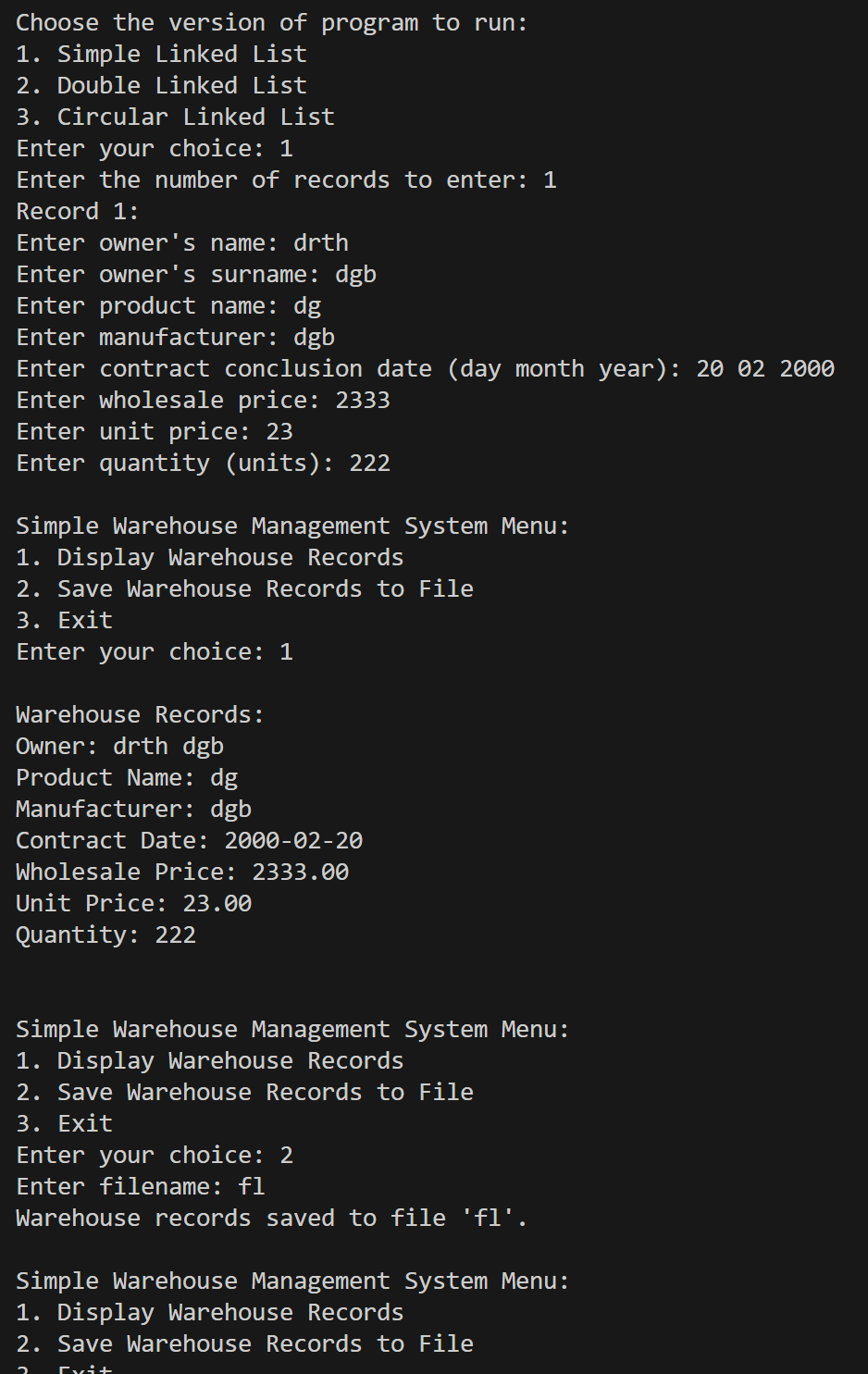
            printf("Invalid choice.\n");

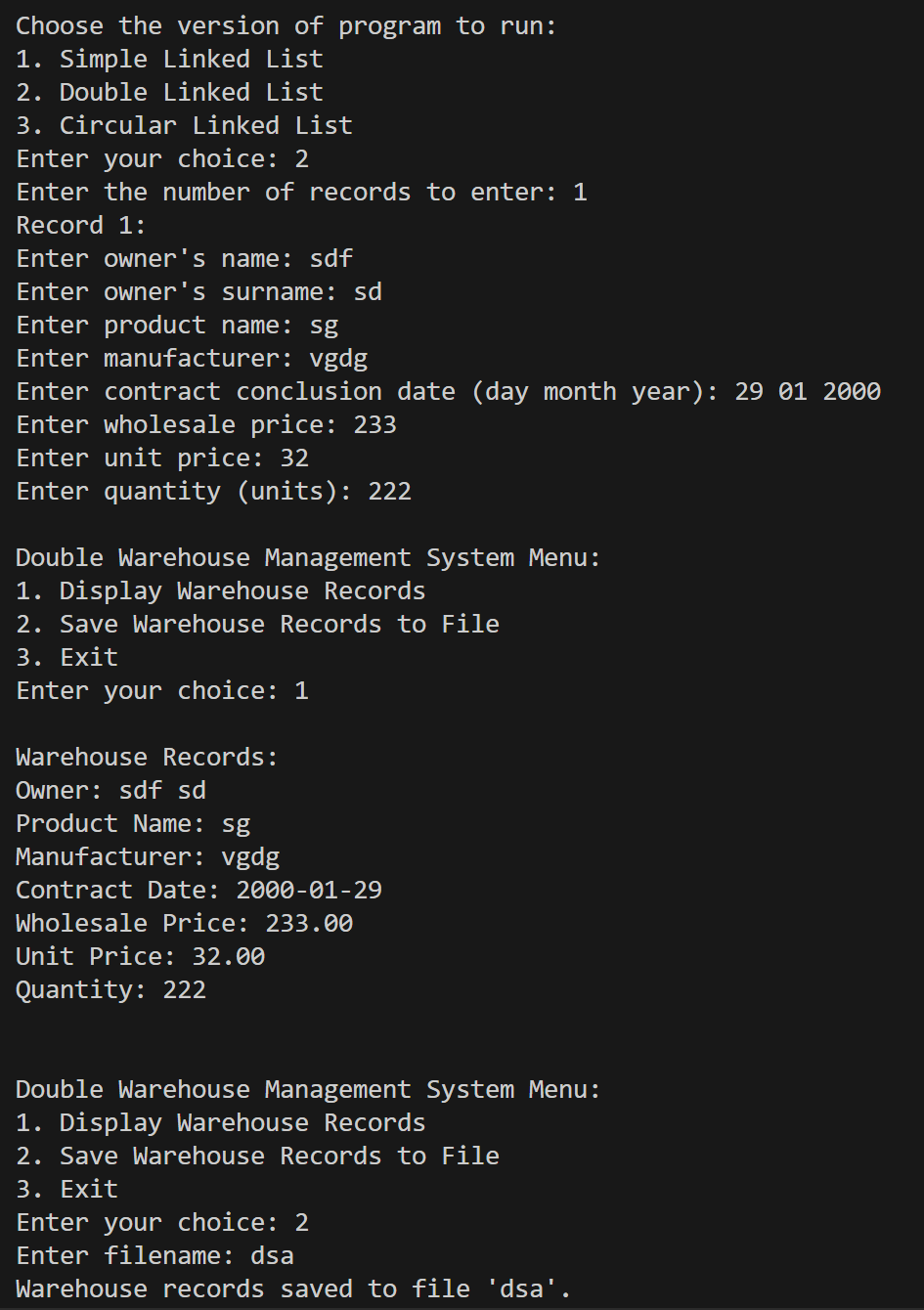
    }

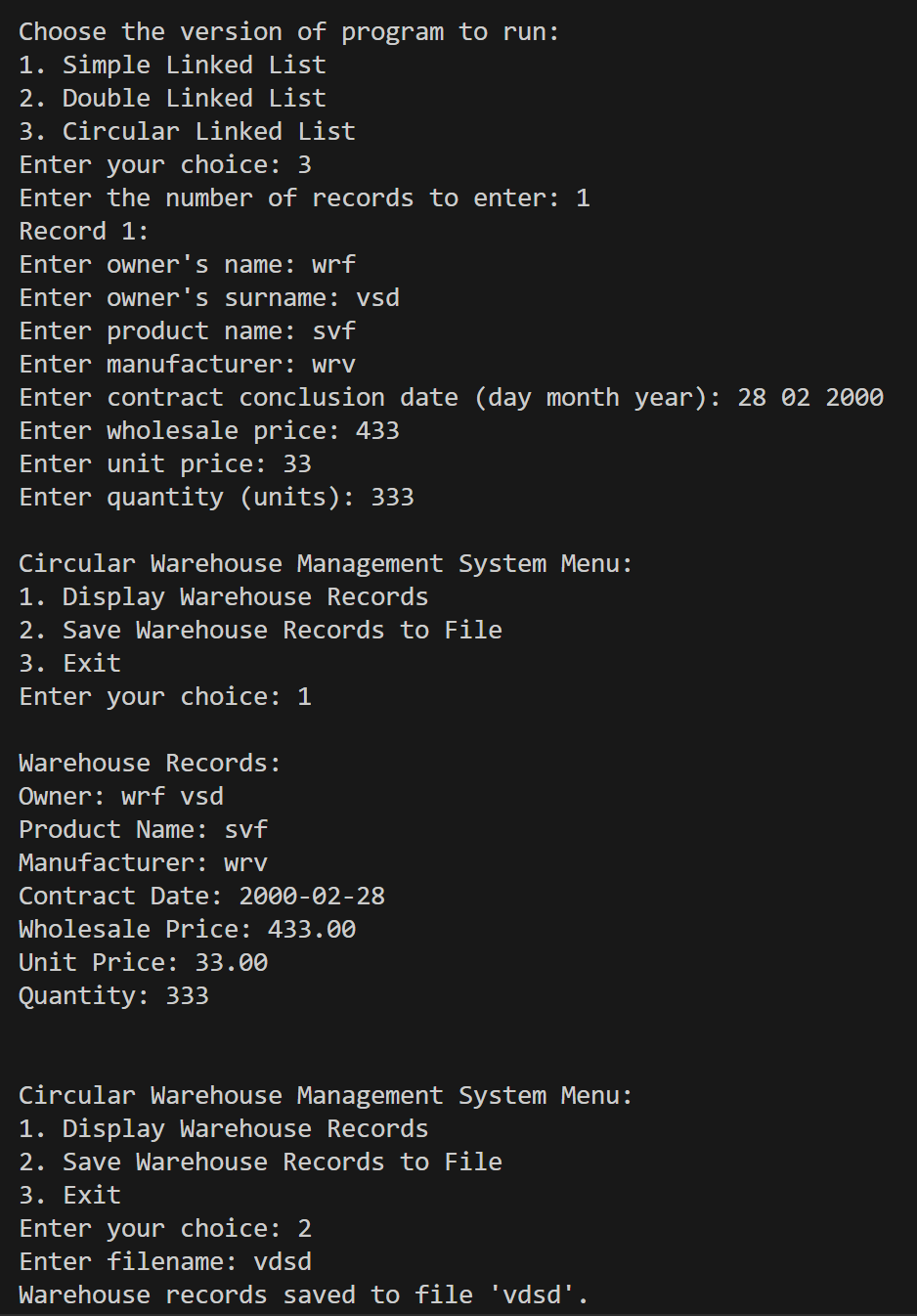
    return 0;

}

**Output:**

****

****



**Conclusion:**

Combining file handling and structures in C facilitates efficient data management by organizing related information into structured units. Structures provide clarity and organization to data, while file handling operations enable reading from and writing to external files. Together, these concepts empower programmers to build robust applications for tasks like data processing and database management. By defining appropriate structures, complex data entities can be modeled, making it easier to work with them in code. The synergy between file handling and structures allows for the creation of efficient, organized, and scalable solutions for a wide range of programming challenges.