Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт

Кафедра «Информатика»
кафедра

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №7

Связные списки
Тема

Преподаватель _____ А. С. Черниговский
подпись, дата — инициалы, фамилия

Студент КИ19-17/1б 031939175 _____ А. Д. Непомнящий
номер группы, зачетной книжки — подпись, дата — инициалы, фамилия

Красноярск 2020

1 Цель

Цель настоящей работы состоит в том, чтобы продолжить ознакомление с синтаксисом языка СИ, стандартом кодирования, принципами работы с связными списками, ознакомиться с отношением многие ко многие.

2 Задачи

Сущности А и Б имеют отношение многие ко многим. Написать программу, моделирующую это отношение:

На оценку 3 балла:

1) написать программу в соответствии с заданием;
2) реализовать возможность добавления элементов А и Б;
3) реализовать возможность создания связи между элементами типа А и Б

На оценку 4 балла:

1) выполнить требования предыдущих пунктов;
2) добавить возможность сохранять и загружать данные из файла (в т.ч. связи);
3) добавить возможность выводить на экран все элементы А или Б (на усмотрение пользователя);
4) добавить возможность выводить на экран все элементы А, связанные с выбранным элементом Б и наоборот;

На оценку 5 баллов:

1) выполнить требования предыдущих пунктов;
2) добавить возможность удаления выбранного элемента типа А или типа Б;
3) добавить сортировку по одному из выбранных полей.

3 Описание задания

Товары и заказы.

А: товар (наименование, цена, масса).

Б: заказ (ФИО заказчика, срок доставки, дата оформления).

Один товар может входить в несколько заказов, один заказ может содержать несколько товаров.

4 Ход выполнения

Ниже представлен листинг программы по заданию.

Листинг 1 – Код программы, решающей задачу, файл main.c

```
#include <stdio.h>
#include <stdbool.h>
#include <malloc.h>

#include "Constants.h"
#include "Input.h"
#include "ListLink.h"
#include "Data.h"

enum OperationsCodes
{
    LOAD_FROM_FILE = 1,
    SAVE_TO_FILE = 2,
    ADD = 3,
    ADD_ORDER = 1,
    ADD_PRODUCT = 2,
    ADD_RELATION = 3,
    ADD_BACK = 4,
    PRINT = 4,
    PRINT_ORDERS = 1,
    PRINT_PRODUCTS = 2,
    PRINT_BY_ORDER = 3,
    PRINT_BY_PRODUCT = 4,
    PRINT_BACK = 5,
    DELETE = 5,
    DELETE_ORDER = 1,
    DELETE_PRODUCT = 2,
    DELETE_BACK = 3,
    SORT_ORDERS = 6,
    SORT_ORDERS_ORD = 1,
    SORT_ORDERS_SHIP = 2,
    SORT_ORDERS_BACK = 3,
    SORT_PRODUCTS = 7,
    SORT_PRODUCTS_PRICE = 1,
    SORT_PRODUCTS_WEIGHT = 2,
    SORT_PRODUCTS_BACK = 3,
    QUIT = 8,
```

Продолжение листинга 1

```
};


bool MainMenuInputChecker(int operationCode)
{
    return operationCode >= LOAD_FROM_FILE && operationCode <= QUIT;
}


bool PositiveIntInputChecker(int intToCheck)
{
    return intToCheck > 0;
}


bool AddMenuInputChecker(int operationCode)
{
    return operationCode >= ADD_ORDER &&
           operationCode <= ADD_BACK;
}


bool PrintMenuInputChecker(int operationCode)
{
    return operationCode >= PRINT_ORDERS && operationCode <= PRINT_BACK;
}


bool DeleteMenuInputChecker(int operationCode)
{
    return operationCode >= DELETE_ORDER && operationCode <= DELETE_BACK;
}


bool SortOrdersMenuInputChecker(int operationCode)
{
    return operationCode >= SORT_ORDERS_ORD &&
           operationCode <= SORT_ORDERS_BACK;
}
```

Продолжение листинга 1

```
bool SortProductsMenuInputChecker(int operationCode)
{
    return operationCode >= SORT_PRODUCTS_PRICE &&
           operationCode <= SORT_PRODUCTS_BACK;
}


bool CompOrdersOrderDate(Order* first, Order* second)
{
    if (first->ordYear == second->ordYear)
    {
        if (first->ordMonth == second->ordMonth)
        {
            return (first->ordDay > second->ordDay);
        }
        return (first->ordMonth > second->ordMonth);
    }
    return first->ordYear > second->ordYear;
}


bool CompOrdersShipDate(Order* first, Order* second)
{
    if (first->shipYear == second->shipYear)
    {
        if (first->shipMonth == second->shipMonth)
        {
            return (first->shipDay > second->shipDay);
        }
        return (first->shipMonth > second->shipMonth);
    }
    return first->shipYear > second->shipYear;
}


bool CompProductsPrice(Product* first, Product* second)
{
    return first->price > second->price;
}
```

Продолжение листинга 1

```
bool CompProductsWeight(Product* first, Product* second)
{
    return first->weight > second->weight;
}


void SaveData(FILE* saveFile, ListLink* orders, ListLink* products,
              ListLink* relations)
{
    ListLink* i;

    int ordersSize;
    ordersSize = ListLinkSize(orders);
    fprintf(saveFile, "%d\n", ordersSize);
    i = ListLinkHead(orders);
    while (i != NULL)
    {
        PrintFileOrder(saveFile, i->content);
        i = i->next;
    }

    int productsSize;
    productsSize = ListLinkSize(products);
    fprintf(saveFile, "%d\n", productsSize);
    i = ListLinkHead(products);
    while (i != NULL)
    {
        PrintFileProduct(saveFile, i->content);
        i = i->next;
    }

    int relationsSize;
    relationsSize = ListLinkSize(relations);
    fprintf(saveFile, "%d\n", relationsSize);
    i = ListLinkHead(relations);
    while (i != NULL)
    {
        PrintFileRelation(saveFile, i->content);
        i = i->next;
    }
}
```

Продолжение листинга 1

```
void LoadData(FILE* saveFile, ListLink** orders, ListLink** products,
              ListLink** relations)
{
    int ordersSize;
    fscanf(saveFile, "%d", &ordersSize);
    for (int i = 0; i < ordersSize; i++)
    {
        char owner[INPUT_SIZE];
        int ordDay;
        int ordMonth;
        int ordYear;
        int shipDay;
        int shipMonth;
        int shipYear;
        fscanf(saveFile, "%s", owner);
        fscanf(saveFile, "%d", &ordMonth);
        fscanf(saveFile, "%d", &ordDay);
        fscanf(saveFile, "%d", &ordYear);
        fscanf(saveFile, "%d", &shipMonth);
        fscanf(saveFile, "%d", &shipDay);
        fscanf(saveFile, "%d", &shipYear);
        *orders = AddOrder(*orders, owner, ordDay, ordMonth, ordYear, shipDay,
                           shipMonth, shipYear);

    }
    int productsSize;
    fscanf(saveFile, "%d", &productsSize);
    for (int i = 0; i < productsSize; i++)
    {
        char name[INPUT_SIZE];
        int price;
        int weight;
        fscanf(saveFile, "%s", name);
        fscanf(saveFile, "%d", &price);
        fscanf(saveFile, "%d", &weight);
        *products = AddProduct(*products, name, price, weight);
    }
    int relationsSize;
    fscanf(saveFile, "%d", &relationsSize);
```

Продолжение листинга 1

```
    for (int i = 0; i < relationsSize; i++)
    {
        char owner[INPUT_SIZE];
        char product[INPUT_SIZE];
        fscanf(saveFile, "%s", product);
        fscanf(saveFile, "%s", owner);
        ListLink* ownerLink;
        ListLink* productLink;
        ownerLink = FindOrder(*orders, owner);
        productLink = FindProduct(*products, product);
        *relations = AddRelation(*relations, ownerLink, productLink);
    }
}


int main()
{
    ListLink* orders;
    ListLink* products;
    ListLink* relations;
    orders = NULL;
    products = NULL;
    relations = NULL;

    int operationCode;
    while (true)
    {
        int subOperationCode;
        printf("\n1. Load data from savefile.\n"
               "2. Save current data to savefile.\n"
               "3. Add new element.\n"
               "4. Print information.\n"
               "5. Delete elements.\n"
               "6. Sort orders.\n"
               "7. Sort products.\n"
               "8. Quit without saving.\n\n");
        operationCode = CycleInputInt(
                "Choose the command and enter its number",
                MainMenuInputChecker);

        if (operationCode == LOAD_FROM_FILE)
```

Продолжение листинга 1

```
        {
            FILE* savefile;
            savefile = fopen("savefile.txt", "r");
            if (savefile == NULL)
            {
                printf("Couldn't open savefile!\n");
            }
            else
            {
                ListLinkFree(orders);
                orders = NULL;
                ListLinkFree(products);
                products = NULL;
                ListLinkFree(relations);
                relations = NULL;
                LoadData(savefile, &orders, &products, &relations);
                printf("Loaded!\n");
            }
            fclose(savefile);
        }

        if (operationCode == SAVE_TO_FILE)
        {

            FILE* savefile;
            savefile = fopen("savefile.txt", "w");
            if (savefile == NULL)
            {
                printf("Couldn't write new data!\n");
            }
            else
            {
                SaveData(savefile, orders, products, relations);
                printf("Saved!\n");
            }
            fclose(savefile);
        }

        if (operationCode == ADD)
        {
            subOperationCode = CycleInputInt("\n1. Add order."
```

Продолжение листинга 1

```
                                        "\n2. Add product."
                                        "\n3. Add relation."
                                        "\n4. Back.\n",
                                        AddMenuInputChecker);
    if (subOperationCode == ADD_ORDER)
    {
        ListLink* addReturn;
        char* owner;
        int ordDay;
        int ordMonth;
        int ordYear;
        int shipDay;
        int shipMonth;
        int shipYear;

        owner = CycleInputString("Enter owner's name");
        ordMonth = CycleInputInt(
                "Enter month (number) of order application",
                PositiveIntInputChecker);
        ordDay = CycleInputInt("Enter day of order application",
                               PositiveIntInputChecker);
        ordYear = CycleInputInt("Enter year of order application",
                                PositiveIntInputChecker);
        shipMonth = CycleInputInt("Enter month (number) of shipment",
                                  PositiveIntInputChecker);
        shipDay = CycleInputInt("Enter day of shipment",
                                PositiveIntInputChecker);
        shipYear = CycleInputInt("Enter year of shipment",
                                 PositiveIntInputChecker);
        addReturn = AddOrder(orders, owner, ordDay, ordMonth, ordYear,
                             shipDay, shipMonth, shipYear);
        if (addReturn != NULL)
        {
            orders = addReturn;
        }
        else
        {
            printf("Order by such person already exists!");
        }
        free(owner);
    }
```

Продолжение листинга 1

```
        if (subOperationCode == ADD_PRODUCT)
        {
            ListLink* addReturn;
            char* name;
            int price;
            int weight;

            name = CycleInputString("Enter product's name");
            price = CycleInputInt("Enter product's price", NULL);
            weight = CycleInputInt("Enter product's weight",
                                   PositiveIntInputChecker);
            addReturn = AddProduct(products, name, price, weight);
            if (addReturn != NULL)
            {
                products = addReturn;
                printf("Added!");
            }
            else
            {
                printf("Product with such name already exists!");
            }
            free(name);
        }
        if (subOperationCode == ADD_RELATION)
        {
            ListLink* addReturn;
            ListLink* order;
            ListLink* product;
            char* orderName;
            char* productName;
            orderName = CycleInputString("Enter owner's name");
            productName = CycleInputString("Enter product's name");
            order = FindOrder(orders, orderName);
            product = FindProduct(products, productName);

            if (order != NULL && product != NULL)
            {
                relations = AddRelation(relations, order, product);
                printf("Added!");
            }
            else
```

Продолжение листинга 1

```
            {
                printf("Such relation already exists, or product"
                       "or order wasn't found!");
            }
            free(orderName);
            free(productName);

        }
    }

    if (operationCode == PRINT)
    {
        subOperationCode = CycleInputInt("\n1. Print all orders."
                                         "\n2. Print all products."
                                         "\n3. Print order's content."
                                         "\n4. Print product's related"
                                         "orders.\n"
                                         "\n5. Back.\n",
                                         PrintMenuInputChecker);
        if (subOperationCode == PRINT_ORDERS)
        {
            PrintAllOrders(orders);
        }
        if (subOperationCode == PRINT_PRODUCTS)
        {
            PrintAllProducts(products);
        }
        if (subOperationCode == PRINT_BY_ORDER)
        {
            char* name;
            fflush(stdout);
            name = CycleInputString("Enter owner's name");
            ListLink* order;
            order = FindOrder(orders, name);
            if (order == NULL)
            {
                printf("No such order!");
            }
            else
            {
                PrintProductsByOrder(order->content, relations);
```

Продолжение листинга 1

```
            }
        }
        if (subOperationCode == PRINT_BY_PRODUCT)
        {
            char* name;
            fflush(stdout);
            name = CycleInputString("Enter product's name");
            ListLink* product;
            product = FindOrder(products, name);
            if (product == NULL)
            {
                printf("No such product!");
            }
            else
            {
                PrintOrdersByProduct(product->content, relations);
            }
        }
    }

    if (operationCode == DELETE)
    {
        printf("\n1. Delete order.\n"
               "2. Delete product.\n"
               "3.Back\n\n");
        subOperationCode = CycleInputInt(
                "Choose the command and enter its number",
                DeleteMenuInputChecker);
        if (subOperationCode == DELETE_ORDER)
        {
            char* name;
            name = CycleInputString("Enter owner's name");
            if (FindOrder(orders, name) != NULL)
            {
                orders = DeleteOrder(FindOrder(orders, name), &relations);
                free(name);
            }
        }
        if (subOperationCode == DELETE_PRODUCT)
        {
            char* name;
```

Продолжение листинга 1

```
            name = CycleInputString("Enter product's name");
            if (FindProduct(products, name) != NULL)
            {
                products = DeleteProduct(FindProduct(products, name),
                                         &relations);
                free(name);
            }
        }
    }

    if (operationCode == SORT_ORDERS)
    {
        subOperationCode = CycleInputInt("\n1. Sort by order date."
                                         "\n2. Sort by shipment date."
                                         "\n3. Back.\n",
                                         SortOrdersMenuInputChecker);
        if (subOperationCode == SORT_ORDERS_ORD)
        {
            ListLinkBubbleSort(orders, CompOrdersOrderDate);
        }

        if (subOperationCode == SORT_ORDERS_SHIP)
        {
            ListLinkBubbleSort(orders, CompOrdersShipDate);
        }
    }

    if (operationCode == SORT_PRODUCTS)
    {
        subOperationCode = CycleInputInt("\n1. Sort by products price."
                                         "\n2. Sort by products weight."
                                         "\n3. Back.\n",
                                         SortProductsMenuInputChecker);
        if (subOperationCode == SORT_PRODUCTS_PRICE)
        {
            ListLinkBubbleSort(products, CompProductsPrice);
        }
        if (subOperationCode == SORT_PRODUCTS_WEIGHT)
        {
            ListLinkBubbleSort(products, CompProductsWeight);
        }
```

Окончание листинга 1

```
        }

        if (operationCode == QUIT)
        {
            break;
        }
    }
    ListLinkFree(orders);
    ListLinkFree(products);
    ListLinkFree(relations);
    return 0;
}
```

Листинг 2 - Код программы, решающей задачу, файл Constants.h

```
#ifndef CONSTANTS_H
#define CONSTANTS_H

enum
{
    INPUT_SIZE = 100
};

#endif //CONSTANTS_H
```

Листинг 3 - Код программы, решающей задачу, файл ListLink.h

```
#ifndef LISTLINK_H
#define LISTLINK_H

#include <stdbool.h>

typedef struct ListLink
{
    struct ListLink* previous;
    struct ListLink* next;
    void* content;
} ListLink;

ListLink* ListLinkHead(ListLink* objectLink);
```

Окончание листинга 3

```
ListLink* ListLinkTail(ListLink* objectLink);

ListLink* ListLinkAdd(ListLink* objectList, void* content);

ListLink* ListLinkDelete(ListLink* linkToDelete);

void ListLinkSwap(ListLink* firstLink, ListLink* secondLink);

int ListLinkSize(ListLink* objectLink);

void
ListLinkBubbleSort(ListLink* objectLink, bool(* comparator)(void*, void*));

void ListLinkFree(ListLink* objectLink);


#endif //LISTLINK_H
```

Листинг 4 - Код программы, решающей задачу, файл ListLink.c

```
#include <malloc.h>
#include <stdlib.h>
#include <stdbool.h>

#include "ListLink.h"

ListLink* ListLinkHead(ListLink* objectLink)
{
    ListLink* position;
    position = objectLink;
    if (objectLink == NULL)
    {
        return NULL;
    }
    while (position->previous != NULL)
    {
        position = position->previous;
    }
    return position;
}
```

Продолжение листинга 4

```
ListLink* ListLinkTail(ListLink* objectLink)
{
    ListLink* position;
    position = objectLink;
    if (position == NULL)
    {
        return NULL;
    }
    while (position->next != NULL)
    {
        position = position->next;
    }
    return position;
}


ListLink* ListLinkAdd(ListLink* objectList, void* content)
{
    ListLink* linkToAdd;
    linkToAdd = (ListLink*) malloc(sizeof(ListLink));
    linkToAdd->next = NULL;
    linkToAdd->previous = ListLinkTail(objectList);
    linkToAdd->content = content;

    if (linkToAdd->previous != NULL)
    {
        linkToAdd->previous->next = linkToAdd;
    }
    return linkToAdd;
}


ListLink* ListLinkDelete(ListLink* linkToDelete)
{
    ListLink* linkToReturn;
    if (linkToDelete->next != NULL)
    {
        linkToDelete->next->previous = linkToDelete->previous;
    }
    linkToReturn = ListLinkHead(linkToDelete);
```

Продолжение листинга 4

```
    if (linkToReturn != linkToDelete)
    {
        linkToDelete->previous->next = linkToDelete->next;
    }
    else
    {
        linkToReturn = linkToDelete->next;
    }
    free(linkToDelete->content);
    free(linkToDelete);
    return linkToReturn;
}


void ListLinkSwap(ListLink* firstLink, ListLink* secondLink)
{
    void* contentBox;
    contentBox = firstLink->content;
    firstLink->content = secondLink->content;
    secondLink->content = contentBox;
}


int ListLinkSize(ListLink* objectLink)
{
    ListLink* position;
    if (objectLink == NULL)
    {
        return 0;
    }
    position = ListLinkHead(objectLink);
    int size;
    size = 1;
    while (position->next != NULL)
    {
        size++;
        position = position->next;
    }
    return size;
}
```

Окончание листинга 4

```
void
ListLinkBubbleSort(ListLink* objectLink, bool(* comparator)(void*, void*))
{
    ListLink* position;
    position = NULL;
    ListLink* index;
    index = NULL;
    for (position = ListLinkHead(objectLink);
         position->next != NULL; position = position->next)
    {
        for (index = position->next;
             index != NULL; index = index->next)
        {
            if (comparator(index->content, position->content))
            {
                ListLinkSwap(position, index);
            }
        }
    }
}


void ListLinkFree(ListLink* objectLink)
{
    ListLink* i;
    ListLink* linkToFree;
    i = ListLinkHead(objectLink);
    while (i != NULL)
    {
        linkToFree = i;
        i = i->next;
        free(linkToFree->content);
        free(linkToFree);
    }

}
```

Листинг 5 - Код программы, решающей задачу, файл Data.h

```
#ifndef DATA_H
#define DATA_H

#include "Constants.h"


typedef struct
{
    char owner[INPUT_SIZE];
    int ordDay;
    int ordMonth;
    int ordYear;
    int shipDay;
    int shipMonth;
    int shipYear;
} Order;


typedef struct
{
    char name[INPUT_SIZE];
    int price;
    int weight;
} Product;


typedef struct
{
    Product* product;
    Order* order;
} Relation;


void PrintOrder(Order* objectOrder);

void PrintProduct(Product* objectProduct);

void PrintAllOrders(ListLink* objectList);

void PrintAllProducts(ListLink* objectList);
```

Окончание листинга 5

```
void PrintProductsByOrder(Order* objectOrder, ListLink* relations);

void PrintOrdersByProduct(Product* objectProduct, ListLink* relations);

ListLink* FindOrder(ListLink* objectList, char* owner);

ListLink* FindProduct(ListLink* objectList, char* name);

ListLink* FindRelation(ListLink* objectList, Order* order, Product* product);

ListLink* AddOrder(ListLink* objectList, char* owner, int ordDay,
                   int ordMonth, int ordYear, int shipDay, int shipMonth,
                   int shipYear);

ListLink* AddProduct(ListLink* objectList, char* name, int price, int weight);

ListLink* AddRelation(ListLink* objectList, ListLink* order,
                      ListLink* product);

ListLink* DeleteRelation(ListLink* relationToDelete);

ListLink* DeleteOrder(ListLink* orderLinkToDelete, ListLink** relations);

ListLink* DeleteProduct(ListLink* productLinkToDelete, ListLink** relations);

void PrintFileOrder(FILE* saveFile, Order* objectOrder);

void PrintFileProduct(FILE* saveFile, Product* objectProduct);

void PrintFileRelation(FILE* saveFile, Relation* relation);

#endif //DATA_H
```

Листинг 6 - Код программы, решающей задачу, файл Data.c

```
#include <stdio.h>
#include <malloc.h>
#include <string.h>

#include "ListLink.h"
#include "Data.h"
```

Продолжение листинга 6

```
void PrintOrder(Order* objectOrder)
{
    printf("ORDER by: %s\nDate of order: %d.%d.%d\nDate of shipment: "
           "%d.%d.%d\n", objectOrder->owner, objectOrder->ordMonth,
           objectOrder->ordDay, objectOrder->ordYear, objectOrder->shipMonth,
           objectOrder->shipDay, objectOrder->shipYear);
}


void PrintFileOrder(FILE* saveFile, Order* objectOrder)
{
    fprintf(saveFile, "%s\n%d\n%d\n%d\n%d\n%d\n%d\n", objectOrder->owner,
            objectOrder->ordMonth, objectOrder->ordDay, objectOrder->ordYear,
            objectOrder->shipMonth,
            objectOrder->shipDay, objectOrder->shipYear);
}

void PrintProduct(Product* objectProduct)
{
    printf("PRODUCT: %s\nPrice: %d\nWeight: %d\n", objectProduct->name,
           objectProduct->price, objectProduct->weight);
}


void PrintFileProduct(FILE* saveFile, Product* objectProduct)
{
    fprintf(saveFile, "%s\n%d\n%d\n", objectProduct->name,
            objectProduct->price,
            objectProduct->weight);
}


void PrintAllOrders(ListLink* objectList)
{
    ListLink* i = ListLinkHead(objectList);
    while (i != NULL)
    {
        PrintOrder(i->content);
        printf("\n");
```

Продолжение листинга 6

```
        i = i->next;
    }
}


void PrintAllProducts(ListLink* objectList)
{
    ListLink* i;
    i = ListLinkHead(objectList);
    while (i != NULL)
    {
        PrintProduct(i->content);
        printf("\n");
        i = i->next;
    }
}


void PrintProductsByOrder(Order* objectOrder, ListLink* relations)
{
    PrintOrder(objectOrder);
    printf("----Contents:----\n");
    ListLink* i = ListLinkHead(relations);
    while (i != NULL)
    {
        Relation* relation;
        relation = i->content;
        if (relation->order == objectOrder)
        {
            PrintProduct(relation->product);
        }
        i = i->next;
    }
}


void PrintOrdersByProduct(Product* objectProduct, ListLink* relations)
{
    PrintProduct(objectProduct);
    printf("----Contents:----\n");
    ListLink* i = ListLinkHead(relations);
```

Продолжение листинга 6

```
    while (i != NULL)
    {
        Relation* relation;
        relation = i->content;
        if (relation->product == objectProduct)
        {
            PrintOrder(relation->order);
        }
        i = i->next;
    }
}


void PrintFileRelation(FILE* saveFile, Relation* relation)
{
    fprintf(saveFile, "%s\n%s\n", relation->product->name,
            relation->order->owner);
}


ListLink* FindOrder(ListLink* objectList, char* owner)
{
    ListLink* i;
    i = ListLinkHead(objectList);
    while (i != NULL)
    {
        Order* order;
        order = i->content;
        if (strcmp(order->owner, owner) == 0)
        {
            return i;
        }
        i = i->next;
    }
    return NULL;
}


ListLink* FindProduct(ListLink* objectList, char* name)
{
    ListLink* i;
```

Продолжение листинга 6

```
    i = ListLinkHead(objectList);
    while (i != NULL)
    {
        Product* product;
        product = i->content;
        if (strcmp(product->name, name) == 0)
        {
            return i;
        }
        i = i->next;
    }
    return NULL;
}


ListLink* FindRelation(ListLink* objectList, Order* order, Product* product)
{
    ListLink* i;
    i = ListLinkHead(objectList);
    while (i != NULL)
    {
        Relation* relation;
        relation = i->content;
        if (relation->order == order && relation->product == product)
        {
            return i;
        }
        i = i->next;
    }
    return NULL;
}


ListLink* AddOrder(ListLink* objectList, char* owner, int ordDay,
                   int ordMonth, int ordYear, int shipDay, int shipMonth,
                   int shipYear)
{
    if (FindOrder(objectList, owner) != NULL)
    {
        return NULL;
    }
```

Продолжение листинга 6

```
    Order* order;
    order = (Order*) malloc(sizeof(Order));
    strcpy(order->owner, owner);
    order->ordDay = ordDay;
    order->ordMonth = ordMonth;
    order->ordYear = ordYear;
    order->shipDay = shipDay;
    order->shipMonth = shipMonth;
    order->shipYear = shipYear;

    return ListLinkAdd(objectList, order);
}


ListLink* AddProduct(ListLink* objectList, char* name, int price, int weight)
{
    if (FindProduct(objectList, name) != NULL)
    {
        return NULL;
    }

    Product* product;
    product = (Product*) malloc(sizeof(Product));
    strcpy(product->name, name);
    product->weight = weight;
    product->price = price;

    return ListLinkAdd(objectList, product);
}


ListLink* AddRelation(ListLink* objectList, ListLink* order,
                      ListLink* product)
{

    if (FindRelation(objectList, order->content, product->content) != NULL)
    {
        return NULL;
    }
```

Продолжение листинга 6

```
    Relation* relation;
    relation = (Relation*) malloc(sizeof(Relation));
    relation->product = product->content;
    relation->order = order->content;

    return ListLinkAdd(objectList, relation);
}


ListLink* DeleteRelation(ListLink* relationToDelete)
{
    if (relationToDelete == NULL)
    {
        return NULL;
    }
    return ListLinkDelete(relationToDelete);
}


ListLink* DeleteOrder(ListLink* orderLinkToDelete, ListLink** relations)
{
    if (orderLinkToDelete == NULL)
    {
        return NULL;
    }
    ListLink* i = ListLinkHead(*relations);
    while (i != NULL)
    {
        ListLink* relationLinkToDelete;
        Relation* relation;
        relation = i->content;
        relationLinkToDelete = i;
        i = i->next;
        if (relation->order == orderLinkToDelete->content)
        {
            *relations = DeleteRelation(relationLinkToDelete);
        }
    }
    return ListLinkDelete(orderLinkToDelete);
}
```

Окончание листинга 6

```
ListLink* DeleteProduct(ListLink* productLinkToDelete, ListLink** relations)
{
    ListLink* i = ListLinkHead(*relations);
    while (i != NULL)
    {
        ListLink* relationLinkToDelete;
        Relation* relation;
        relation = i->content;
        relationLinkToDelete = i;
        i = i->next;
        if (relation->product == productLinkToDelete->content)
        {
            *relations = DeleteRelation(relationLinkToDelete);
        }
    }
    return ListLinkDelete(productLinkToDelete);
}
```

Листинг 7 - Код программы, решающей задачу, файл Input.h

```
#ifndef INPUT_H
#define INPUT_H

#include <stdbool.h>


int CycleInputInt(char* stringToOutput, bool(* pChecker)(int));

char* StrDynInput();

char* CycleInputString(char* stringToOutput);


#endif //INPUT_H
```

Листинг 8 - Код программы, решающей задачу, файл Input.c

```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
```

Продолжение листинга 8

```
#include <malloc.h>

#include "Input.h"
#include "Constants.h"

enum SymbolCodes
{
    START_CHAR_RANGE = 32,
    END_CHAR_RANGE = 126,
    BACKSPACE_KEY = 8
};


int CycleInputInt(char* stringToOutput, bool(* pChecker)(int))
{
    int number;
    int position;
    char input[INPUT_SIZE];

    while (true)
    {
        printf("%s\n", stringToOutput);
        fflush(stdout);
        char* fgetsRet = fgets(input, INPUT_SIZE, stdin);
        if (fgetsRet == NULL)
        {
            printf("Wrong format!\n");
            continue;
        }
        int inputLength = strlen(input) - 1;
        input[inputLength] = '\0';
        int sscanfRet = sscanf(input, "%d%n", &number, &position);
        if (position != inputLength)
        {
            printf("Wrong format!\n");
            continue;
        }
        if (pChecker && !pChecker(number))
        {
            printf("Wrong format!\n");
            continue;
```

Продолжение листинга 8

```
        }
        if (sscanfRet == 1) break;
        printf("Wrong format!\n");
    }
    return number;
}


char* StrDynInput()
{
    char* userStr = (char*) malloc(1 * sizeof(char));
    userStr[0] = '\0';
    char curChar = 0;
    int curSize = 1;
    while (curChar != '\n')
    {
        curChar = getchar();
        int deltaVal = 0;
        int lengthDif = 0;
        if (curChar == BACKSPACE_KEY)
        {
            deltaVal = -1;
            lengthDif = 1;
        }
        else
        {
            if (curChar >= START_CHAR_RANGE && curChar <=
                                                END_CHAR_RANGE)
            {
                deltaVal = 1;
                lengthDif = 2;
            }
            else
            {
                continue;
            }
        }
        int newSize = curSize + deltaVal;
        if (newSize == 0)
        {
            continue;
```

Окончание листинга 8

```
      }
        char* tmpStr = (char*) malloc(newSize * sizeof(char));
        if (tmpStr)
        {
            for (int i = 0; i < newSize - lengthDif; ++i)
            {
                tmpStr[i] = userStr[i];
            }

            if (curChar != BACKSPACE_KEY)
            {
                tmpStr[newSize - 2] = curChar;
            }
            tmpStr[newSize - 1] = '\0';
            free(userStr);
            userStr = tmpStr;
            curSize = newSize;
        }
        else
        {
            printf("Couldn't allocate memory!");
            break;
        }
    }

    return userStr;
}


char* CycleInputString(char* stringToOutput)
{
    printf("%s\n", stringToOutput);
    return StrDynInput();
```

5 Результат

Ниже представлены скриншоты с консольным выводом.

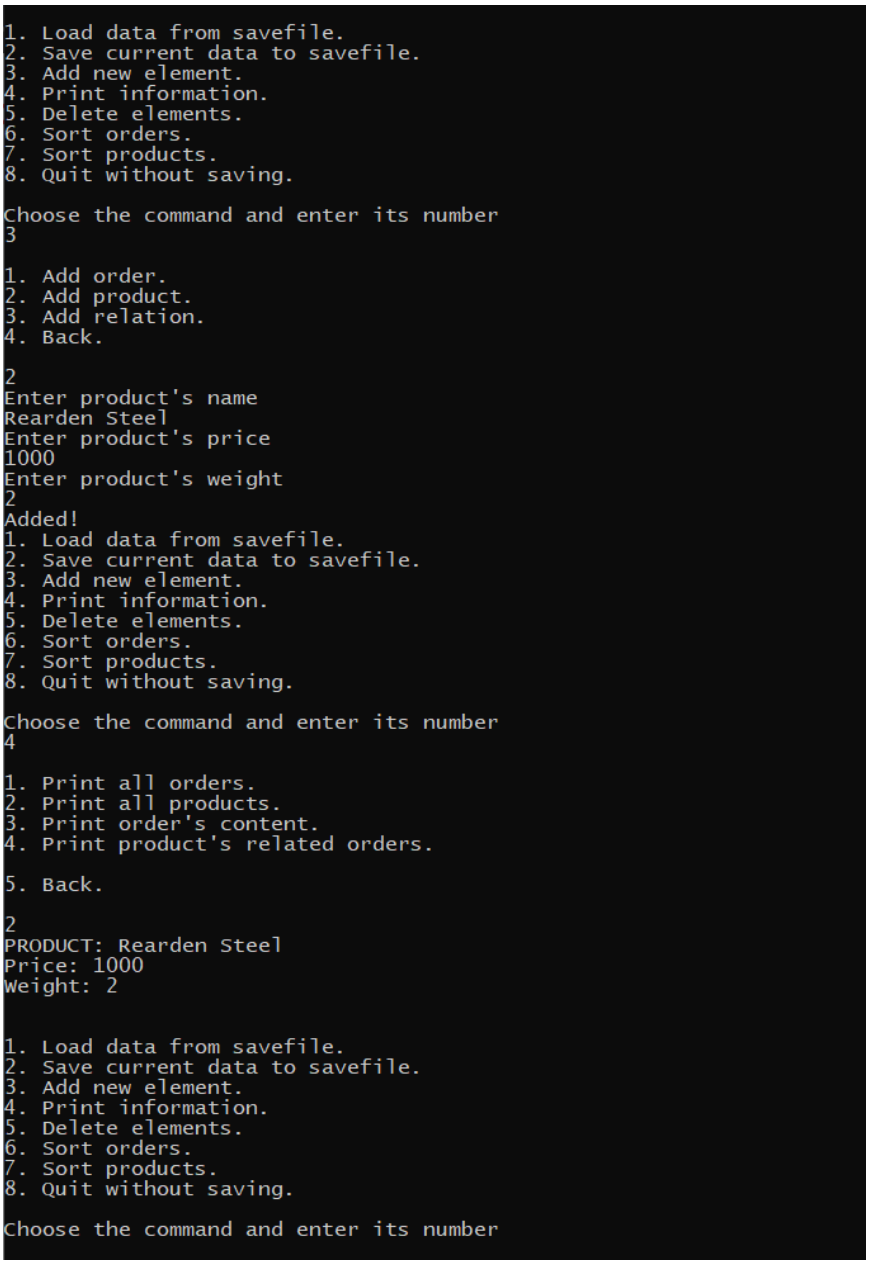

Рисунок 1 – Создание нового товара

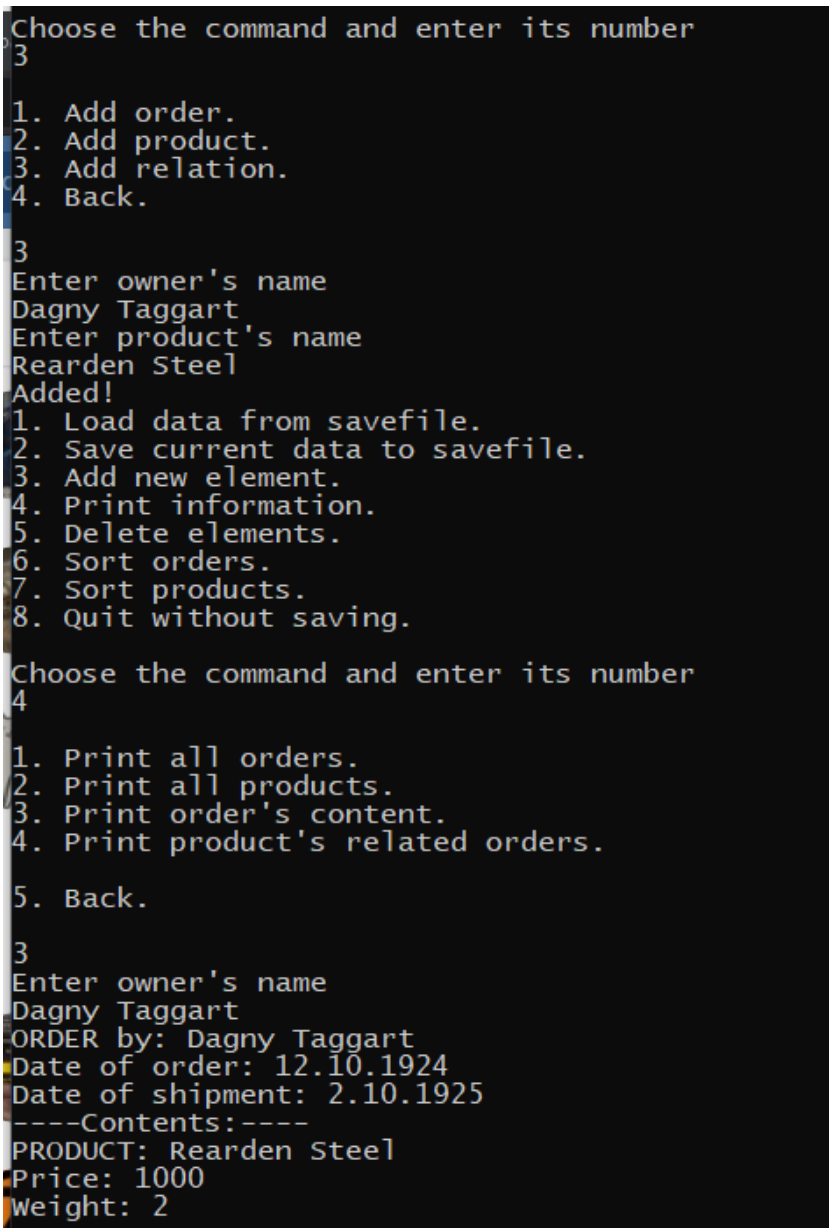

Рисунок 2 – Добавление связи товара и заказа

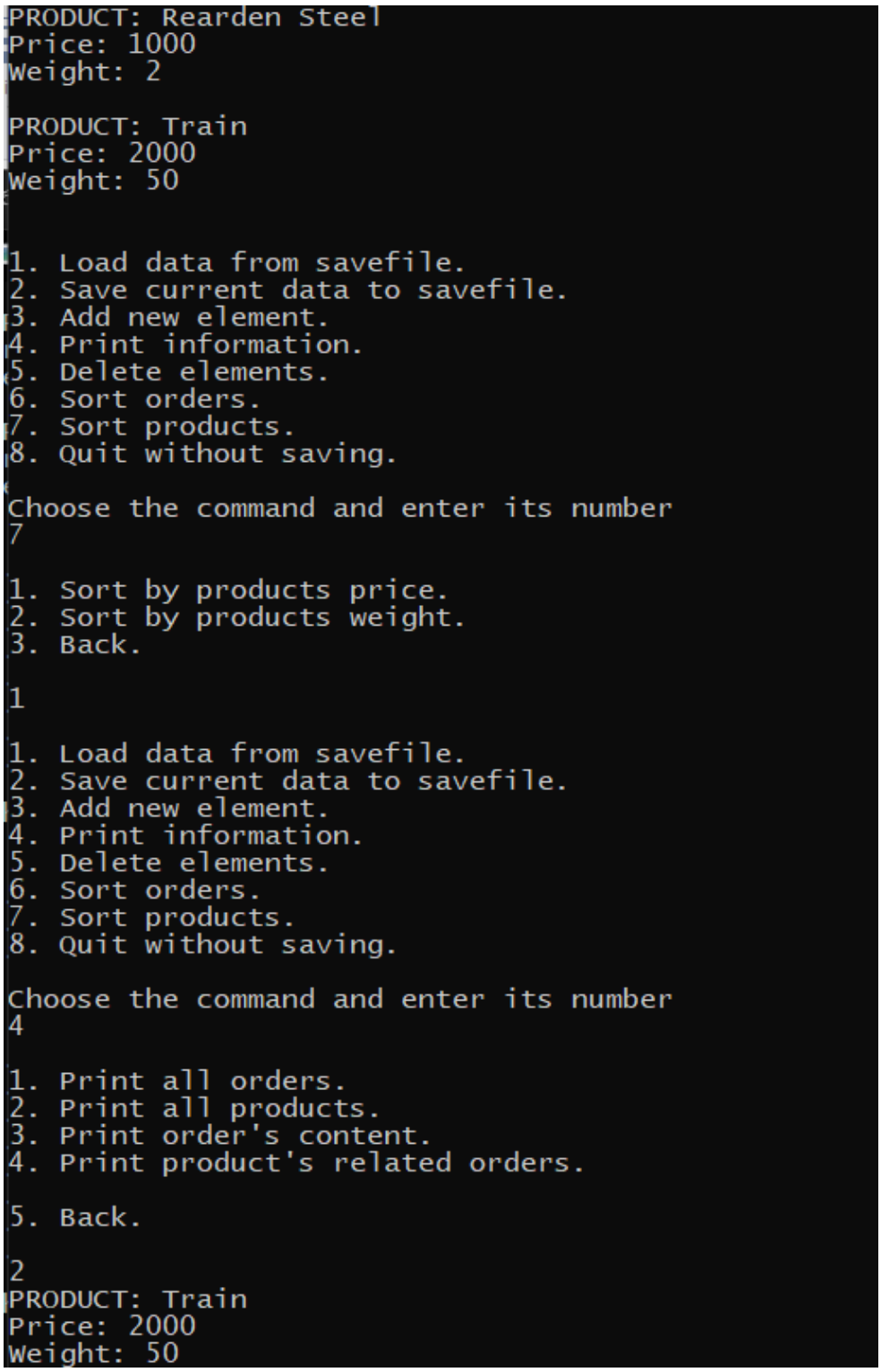

Рисунок 3 – Сортировка продуктов по цене

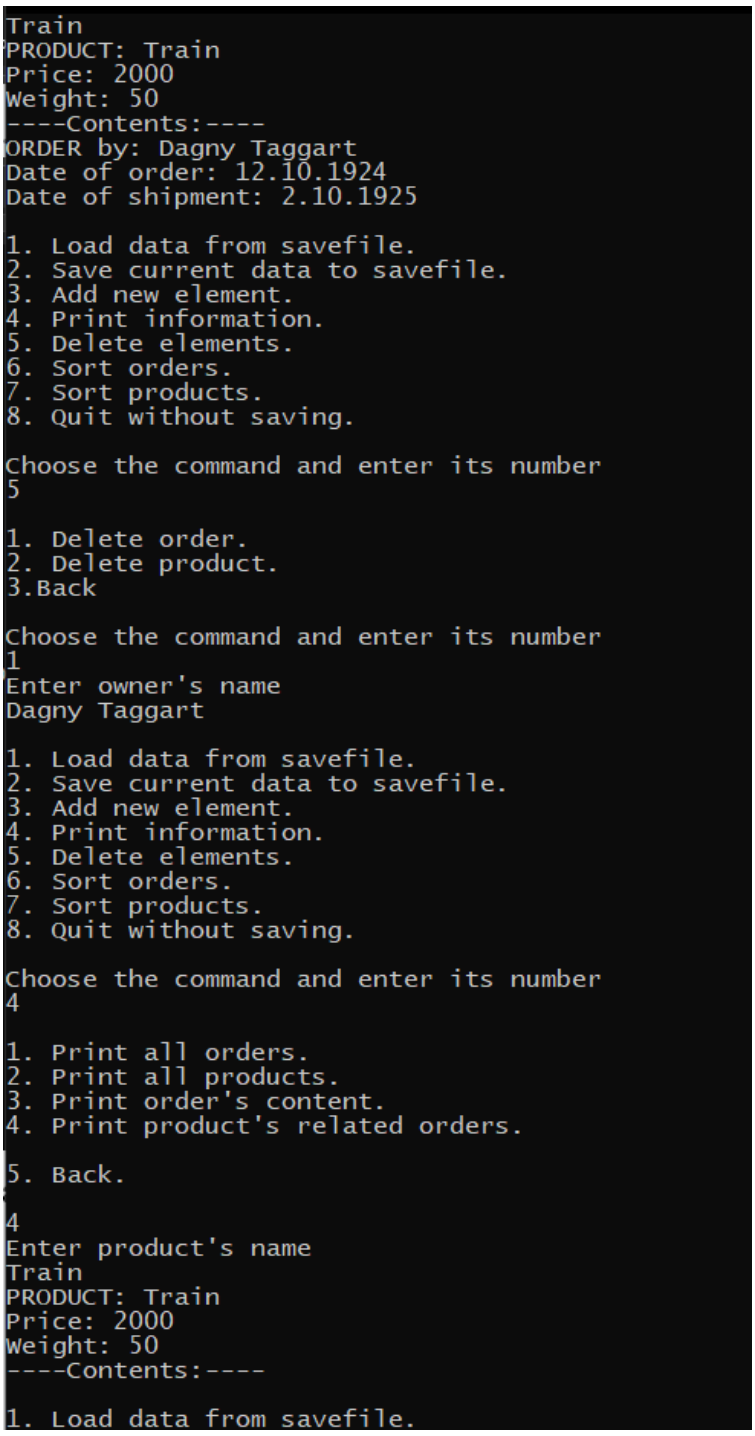

Рисунок 4 – Удаление элемента

6 Выводы

Была написана программа, соответствующая поставленным задачам. В ходе работы было продолжено изучение такой базовой структуры данных в информатике, как связный список, а также изучены некоторые аспекты синтаксиса языка программирования Си при работе с такими структурами. Код был отформатирован в соответствии со стандартом, принятым в учебном заведении.