Student Union POS System

Hong Kong Examinations and Assessment Authority

Hong Kong Diploma of Secondary Education Examination 2024 Information and Communication Technology (SBA)

Sha Tin Methodist College

AU KA LONG 6A (14)

Table of Contents

1. Introduction

A. Student Union’s shop

The student union of a secondary school in Hong Kong is planning to provide a self-serve stationery shop. The shop will offer various stationery items. However, managing the shop remains as a challenge.
One of the main challenges is the lack of a proper system to record the sales and inventory of the stationery items. Currently, the student union relies on manual methods like physical counting to keep track of the transactions and stock levels. These methods are vulnerable to human errors. Moreover, they do not provide any insights into the sales trends, customer preferences, or profitability of the shop.

Another challenge is the lack of labour to manage the store. The student union has limited resources and volunteers to operate the shop during school hours. This means that the shop may not be open at all times or may have long queues and waiting times for the customers. Moreover, it may be difficult to train and supervise the staff to ensure quality and consistency in the service.

To overcome these challenges, the student union needs a POS (Point of Sale) system that can automate and streamline the operations of the stationery shop. A POS system is a computerized network that consists of a main computer linked with several checkout terminals and supported by different hardware features such as barcode scanners and card payment terminals. A POS system can offer several benefits for the student union and the customers, such as:

Improved accuracy and efficiency: A POS system can scan the barcode of the stationery items and automatically calculate the total amount and change due for each transaction. It can also update the inventory levels in real time and alert the staff when an item is running low or out of stock. This can reduce human errors, save time, and

increase productivity.

Enhanced security and accountability: A POS system can track and record every transaction and inventory movement in a secure database. It can also generate reports and receipts that can be used for auditing and accounting purposes. This can prevent fraud, theft, and discrepancies in the shop.

Better customer service and satisfaction: A POS system can offer more payment options for the customers such as credit cards, debit cards, mobile wallets, etc. It can also provide faster checkout times, personalized offers, loyalty programs, etc. This can improve customer convenience, loyalty, and retention.

Greater insights and decision making: A POS system can analyse the data collected from the transactions and inventory movements and provide useful information such as sales trends, customer preferences, best-selling items, profit margins, etc. This can help the student union to optimize their pricing, marketing, purchasing, and stocking strategies.

Reduced labour costs and increased flexibility: A POS system can enable customers to self-serve by scanning and paying for their own items without the need for staff assistance. This can reduce the labour costs and workload for the student union and allow them to focus on other tasks or activities. It can also increase the flexibility of opening hours and locations for the shop.

Therefore, the POS system would be a suitable choice for the student union. In this proposal, I will describe the prototype of the POS system for the student union’s stationery shop.

B. Goal of the program

1. Allow self-serve, freeing staff[1]
2. Accurate transaction management
3. Advanced sorting for management
4. Simple UI for prototyping
5. Preserve easy way to extend the system
6. Preserve cross-platform compatibility[2]
7. Back-end code and front-end code is seperate[3]

[1] May required installation of CCTV, and AI detection system to alert theft to free staff totally

[2] Allows deploying on either common laptop or Linux enterprise devices

[3] Allows to separate front-end and back-end to server and client in the future

2. Program Design

A. Requirements

System	Windows/Linux (or Other Unix-like OS)
CPU	Any
Storage	Different for OS (small memory required) Program: 50-100KIB Storage: 1-10KIB(get larger by time)
Compiler	GNU Compiler Collection (GCC)
Compiler args	-lm

B. Functionality of the program

Basic functionality

The program provides the following functionality:

1. Self-serve Sales Transaction Handling
    a. Searching items
    b. Listing items (Include Sorting)
    c. Adding items to cart
    d. Pay
2. Admin Management
    a. Inventory management[1]
    b. Transaction management[1]
    c. Advanced user management[1,2]
3. User Authentication
    a. To allow access admin
    b. Allow payment by student account[3]
4. Basic Storage[4]

[1] Management includes:

- Listing
- Sorting (Allow finding hot items, and more advanced analysis)
- Editing (Full Control on stuff)

[2] Advanced means include roles

[3] Relies on school payment system

[4] Using Simple file RW, should include encryption and/or upgrade to database in deployment

Assumption

1. User is assumed to be registered in the School Intranet, which the data is share to this system, which also allowing payment through the School Intranet
2. An ID card reader is assumed to be installed, and all students and school staffs should have their own ID card

C. Design philosophy

Modular programming

Despite C is not an Object-oriented programming, it can still imply principles of modular programming. For example, splitting program into different sub-programs and creating header file for different part or scene.
Modular programming is heavily adopted in this program. The program is split to multiple part(splitting different modules into different header files), and different data is supported by using “struct”.
The above measures ensure the program to be readable and maintainable in long run. Also, it makes code more reusable, less code need to be written when a new kind of data is introduced. For instance, adding data type “user”, all of the common db, common listing function, common sorting function can be reuse. The amount of work to implement new kinds of feature is greatly reduced.

D. Interface of the program

This program implements a minimalist stylish interface, while the program preserves easy way to tweak and to upgrade the interface[1].

This program uses general inference for different stages.

1. General Menu
2. General List
3. General Choice Prompt

[1] the actual methodology would be discussed in the program structure part

The other message is rather normal, so and in same style. Therefore, only general interface and some example interface would be shown here.

General Menu

[Welcome message/Header info]
[Select An Option]
1. Options 1
2. Option 2
...
N+1. [Exit]
>[input]

```
><Welcome to the Student Union POS system><
Select An Option:
1. Admin
2. Normal User
3. Exit
>
```

Examples

```
><Welcome to the Student Union POS system><
Select An Option:
1. scan barcode
2. cart/checkout
3. Exit
>
```

```
><Welcome to the Student Union POS system><
Normal User Menu
Select An Option:
1. List items
2. Search item
3. Self help sale system
4. Exit
>
```

```
><Welcome to the Student Union POS system><
Select An Option:
1. scan barcode
2. cart/checkout
3. Exit
>4
Invalid choice...press any key to retry....
```

Data validation is also be done in this interface, further methodology will be discussed in Interface Implementation.

General List

[List name]

[Exit]

[Sort Options]

[Items]

[page control options]

[page info]

>[input]

```
><Welcome to the Student Union POS system><
List of items list
0 exit
1 sort Name decending
2 sort Name ascending
3 sort Price decending
4 sort Price ascending
5 sort Brand decending
6 sort Brand ascending
7 sort Category decending
8 sort Category ascending
No.  Product        Brand          Category  Price     Stock     Barcode
9    Testing Pr...  Testing Co...  Test      15.00     0         1122
10   Orange Jui...  Mr Juicy       Juice     9.90      0         9665
11   Apple & Al...  F&N Fruit ...  Juice     16.00     0         6479
12   Apple & Al...  F&N Fruit ...  Juice     22.90     0         5155
13   Cold Press...  Homegrown ...  Juice     35.50     0         6470
14   T2 Product     T2 Brand       T2        20.00     0         2983
15 next page
16 previous page
17 set page size
20/1/1(page size/page number/total)
Please input your choice
>
```

Pagination

Using pagination avoid overflooding the whole screen. Overflooding would cause information hard to locate when the number of data scale up.

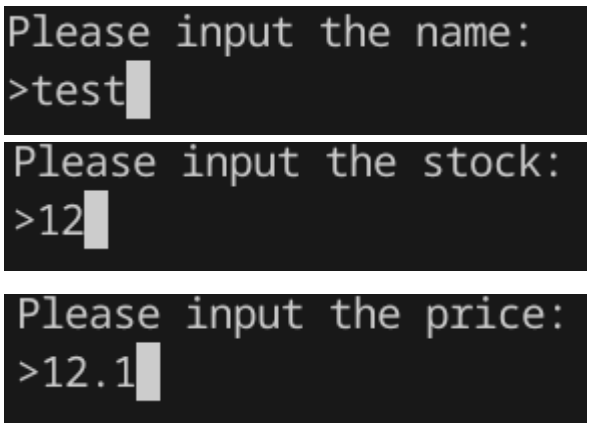
Also, an option allows user can adjust the page size according to their display size is provided. User can have the most information they need in good format.

Formatting

Fixed column size is implemented. This addresses situation when a column string or information is oversized, which causes character overflowing to the next column or inconsistent foramt. For example, an item having a name of 20 characters. So, string compression(i.e. limiting the number of characters to display for a string) is implemented. This technique can keep the column size fixed, so that no overflow and overlapping will occurs, while keeping a reasonable amount of information. If the user wants more information, they can select the item for full information.

General Input prompt

[Input prompt]:
>[Input]



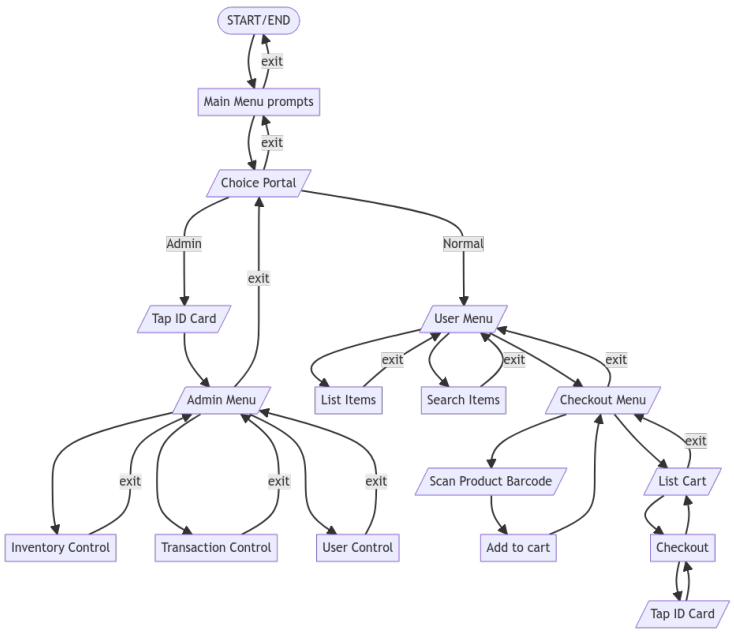
Data verification is done to ensure the data input is the correct format(char* ,int, long, etc.)

3. Program Implementation

A. Introduction

In this section, Program Implementation, it would contain explanation of key codes. And it would be split into parts of overview, data processing, interface implementations, optimizations. At the very end, the full code would be included.

B. Overview

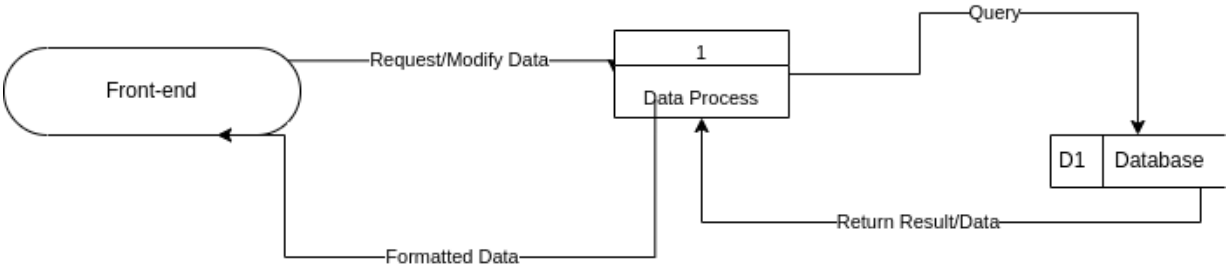


The above flowchart shows the flow of how the user should interact with the program.
* Simple Note: Authentication is requiring on hardware endpoints of tapping cards, as hardware is not present, so the Authentication part will not be mention. On production, encryption would be used to communicate, ensuring high security level.

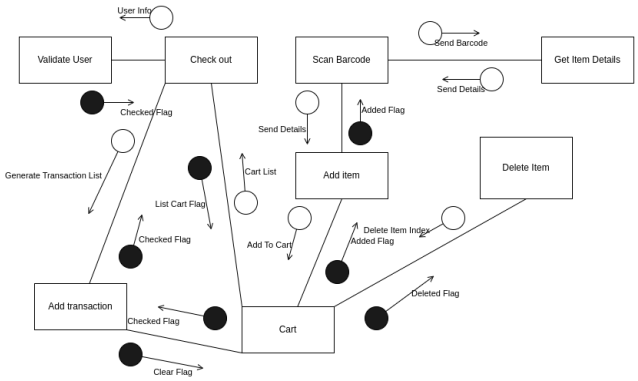
C. Data Processing

Overview

This program use a modular structure to process different kinds of data.
Each module has a similar structure and similar way to process data.



For the database implementation, a simple file management is being used, while the structure of the program allows to switch to other type of implementation int the future.
The designed structure in below:



List of files

1. data_file_inventory.txt (Inventory)
2. data_file_transaction.txt (Transaction)
3. data_file_user.txt (User List)
4. data_file_admin.txt (Admin Roles List)

General structure and Common functions

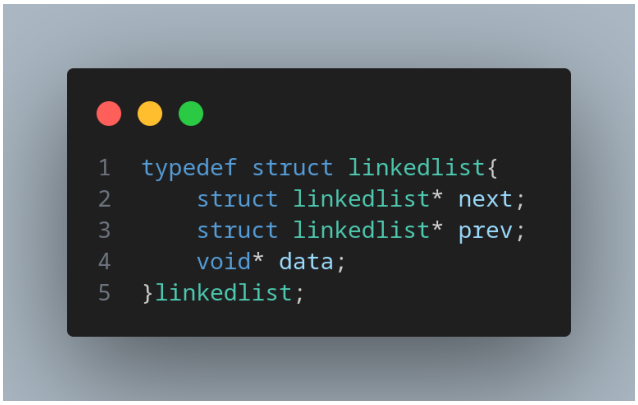
General Database struct for common attributes of databases

typedef struct basic_db{
    int row_count;
    bool init_status;//not really used can be useful
    //array of struct of row
}basic_db;

row_count: Use to save the number of rows

init_status: Indicates whether the structure initialize successfully

Linked List



* Structure of the Linked List

Function for Linked List

```c
int sizeofLinkedlist(struct linkedlist* list){
    linkedlist* start = list;
    int size = 0;
    linkedlist* current = list;
    while (current != NULL){
        size++;
        current = current->next;
    }
    current = start;
    while(current->prev != NULL){
        size++;
        current = current->prev;
    }
    return size;
}
struct linkedlist* getLinkedList(struct linkedlist* list,int pos){
    linkedlist* cur = list;
    while(cur->prev != NULL){
        cur = cur->prev;
    }
    struct linkedlist* start = cur;
    for(int i=0;i<pos;i++){
        cur = cur->next;
        if(cur == NULL){
            return start;//return start if pos is out of bounds
        }
    }
    return cur;
}
```

Linked list is used for cases:

1. When the size cannot be predefined
2. When deleting/adding items in between item is rapidly needed

It would required a lot of memory and operation for same situation when using array.

Example case:

- Shopping Cart management
    - As the number of items is changing on every add/delete
    - Deletion in between item is highly required

Pointer is heavily used in implementing Linked list, as it provides flexible control on elements.

Inventory

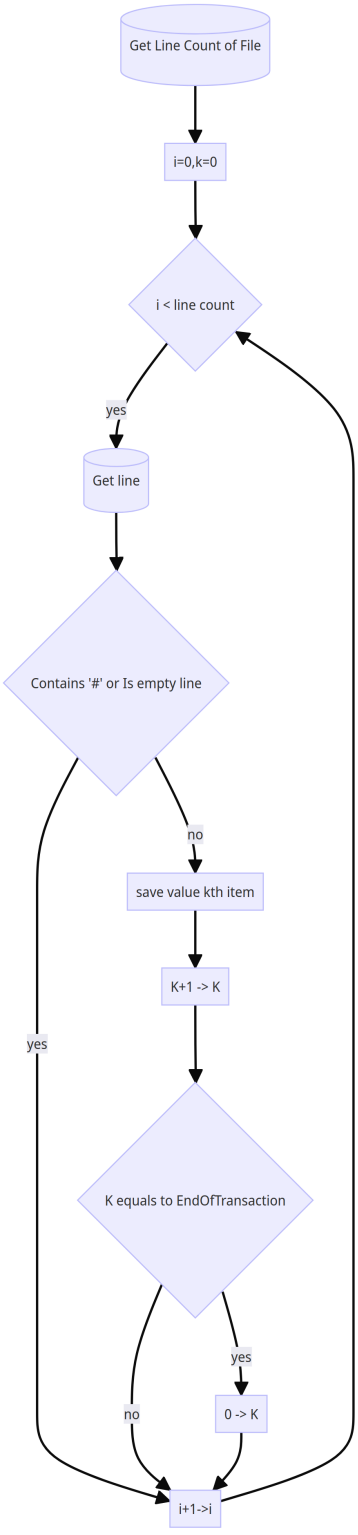
Structures of Inventory

```c
#define INVENTORY_DB "data_file_inventory.txt"
typedef struct inventory{
    struct basic_db db;
    struct inventory_row* row;

}inventory;
typedef struct inventory_row{
    char category[100];
    char brand[100];
    char product[100];
    double price;
    int stock;
    long barcode;
    bool isdeleted;//common for all rows,default is false
}inventory_row;
```

Function of Inventory

```c
struct inventory read_db_invt(){//please open file in read mode
    FILE* fp = fopen(INVENTORY_DB, "r");
    struct inventory db;

    //gets the number of rows in the file
    int row_count = basic_get_row_count(ENDOFINVENTORY,fp);
    db.row = (struct inventory_row*)malloc(row_count * sizeof(struct inventory_row));
    db.db.row_count = row_count;

    fseek(fp,0,SEEK_SET);//reset fp to the beginning of the file

    //read data
    for(int i=0;i<row_count;i++){
        db.row[i].isdeleted = false;
        for(INVENTORY j=category;j<=ENDOFINVENTORY;j++){
            char buffer[100];
            fgets(buffer, sizeof(buffer),fp);
            if(unlikely(buffer[0] == '#')){//catch comment line and ignore
                j--;//decrement j to prevent skipping next column
            }else{
                buffer[strlen(buffer)] = '\0';
                while(buffer[strlen(buffer)-1] == '\n' || (int)buffer[strlen(buffer)-1] == 13){
                    buffer[strlen(buffer)-1] = '\0';

                }

                switch(j){
                    case category:
                        strcpy(db.row[i].category,buffer);
                        break;
                    case brand:
                        strcpy(db.row[i].brand,buffer);
                        break;
                    case product:
                        strcpy(db.row[i].product,buffer);
                        break;
                    case price_inv:
                        db.row[i].price = atof(buffer);
                        break;
                    case stock:
                        db.row[i].stock = atoi(buffer);
                        break;
                    case barcode_inv:
                        db.row[i].barcode = atol(buffer);
                        break;

                }
            }

        }
    }
    if(ferror(fp)){
        printf("Error in reading file\n");
        db.db.init_status = false;
        return db;
    }
    fclose(fp);
    db.db.init_status = true;
    return db;


}
```

Flowchart of reading data files

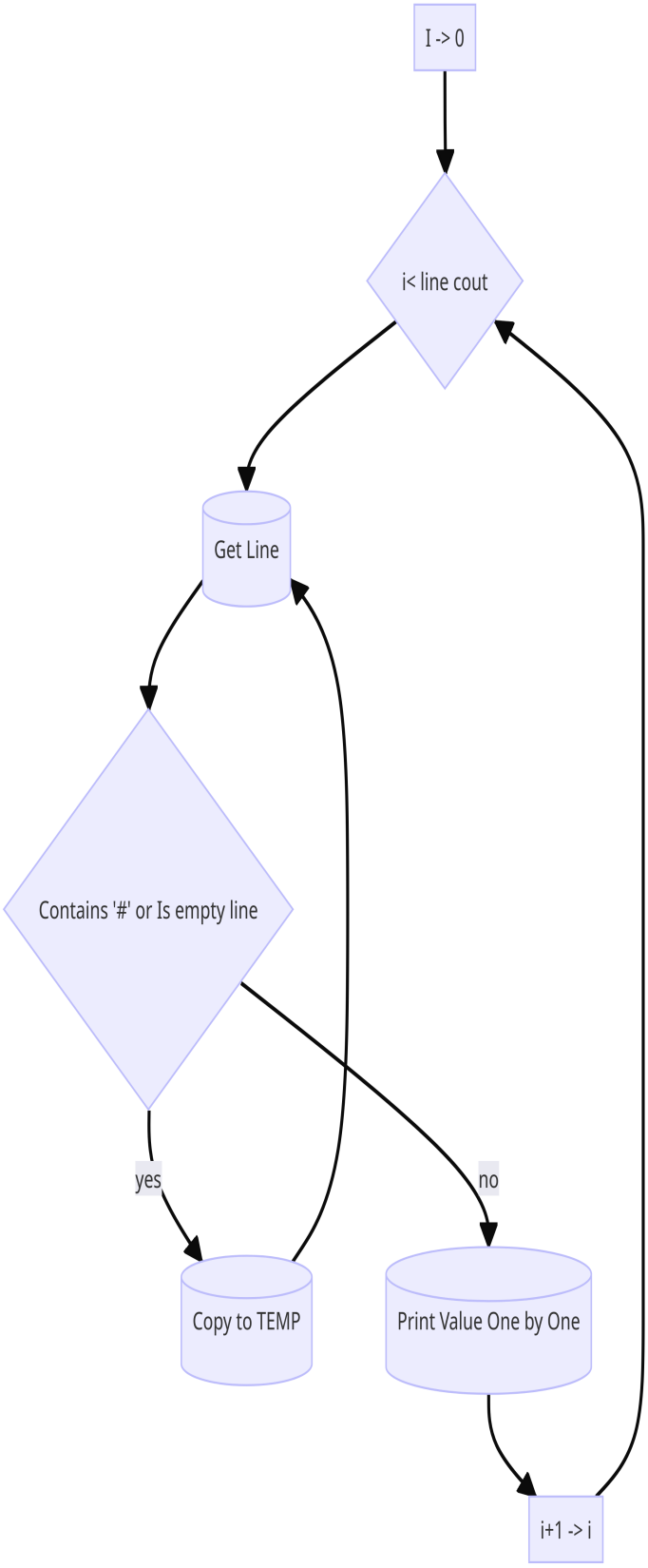


Actual Implementation

```c
typedef enum  {
    category = 1, brand = 2, product = 3, price_inv = 4, stock = 5, barcode_inv = 6
 }INVENTORY;
#define ENDOFINVENTORY 6
```

```c
struct inventory read_db_invt(){//please open file in read mode
    FILE* fp = fopen(INVENTORY_DB, "r");
    struct inventory db;

    //gets the number of rows in the file
    int row_count = basic_get_row_count(ENDOFINVENTORY,fp);
    db.row = (struct inventory_row*)malloc(row_count * sizeof(struct inventory_row));
    db.db.row_count = row_count;

    fseek(fp,0,SEEK_SET);//reset fp to the beginning of the file

    //read data
    for(int i=0;i<row_count;i++){
        db.row[i].isdeleted = false;
        for(INVENTORY j=category;j<=ENDOFINVENTORY;j++){
            char buffer[100];
            fgets(buffer, sizeof(buffer),fp);
            if(unlikely(buffer[0] == '#')){//catch comment line and ignore
                j--;//decrement j to prevent skipping next column
            }else{
                buffer[strlen(buffer)] = '\0';
                while(buffer[strlen(buffer)-1] == '\n' || (int)buffer[strlen(buffer)-1] == 13){
                    buffer[strlen(buffer)-1] = '\0';

                }

                switch(j){
                    case category:
                        strcpy(db.row[i].category,buffer);
                        break;
                    case brand:
                        strcpy(db.row[i].brand,buffer);
                        break;
                    case product:
                        strcpy(db.row[i].product,buffer);
                        break;
                    case price_inv:
                        db.row[i].price = atof(buffer);
                        break;
                    case stock:
                        db.row[i].stock = atoi(buffer);
                        break;
                    case barcode_inv:
                        db.row[i].barcode = atol(buffer);
                        break;

                }
            }

        }
    }
    if(ferror(fp)){
        printf("Error in reading file\n");
        db.db.init_status = false;
        return db;
    }
    fclose(fp);
    db.db.init_status = true;
    return db;
}
```

Flowchart of Writing Data



Actual Implementation

```c
bool update_db_invt(struct inventory invt){
    FILE* fpR = fopen(INVENTORY_DB,"r");
    char temp[30] = INVENTORY_DB;
    strcat(temp,".temp");
    FILE* fpW = fopen(temp,"w");
    if(fpR == NULL || fpW == NULL){
        printf("Error in opening file\n");
        return false;
    }
    for(int i=0;i<invt.db.row_count;i++){
        if(invt.row[i].barcode == -1024 || invt.row[i].isdeleted == true)//skip create new row
            continue;
        for(INVENTORY j=category;j<=ENDOFINVENTORY;j++){
            char buffer[100];
            do{
                if(unlikely(feof(fpR))){
                    break;
                }
                fgets(buffer, sizeof(buffer),fpR);
                if(buffer[0] == '#'){//catch comment line and print back to new file
                    fputs(buffer,fpW);
                }
            }while(buffer[0] == '#');
            switch(j){
                case category:
                    fprintf(fpW,"%s",invt.row[i].category);
                    break;
                case brand:
                    fprintf(fpW,"%s",invt.row[i].brand);
                    break;
                case product:
                    fprintf(fpW,"%s",invt.row[i].product);
                    break;
                case price_inv:
                    fprintf(fpW,"%.1f",invt.row[i].price);
                    break;
                case stock:
                    fprintf(fpW,"%d",invt.row[i].stock);
                    break;
                case barcode_inv:
                    fprintf(fpW,"%ld",invt.row[i].barcode);
                    break;
            }
            fprintf(fpW,"\n");
        }
    }
    //close file and replace old file with new file
    fclose(fpR);
    fclose(fpW);
    remove(INVENTORY_DB);
    rename(temp,INVENTORY_DB);
    return true;

}
```

Transaction

```c
typedef struct transaction{
    struct basic_db db;
    struct transaction_row* row;

}transaction;
typedef struct transaction_row{
    struct Date date;
    struct Time time;
    long id;
    double price;
    int quantity;
    long barcode;
    bool isdeleted;//common for all rows,default is false
}transaction_row;
```

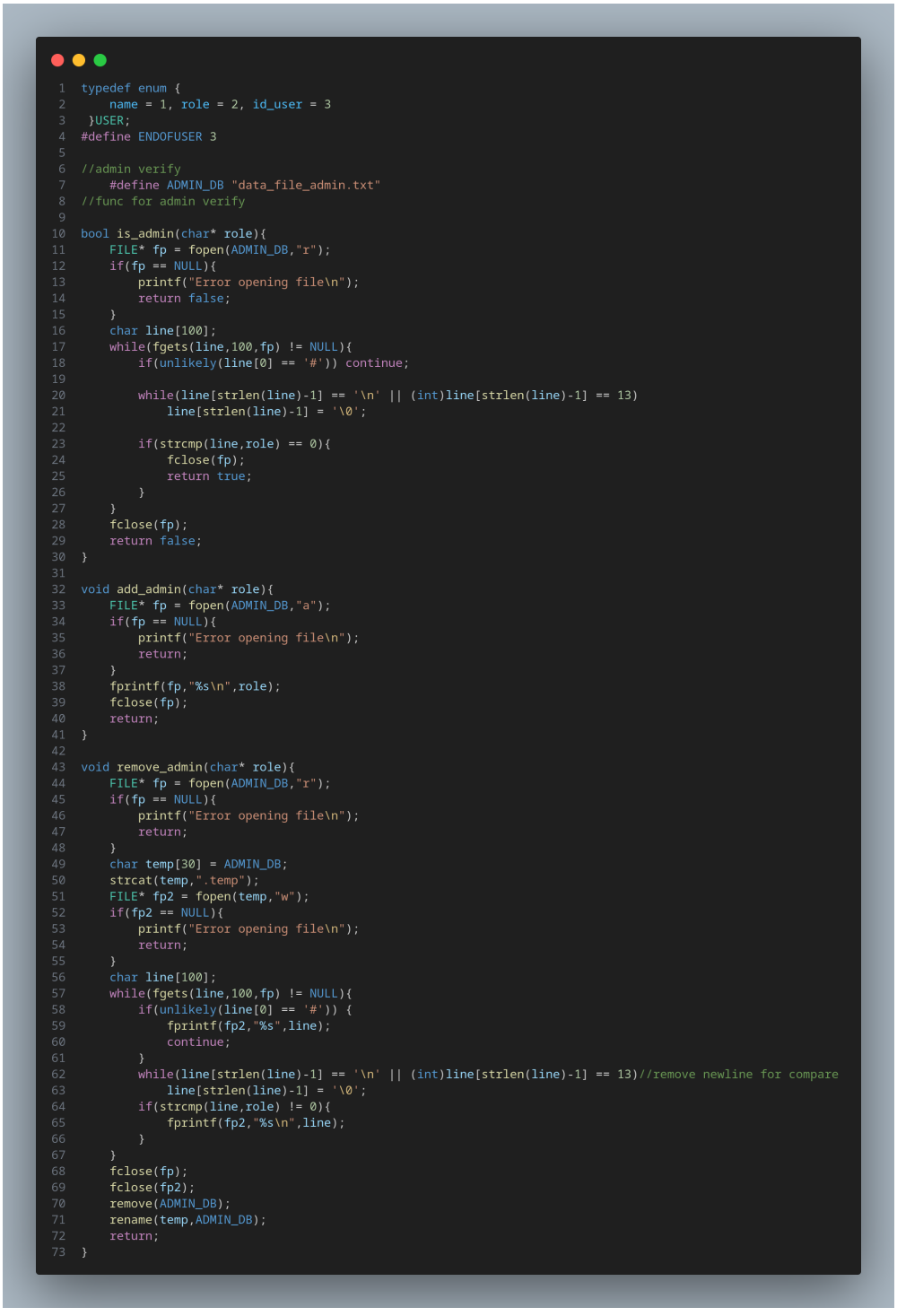
* The functions have similar workflow as the inventory module, you can see the actual implementation in the program listing part

User

```c
#define USER_DB "data_file_user.txt"
typedef struct user{
    struct basic_db db;
    struct user_row* row;
}user;
typedef struct user_row{
    char name[100];
    char role[100];
    long id;
    bool isdeleted;//common for all rows,default is false
}user_row;
```

* The functions have similar workflow as the inventory module, you can see the actual implementation in the program listing part

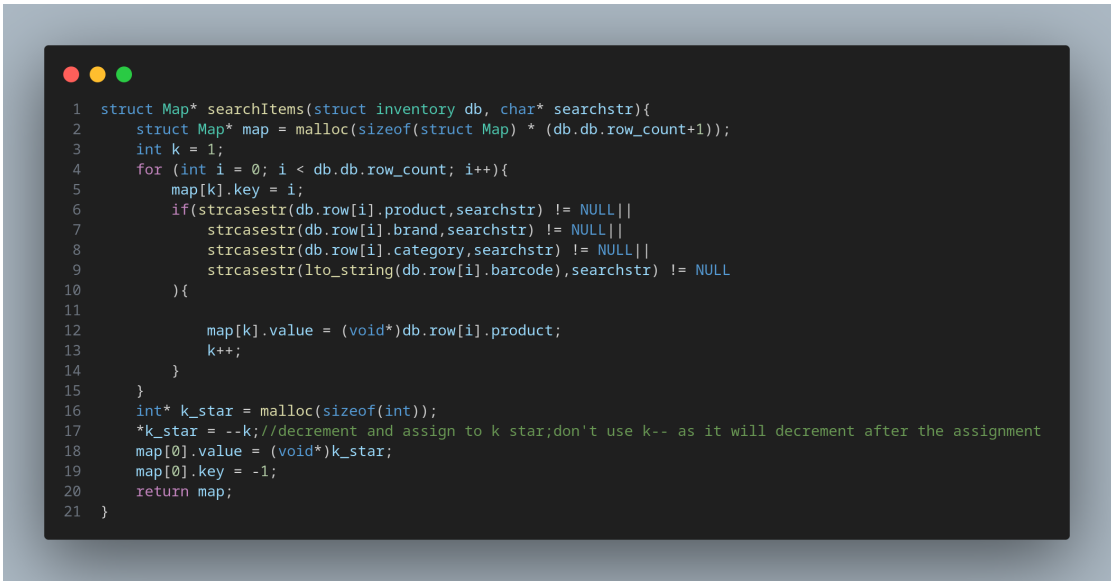
Admin Role



```c
typedef enum {
    name = 1, role = 2, id_user = 3
 }USER;
#define ENDOFUSER 3

//admin verify
    #define ADMIN_DB "data_file_admin.txt"
//func for admin verify

bool is_admin(char* role){
    FILE* fp = fopen(ADMIN_DB,"r");
    if(fp == NULL){
        printf("Error opening file\n");
        return false;
    }
    char line[100];
    while(fgets(line,100,fp) != NULL){
        if(unlikely(line[0] == '#')) continue;

        while(line[strlen(line)-1] == '\n' || (int)line[strlen(line)-1] == 13)
            line[strlen(line)-1] = '\0';

        if(strcmp(line,role) == 0){
            fclose(fp);
            return true;
        }
    }
    fclose(fp);
    return false;
}

void add_admin(char* role){
    FILE* fp = fopen(ADMIN_DB,"a");
    if(fp == NULL){
        printf("Error opening file\n");
        return;
    }
    fprintf(fp,"%s\n",role);
    fclose(fp);
    return;
}

void remove_admin(char* role){
    FILE* fp = fopen(ADMIN_DB,"r");
    if(fp == NULL){
        printf("Error opening file\n");
        return;
    }
    char temp[30] = ADMIN_DB;
    strcat(temp,".temp");
    FILE* fp2 = fopen(temp,"w");
    if(fp2 == NULL){
        printf("Error opening file\n");
        return;
    }
    char line[100];
    while(fgets(line,100,fp) != NULL){
        if(unlikely(line[0] == '#')) {
            fprintf(fp2,"%s",line);
            continue;
        }
        while(line[strlen(line)-1] == '\n' || (int)line[strlen(line)-1] == 13)//remove newline for compare
            line[strlen(line)-1] = '\0';
        if(strcmp(line,role) != 0){
            fprintf(fp2,"%s\n",line);
        }
    }
    fclose(fp);
    fclose(fp2);
    remove(ADMIN_DB);
    rename(temp,ADMIN_DB);
    return;
}
```

Searching

- Linear search algorithm is implemented
  - Binary search is not feasible
    - Sorting would not be so feasible as multiple column matching is required
- Case-insentive match is implemented



```c
struct Map* searchItems(struct inventory db, char* searchstr){
    struct Map* map = malloc(sizeof(struct Map) * (db.db.row_count+1));
    int k = 1;
    for (int i = 0; i < db.db.row_count; i++){
        map[k].key = i;
        if(strcasestr(db.row[i].product,searchstr) != NULL||
            strcasestr(db.row[i].brand,searchstr) != NULL||
            strcasestr(db.row[i].category,searchstr) != NULL||
            strcasestr(lto_string(db.row[i].barcode),searchstr) != NULL
        ){

            map[k].value = (void*)db.row[i].product;
            k++;
        }
    }
    int* k_star = malloc(sizeof(int));
    *k_star = --k;//decrement and assign to k star;don't use k-- as it will decrement after the assignment
    map[0].value = (void*)k_star;
    map[0].key = -1;
    return map;
}
```

strcasestr() : is the implementation of the case-insensitive match. It is provided in GNU implementation, the way to make this multi-platform will be discussed in General Functions section.

Sorting

Quick sort is being used, further information will be discussed in General Functions, Sorting Functions section.

D. Interface Implementation

General Interface Implementations

Choice Selector

```c
int choices_selecter(char * Items[],int items,char * welcome_messages){
  int choice = 0;
  do{
    Cls();

    if (welcome_messages !=NULL){
      printf("%s",welcome_messages);
  }
  printf("Select An Option:\n");
    int i;
    for (i=0; i<items;i++){
      printf("%d. %s\n",i+1,Items[i]);
    }
    printf("%d. Exit\n",i+1);
    printf(">");
    fflush_stdin();
    scanf("%d",&choice);
    if(choice < 1 || choice > items+1){
        printf("Invalid choice...press any key to retry....\n");
        fflush_stdin();
        getchar();
    }
  }while(choice < 1 || choice > items+1);
  return choice;

}
```

This is used on different Menu for selecting different portal. Simply modifying the code of this function can easily update the global interface.
This function also provides a **universal data validation**, prevent any outbound data or invalid input to be pass into the flow of the program. It **prevents** any **invalid choice** to be inputted to the **main loop**.

Prompt Item & Item Inputer

Prompt Item is used for UI, while Item Inputer format the data into different data type

```c
char* prompt_item(char* prompt,bool empty_allowed){
    char* item = malloc(sizeof(char) * 100);
    do{
        printf("%s",prompt);
        fflush_stdin();
        scanf("%[^\n]",item);//input until /n
        if(!empty_allowed&&strcmp(item,"") == 0){//check if temp is not empty
            printf("Invalid input, try again?(y/n)\n");
            char temp[100];
            fflush_stdin();
            scanf("%[^\n]",temp);
            if(strcmp(temp,"n") == 0){
                return NULL;
            }
        }
    }while(!empty_allowed&&strcmp(item,"") == 0);
    return item;
}
```

* Prompt Item provides a **universal data validation and verfication**, it handles **null input** as well as **invalid type**.

```c
typedef enum{
  INT=1,STRING=2,LONG=3,DOUBLE=4
}INPUT_TYPE;
bool item_inputer(char * varname,INPUT_TYPE type,void * item,bool empty_allowed){
  char output[1024] = "Please input the ";
  strcat(output,varname);
  strcat(output,": \n>");
  char* temp = prompt_item(output,empty_allowed);
  if(temp == NULL){
        return false;
  }
  switch(type){
    case INT:
      *((int*)item) = atoi(temp);
      break;
    case STRING:
      strcpy(item,temp);
      break;
    case LONG:
      *((long*)item) = atol(temp);
      break;
    case DOUBLE:
      *((double*)item) = atof(temp);
      break;
    default:
      return false;
  }
  return true;
}
```

* This also auto casting to corresponding data type

Lister

```c
void lister(
  void * db,
  struct Map* map,
   int row,
   char * lister_name,
   char * SortItems[],
   int NoSortItems,
   void (*page_printer)(void*,int,int,struct Map*) ,
    struct Map* (*sorter)(void *,int),void * (*showitem)(void *,int))
{
  int choice = -1;
    int page = 0;
    int page_size = SIZE_OF_PAGE;
    int total_pages = ceil((double)row / page_size);
    do{
    Cls();
    welcome_message(stdout);
    printf("%s list\n",lister_name);
    printf("0 exit\n");
    for(int i=0;i<NoSortItems*2;i+=2){
      printf("%d sort %s decending\n",i+1,SortItems[i/2]);
      printf("%d sort %s ascending\n",i+2,SortItems[i/2]);
    }
    if(page+1 == total_pages){
        (*page_printer)(db,page*page_size,row,map);
    }else{
        (*page_printer)(db,page*page_size,(page+1)*page_size,map);
    }
    //page control
    int current_page_size = page_size+NoSortItems*2;
    if(page+1 == total_pages){
        current_page_size = row - page*page_size+NoSortItems*2;
    }
    printf("%d next page\n", current_page_size+1);
    printf("%d previous page\n", current_page_size+2);
    printf("%d set page size\n", current_page_size+3);
    printf("%d/%d/%d(page size/page number/total)\n",page_size, page+1,total_pages);

    bool valid = true;
    do{
        valid = true;
        printf("Please input your choice\n>");
        fflush_stdin();
        scanf("%d",&choice);
         if(choice <=NoSortItems*2 && choice > 0){
            printf("sorting...\n");
            map = (*sorter)(db,choice);
        }else if(choice == current_page_size+1 ){
            if(page + 1 < total_pages){
                page++;
            }else{
                printf("Already at last page\n");
                valid = false;
            }
        }else if(choice == current_page_size+2){
            if(page > 0){
                page--;
            }else{
                printf("Already at first page\n");
                valid = false;
            }
        }else if(choice == current_page_size+3){
            printf("Enter page size: ");
            fflush_stdin();
            scanf("%d", &page_size);
            total_pages = ceil((double)row / page_size);
        }else if(choice > NoSortItems*2 && choice <= current_page_size){
            if(map == NULL){
                db = (*showitem)(db,choice - NoSortItems*2-1 + page_size*page);
            }else{
                db = (*showitem)(db,map[choice - NoSortItems*2-1 + page_size*page].key);
            }
        }else if(choice != 0){
            printf("Invalid choice\n");
            valid = false;
        }

    }while(!valid);
    }while(choice != 0);
    return;
}
```

Details explanation on next page.

Lister Allows to pass in different function for deep customization for each listing menus.

void * db	Pointer to database struct
struct * Map	Use as a dictionary map to map index to different position, use for sorting and searching
int row	Number of database rows
char * lister_name	For banner string
char * SortItems[]	String Array of Names of column of database that requires to show on the list
Int NoSortItems	Show the number of items of char * SortItems[]
void (*page_printer) (void*,int,int,struct Map*)	Function pointer of a function to print the database items
struct Map* (*sorter) (void *,int),void * (*showitem)(void *,int))	Function pointer of a function to sort the database

This function provides the general code and unified format, while still allows some customization on different lists. It also allows a easy modification on the UI.

Example of calling lister

```
struct inventory db = read_db_invt();
                char * SortItems[] = {
                    "Name",
                    "Price",
                    "Brand",
                    "Category",
                };
                lister(&db,NULL,db.db.row_count,"List of items",SortItems,4,print_page,sortItems,show_item);
```

```
void print_page(void * ddb, int cur, int end,struct Map* map){
    struct inventory db = *((struct inventory*)ddb);
    printf("%-5s%-15s%-15s%-10s%-10s%-10s%-10s\n","No.","Product","Brand","Category","Price","Stock","Barcode");
    for (int i = cur; i < end; i++)
    {
        if(map != NULL){
            int index = map[i].key;
            printf("%-5d%-15s%-15s%-10s%-10.2lf%-10d%-10ld\n",i+9,StringCompression(db.row[index].product,13),StringCompression(db.row[index].brand,13),StringCompression(db.row[index].category,8),db.row[index].price,db.row[index].stock,db.row[index].barcode);
        }
        else{
            printf("%-5d%-15s%-15s%-10s%-10.2lf%-10d%-10ld\n",i+9,StringCompression(db.row[i].product,13),StringCompression(db.row[i].brand,13),StringCompression(db.row[i].category,8),db.row[i].price,db.row[i].stock,db.row[i].barcode);
        }
    }
}
```

Formatting of String: “printf(“%-5s….,StringComperssion(...

%-5s: means reserving 5 column for the string it will takes at least 5 column, if the string is smaller than 5 ,it will expand to 5 column anyway.

StringCompression(char *,int size):

This function will cut string when exceed limit(e.g. (“Hello World”,5) → “He…”)

Show Item function

```c
void * show_item(void * ddb,int index){
    struct inventory db = *((struct inventory*)ddb);
    Cls();
    printf("Product: %s\n", db.row[index].product);
    printf("Catergory: %s\n", db.row[index].category);
    printf("Brand: %s\n", db.row[index].brand);
    printf("Price: $%lf\n", db.row[index].price);
    printf("Stock: %d\n", db.row[index].stock);
    printf("Barcode: %ld\n",db.row[index].barcode);
    printf("Press any key to return to the list\n");
    fflush_stdin();
    getchar();
    return ddb;
}
```

It allows customize action and display.

Sort Items

```
struct Map* sortItems(void * ddb, int sort){
    struct inventory db = *((struct inventory*)ddb);
    struct Map *map = malloc(sizeof(struct Map) * db.db.row_count -1);
    for (int i = 0; i < db.db.row_count-1; i++){
        map[i].key = i;

        switch(sort){
            case 1:
            case 2:
                map[i].value = (void*)db.row[i].product;
                break;

            case 3:
            case 4:;
                double price = db.row[i].price * 100;
                double * price_star = malloc(sizeof(long));
                *price_star = price;
                map[i].value = (void*)price_star;//presume there is no price contain  0.001
                break;
            case 5:
            case 6:
                map[i].value = (void*)db.row[i].brand;
                break;
            case 7:
            case 8:
                map[i].value = (void*)db.row[i].category;
                break;
        }
    }
    switch (sort){
        case 1:
        case 5:
        case 7:
            qsort(map, db.db.row_count-1, sizeof(struct Map), compare_decending_str);
            break;
        case 2:
        case 6:
        case 8:
            qsort(map, db.db.row_count-1, sizeof(struct Map), compare_ascending_str);
            break;
        case 3:
            qsort(map, db.db.row_count-1, sizeof(struct Map), compare_decending);
            break;
        case 4:
            qsort(map, db.db.row_count-1, sizeof(struct Map), compare_ascending);
            break;
    }
    map = realloc(map, sizeof(struct Map) * db.db.row_count);
    map[db.db.row_count-1].key = db.db.row_count-1;
    return map;
}

```

It is used to process different structures. It return a new struct map * for the lister to update.

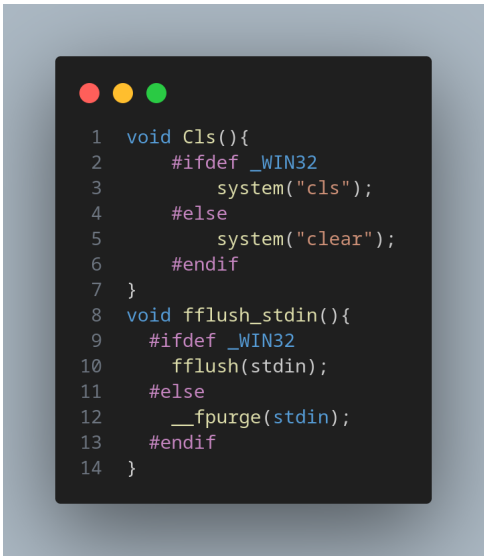
qsort() is being used, as it is a well polished multi-platform usable sorting function, it has a higher speed than some usual sorting algorithms(merge sort, bubble sort,etc). So, it is selected to be use in the program.

Further sorting things will be discussed later in this proposal.

D. General Functions

This section will include sorting details, searching details, Date&Time processing. Some general functions that included on section before might not be included, such as, StringCompression().

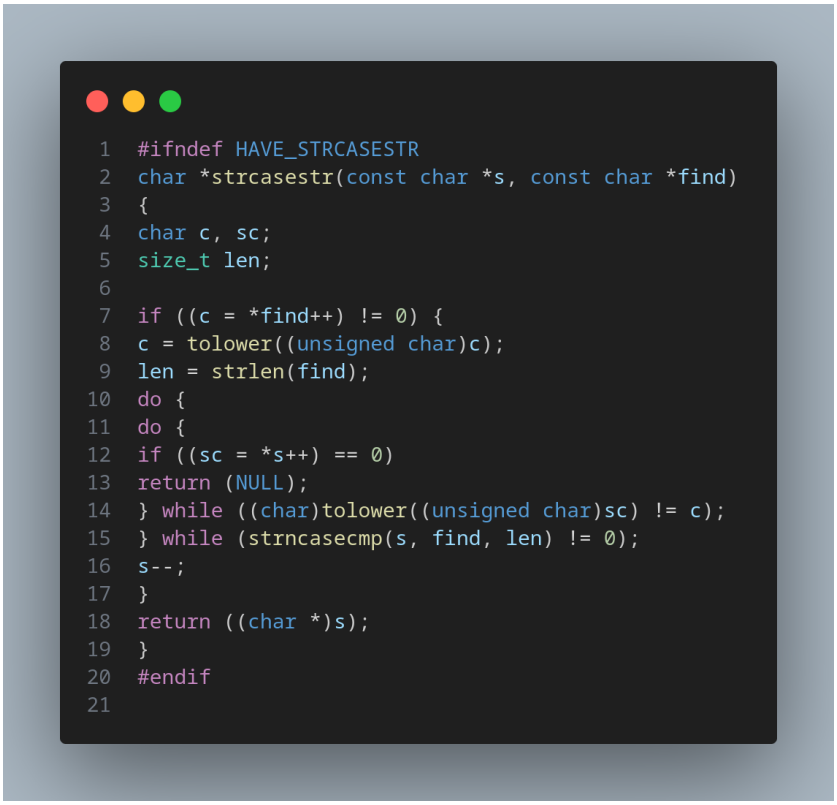
Functions that allow cross-platform support



```c
void Cls(){
    #ifdef _WIN32
        system("cls");
    #else
        system("clear");
    #endif
}
void fflush_stdin(){
  #ifdef _WIN32
    fflush(stdin);
  #else
    __fpurge(stdin);
  #endif
}
```

#ifdef _WIN32
content to run in windows
#else
content to run in linux (or other Unix-like system)
#endif

Cls(): clear screen
fflush_stdin(): clear stdin buffer



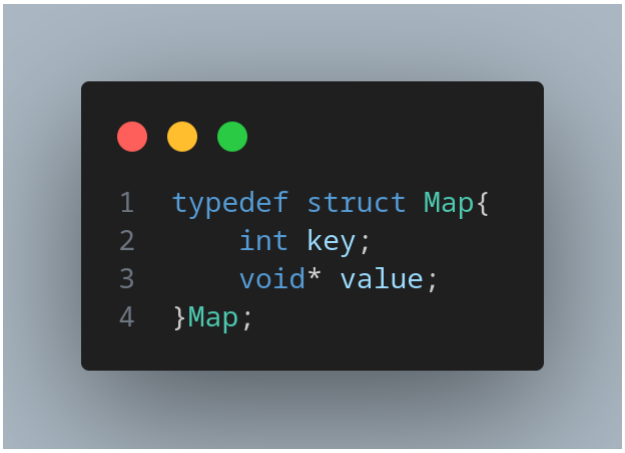
```
#ifndef HAVE_STRCASESTR
char *strcasestr(const char *s, const char *find)
{
char c, sc;
size_t len;

if ((c = *find++) != 0) {
c = tolower((unsigned char)c);
len = strlen(find);
do {
do {
if ((sc = *s++) == 0)
return (NULL);
} while ((char)tolower((unsigned char)sc) != c);
} while (strncasecmp(s, find, len) != 0);
s--;
}
return ((char *)s);
}
#endif

```

Case intensity match function, included on some system glibc, but some does not, so it is included with “#ifndef HAVE_STRCASESTR” to included when needed.

Sorting Functions

qsort is mentioned on the previous section, but the function that works with it is included. Under this it is:

```c
int compare_decending_str(const void *a, const void *b){
    if((*(struct Map *)b).value == NULL){//To prevent null pointer cause error
        return -1;
    }else if((*(struct Map *)a).value == NULL){
        return 1;

    }
    return strcmp((*(struct Map *)b).value,(*(struct Map *)a).value);
}
int compare_ascending_str(const void *a, const void *b){
    if((*(struct Map *)b).value == NULL){
        return -1;
    }else if((*(struct Map *)a).value == NULL){
        return 1;

    }
    return strcmp((*(struct Map *)a).value,(*(struct Map *)b).value);
}
int compare_decending(const void *a, const void *b){
    if((*(struct Map *)b).value == NULL){//To prevent null pointer cause error
        return -1;
    }else if((*(struct Map *)a).value == NULL){
        return 1;

    }
    return (*(struct Map *)b).value-(*(struct Map *)a).value;
}
int compare_ascending(const void *a, const void *b){
    if((*(struct Map *)b).value == NULL){
        return -1;
    }else if((*(struct Map *)a).value == NULL){
        return 1;

    }
    return (*(struct Map *)a).value -(*(struct Map *)b).value;
}
```



```c
typedef struct Map{
    int key;
    void* value;
}Map;
```

Struct Map is used to map value after sort without really moving the database content.

Why qsort()(Quick sort)?

1. Quick sort is blazing fast
2. stable sort is not required

With this two reason, the quick sort algorithm is suitable for this program.
Also, qsort() is provided by the C standard library, so that the function is implemented and optimized in different OS, ensuring high performance on different platform.

Date and Time processing

```c
typedef struct Date{
    int day;
    int month;
    int year;
}Date;
```

```c
typedef struct Time{
    int hour;
    int minute;
    int second;
}Time;
```

```c
struct Date convert_to_date(char* date){
    struct Date d;
    char* token = strtok(date, "-");
    d.year = atoi(token);
    token = strtok(NULL, "-");
    d.month = atoi(token);
    token = strtok(NULL, "-");
    d.day = atoi(token);
    return d;
}
```

10-20-2023 →Month:10,Day:20,Year:2023

```c
struct Time convert_to_time(char* time){
    struct Time t;
    char* token = strtok(time, ":");
    t.hour = atoi(token);
    token = strtok(NULL, ":");
    t.minute = atoi(token);
    token = strtok(NULL, ":");
    t.second = atoi(token);
    return t;
}
```

10:20:30 → Hour:10,Min:20,Sec:30

```c
struct Date get_date(){
    struct Date d;
    time_t rawtime;
    struct tm * timeinfo;
    time(&rawtime);
    timeinfo = localtime(&rawtime);
    d.day = timeinfo->tm_mday;
    d.month = timeinfo->tm_mon + 1;
    d.year = timeinfo->tm_year + 1900;
    return d;
}
```

```c
struct Time convert_to_time(char* time){
    struct Time t;
    char* token = strtok(time, ":");
    t.hour = atoi(token);
    token = strtok(NULL, ":");
    t.minute = atoi(token);
    token = strtok(NULL, ":");
    t.second = atoi(token);
    return t;
}
```

Get current Date and Time respectively

E. Optimization

```
#ifndef likely
#define likely(x) __builtin_expect(!!(x), 1)
#define unlikely(x) __builtin_expect(!!(x), 0)
#endif
```

This likely, unlikely is useful for cpu to overhead and predict the branch that it is going to jump, which can improve performance, as modern cpu uses a technique called “Branch predictor”, telling which branch is likely to be true will improve the percentage of correct prediction.

Example

```
while(fgets(line,100,fp) != NULL){
        if(unlikely(line[0] == '#')) continue;

        while(line[strlen(line)-1] == '\n' || (int)line[strlen(line)-1] == 13)
            line[strlen(line)-1] = '\0';

        if(strcmp(line,role) == 0){
            fclose(fp);
            return true;
        }
    }
```

As it has a significant less commented line than non-commented line, so using unlikely can improve performance.

F. Program listing

Link:https://tommygod.ddns.net/gitea/stmctommyau/sba

Actual Code will be listed at the end of the file. The extra:program listing section.

4. Further Improvements

Firstly, better UI can be use. For example. Qt library can be used to create a GUI interface, as the program leave easy space to change UI, it would not requires a lot of rewrite to implement GUI.

Secondly, encryption can be implemented. We can encrypt data files, to prevent non-admin user getting to peek data on some kind of file system leakage.

Thirdly, the front-end and back-end can be split. With splitting them multiple cashiers can be set up with in sync data. As the program structure splits front-end and back-end code, it would not be a hard job to migrate them to server and client. Also, multi-platform support also benefits this job.

Last but not least, auto analysis export can be implement for admin to learn trend more easily. Currently, admin can only see hot items, by sorting transactions, or getting other information from sorting databases.

5. Conclusion

To conclude, this program still requires hardware support and real life testing to make it deplorable to production environment. But, the program left spaces and the structure allows easy extension to meet needs to fix problems and add features. It would not be a hard work to make it deploy on any POS machines. It has multi-platform feature which allows it to be deploy on any devices, and to support more scenario. This program demonstrated how a POS system can be implemented and to be able to deployed. It shows that POS system is suitable to be deployed in Student Union’s shop.

Extra: Program Listing

main.c

```
//include std lib if they haven’t been included
#ifndef STD_LIB_H
#define STD_LIB_H
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <math.h>
#include <time.h>
#include <ctype.h>
#include <stddef.h>
#endif
#ifndef CUSTOM_H
#define CUSTOM_H
#include "database.h"
#endif // !CUSTOM_H

#include "admin_user.h"
#include "normal_user.h"
#ifndef Utils
#define Utils
#include "sorting.h"
#include "utils.h"
#endif // !Utils
```

```
int main(){
    bool check;
    do{
        Cls();
        //print welcome message
        //prompt weather user is admin or normal user
        //then redirect them to the corisponding menu
        int choice = choices_selecter((char*[]){
            "Admin",
                "Normal User"
        },2,welcome_message(NULL));
        switch (choice)
        {
        case 1://admin
            admin_menu();//refers to admin_user.h
            check = true;
            break;
        case 2://normal user
            normal_menu();//refers to normal_user.h
            check = true;
            break;
        default://Exit
            check = false;
            break;
        }

        fflush_stdin();
    } while (check);

    return 0;
}
```

database.h

```
//include std lib if they havent been included
#ifndef STD_LIB_H
#define STD_LIB_H
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<stdbool.h>
#include<math.h>
#include<time.h>
#include<ctype.h>
#endif
#ifndef DNT_H
#define DNT_H
#include "dateNtime.h"
#endif // !DNT_H
//linked list
#ifndef likely
#define likely(x) __builtin_expect(!!(x), 1)
#define unlikely(x) __builtin_expect(!!(x), 0)
#endif
typedef struct linkedlist{
    struct linkedlist* next;
    struct linkedlist* prev;
    void* data;
}linkedlist;
//linked list functions
int sizeofLinkedlist(struct linkedlist* list){
    linkedlist* start = list;
    int size = 0;
    linkedlist* current = list;
    while (current != NULL){
        size++;
```

```
        current = current->next;
    }
    current = start;
    while(current->prev != NULL){
        size++;
        current = current->prev;
    }
    return size;
}
struct linkedlist* getLinkedList(struct linkedlist* list,int pos){
    linkedlist* cur = list;
    while(cur->prev != NULL){
        cur = cur->prev;
    }
    struct linkedlist* start = cur;
    for(int i=0;i<pos;i++){
        cur = cur->next;
        if(cur == NULL){
            return start;//return start if pos is out of bounds
        }
    }
    return cur;
}
typedef struct basic_db{
    int row_count;
    bool init_status;//not really used can be useful
    //array of struct of row
}basic_db;

//inventory
#define INVENTORY_DB "data_file_inventory.txt"
typedef struct inventory{
    struct basic_db db;
    struct inventory_row* row;

}inventory;
typedef struct inventory_row{
    char category[100];
```

```
    char brand[100];
    char product[100];
    double price;
    int stock;
    long barcode;
    bool isdeleted;//common for all rows,default is false
}inventory_row;
typedef enum  {
    category = 1, brand = 2, product = 3, price_inv = 4, stock = 5, barcode_inv = 6
 }INVENTORY;
#define ENDOFINVENTORY 6
//transaction
#define TRANSACTION_DB "data_file_transaction.txt"
typedef struct transaction{
    struct basic_db db;
    struct transaction_row* row;

}transaction;
typedef struct transaction_row{
    struct Date date;
    struct Time time;
    long id;
    double price;
    int quantity;
    long barcode;
    bool isdeleted;//common for all rows,default is false
}transaction_row;
typedef enum {
    date = 1, TIME = 2, id_tran = 3, price_tran = 4, quantity = 5, barcode_tran = 6
 }TRANSACTION;
#define ENDOFTRANSACTION 6
//user
#define USER_DB "data_file_user.txt"
typedef struct user{
    struct basic_db db;
    struct user_row* row;
}user;
typedef struct user_row{
```

```
    char name[100];
    char role[100];
    long id;
    bool isdeleted;//common for all rows,default is false
}user_row;

typedef enum {
    name = 1, role = 2, id_user = 3
 }USER;
#define ENDOFUSER 3

//admin verify
#define ADMIN_DB "data_file_admin.txt"
//func for admin verify

bool is_admin(char* role){
    FILE* fp = fopen(ADMIN_DB,"r");
    if(fp == NULL){
        printf("Error opening file\n");
        return false;
    }
    char line[100];
    while(fgets(line,100,fp) != NULL){
        if(unlikely(line[0] == '#')) continue;

        while(line[strlen(line)-1] == '\n' || (int)line[strlen(line)-1] == 13)
            line[strlen(line)-1] = '\0';

        if(strcmp(line,role) == 0){
            fclose(fp);
            return true;
        }
    }
    fclose(fp);
    return false;
}

void add_admin(char* role){
```

```
    FILE* fp = fopen(ADMIN_DB,"a");
    if(fp == NULL){
        printf("Error opening file\n");
        return;
    }
    fprintf(fp,"%s\n",role);
    fclose(fp);
    return;
}

void remove_admin(char* role){
    FILE* fp = fopen(ADMIN_DB,"r");
    if(fp == NULL){
        printf("Error opening file\n");
        return;
    }
    char temp[30] = ADMIN_DB;
    strcat(temp,".temp");
    FILE* fp2 = fopen(temp,"w");
    if(fp2 == NULL){
        printf("Error opening file\n");
        return;
    }
    char line[100];
    while(fgets(line,100,fp) != NULL){
        if(unlikely(line[0] == '#')) {
            fprintf(fp2,"%s",line);
            continue;
        }
        while(line[strlen(line)-1] == '\n' || (int)line[strlen(line)-1] == 13)//remove
newline for compare
            line[strlen(line)-1] = '\0';
        if(strcmp(line,role) != 0){
            fprintf(fp2,"%s\n",line);
        }
    }
    fclose(fp);
    fclose(fp2);
```

```
    remove(ADMIN_DB);
    rename(temp,ADMIN_DB);
    return;
}
//list of db func
int basic_get_row_count(int end, FILE *fp){
    fseek(fp, 0, SEEK_SET);//prevent pointer on wrong position
    //get row count
    int row_count = 0;
    int colmun_count = 0;
    while(!feof(fp)&&!ferror(fp)){
        char buffer[100];
        fgets(buffer, sizeof(buffer),fp);
        if(unlikely(buffer[0] == '#' || buffer[0] == '\0')){//catch comment line and ignore
            continue;
        }
        colmun_count++;
        if(colmun_count == end){
            row_count++;
            colmun_count = 0;
        }
    }
    return row_count;
}

struct inventory read_db_invt(){//please open file in read mode
    FILE* fp = fopen(INVENTORY_DB, "r");
    struct inventory db;

    //gets the number of rows in the file
    int row_count = basic_get_row_count(ENDOFINVENTORY,fp);
    db.row = (struct inventory_row*)malloc(row_count * sizeof(struct
inventory_row));
    db.db.row_count = row_count;

    fseek(fp,0,SEEK_SET);//reset fp to the beginning of the file

    //read data
```

```
for(int i=0;i<row_count;i++){
   db.row[i].isdeleted = false;
   for(INVENTORY j=category;j<=ENDOFINVENTORY;j++){
      char buffer[100];
      fgets(buffer, sizeof(buffer),fp);
      if(unlikely(buffer[0] == '#')){//catch comment line and ignore
         j--;//decrement j to prevent skipping next column
      }else{
         buffer[strlen(buffer)] = '\0';
         while(buffer[strlen(buffer)-1] == '\n' || (int)buffer[strlen(buffer)-1] == 13){
            buffer[strlen(buffer)-1] = '\0';

         }

         switch(j){
            case category:
               strcpy(db.row[i].category,buffer);
               break;
            case brand:
               strcpy(db.row[i].brand,buffer);
               break;
            case product:
               strcpy(db.row[i].product,buffer);
               break;
            case price_inv:
               db.row[i].price = atof(buffer);
               break;
            case stock:
               db.row[i].stock = atoi(buffer);
               break;
            case barcode_inv:
               db.row[i].barcode = atol(buffer);
               break;

         }
      }

   }
```

```
    }
    if(ferror(fp)){
        printf("Error in reading file\n");
        db.db.init_status = false;
        return db;
    }
    fclose(fp);
    db.db.init_status = true;
    return db;
}
struct transaction read_db_tran(){
    FILE* fp = fopen(TRANSACTION_DB, "r");
    struct transaction db;

    //gets the number of rows in the file
    int row_count = basic_get_row_count(ENDOFTRANSACTION,fp);
    db.row = (struct transaction_row*)malloc(row_count * sizeof(struct
transaction_row));
    db.db.row_count = row_count;

    fseek(fp,0,SEEK_SET);//reset fp to the beginning of the file

    //read data
    for(int i=0;i<row_count;i++){
        db.row[i].isdeleted = false;
        for(TRANSACTION j=date;j<=ENDOFTRANSACTION;j++){
            char buffer[100];
            fgets(buffer, sizeof(buffer),fp);
            if(unlikely(buffer[0] == '#')){//catch comment line and ignore
                j--;//decrement j to prevent skipping next column
            }else{
                buffer[strlen(buffer)] = '\0';
                while(buffer[strlen(buffer)-1] == '\n' || (int)buffer[strlen(buffer)-1] == 13){
                    buffer[strlen(buffer)-1] = '\0';

                }

                switch(j){
```

```
                case date:
                    db.row[i].date = convert_to_date(buffer);
                    break;
                case TIME:
                    db.row[i].time = convert_to_time(buffer);
                    break;
                case id_tran:
                    db.row[i].id = atol(buffer);
                    break;
                case price_tran:
                    db.row[i].price = atof(buffer);
                    break;
                case quantity:
                    db.row[i].quantity = atoi(buffer);
                    break;
                case barcode_tran:
                    db.row[i].barcode = atol(buffer);
                    break;
            }
        }

    }
  }
  if(ferror(fp)){
    printf("Error in reading file\n");
    db.db.init_status = false;
    return db;
  }
  fclose(fp);
  db.db.init_status = true;
  return db;
}
struct user read_db_user(){
  FILE* fp = fopen(USER_DB,"r");
  struct user db;

  //gets the number of rows in the file
  int row_count = basic_get_row_count(ENDOFUSER,fp);
```

```
db.row = (struct user_row*)malloc(row_count * sizeof(struct user_row));
db.db.row_count = row_count;

fseek(fp,0,SEEK_SET);//reset fp to the beginning of the file

//read data
for(int i=0;i<row_count;i++){
   db.row[i].isdeleted = false;
   for(USER j=name;j<=ENDOFUSER;j++){
      char buffer[100];
      fgets(buffer, sizeof(buffer),fp);
      if(unlikely(buffer[0] == '#')){//catch comment line and ignore
         j--;//decrement j to prevent skipping next column
      }else{
         buffer[strlen(buffer)] = '\0';//prevent any garbage value
         while(buffer[strlen(buffer)-1] == '\n' || (int)buffer[strlen(buffer)-1] == 13){
            buffer[strlen(buffer)-1] = '\0';

         }

         switch(j){
            case name:
               strcpy(db.row[i].name,buffer);
               break;
            case role:
               strcpy(db.row[i].role,buffer);
               break;
            case id_user:
               db.row[i].id = atol(buffer);
               break;
         }
      }

   }
}
if(ferror(fp)){
   printf("Error in reading file\n");
   db.db.init_status = false;
```

```
        return db;
    }
    fclose(fp);
    db.db.init_status = true;
    return db;
}

struct linkedlist* role_list_db(){
    struct user db = read_db_user();
    struct linkedlist* list = (struct linkedlist*)malloc(sizeof(struct linkedlist));
    //put unqie user role into linked list in acending order
    struct linkedlist* current = list;
    //initial the first node
    current->data = db.row[0].role;
    current->next = NULL;
    current->prev = NULL;
    for(int i=1;i<db.db.row_count;i++){
        char* role = db.row[i].role;
        int diff = strcmp(current->data,role);
        if(diff == 0){
            continue;
        }else if(diff > 0){
            while(diff > 0){
                if(diff == 0){
                    break;
                }else if(diff < 0){
                    //insert before current
                    struct linkedlist* new = (struct linkedlist*)malloc(sizeof(struct
linkedlist));
                    new->data = role;
                    new->next = current;
                    new->prev = current->prev;
                    current->prev->next = new;
                    current->prev = new;
                    break;
                }else{
                    if(current->next == NULL){
                        //insert after current
```

```
                struct linkedlist* new = (struct linkedlist*)malloc(sizeof(struct
linkedlist));
                new->data = role;
                new->next = NULL;
                new->prev = current;
                current->next = new;
                break;
            }else{
                current = current->next;
                diff = strcmp(current->data,role);
            }
        }
    }else{
        while(diff < 0){
            if(diff == 0){
                break;
            }else if(diff > 0){
                //insert after current
                struct linkedlist* new = (struct linkedlist*)malloc(sizeof(struct
linkedlist));
                new->data = role;
                new->next = current;
                new->prev = current->prev;
                current->prev->next = new;
                current->prev = new;
                break;
            }else{
                if(current->prev == NULL){
                    //insert before current
                    struct linkedlist* new = (struct linkedlist*)malloc(sizeof(struct
linkedlist));
                    new->data = role;
                    new->next = current;
                    new->prev = NULL;
                    current->prev = new;
                    break;
                }else{
```

```
                current = current->prev;
                diff = strcmp(current->data,role);
            }
        }
    }
  }
  return list;
}

bool update_db_invt(struct inventory invt){
  FILE* fpR = fopen(INVENTORY_DB,"r");
  char temp[30] = INVENTORY_DB;
  strcat(temp,".temp");
  FILE* fpW = fopen(temp,"w");
  if(fpR == NULL || fpW == NULL){
    printf("Error in opening file\n");
    return false;
  }
  for(int i=0;i<invt.db.row_count;i++){
    if(invt.row[i].barcode == -1024 || invt.row[i].isdeleted == true)//skip create new
row
      continue;
    for(INVENTORY j=category;j<=ENDOFINVENTORY;j++){
      char buffer[100];
      do{
        if(unlikely(feof(fpR))){
          break;
        }
        fgets(buffer, sizeof(buffer),fpR);
        if(buffer[0] == '#'){//catch comment line and print back to new file
          fputs(buffer,fpW);
        }
      }while(buffer[0] == '#');
      switch(j){
        case category:
          fprintf(fpW,"%s",invt.row[i].category);
          break;
```

```
            case brand:
                fprintf(fpW,"%s",invt.row[i].brand);
                break;
            case product:
                fprintf(fpW,"%s",invt.row[i].product);
                break;
            case price_inv:
                fprintf(fpW,"%.1f",invt.row[i].price);
                break;
            case stock:
                fprintf(fpW,"%d",invt.row[i].stock);
                break;
            case barcode_inv:
                fprintf(fpW,"%ld",invt.row[i].barcode);
                break;
        }
        fprintf(fpW,"\n");
      }
   }
   //close file and replace old file with new file
   fclose(fpR);
   fclose(fpW);
   remove(INVENTORY_DB);
   rename(temp,INVENTORY_DB);
   return true;

}

bool update_db_tran(struct transaction tran){
   FILE* fpR = fopen(TRANSACTION_DB,"r");
   char temp[30] = TRANSACTION_DB;
   strcat(temp,".temp");
   FILE* fpW = fopen(temp,"w");
   if(fpR == NULL || fpW == NULL){
      printf("Error in opening file\n");
      return false;
   }
   for(int i=0;i<tran.db.row_count;i++){
```

```
        if(tran.row[i].isdeleted == true){
            continue;
        }
        for(TRANSACTION j=date;j<=ENDOFTRANSACTION;j++){
            char buffer[100];
            do{
                if(unlikely(feof(fpR))){
                    break;
                }
                fgets(buffer, sizeof(buffer),fpR);
                if(buffer[0] == '#'){//catch comment line and print back to new file
                    fputs(buffer,fpW);
                }
            }while(buffer[0] == '#');
            switch(j){
                case date:
                    fprintf(fpW,"%d-%02d-
%02d",tran.row[i].date.year,tran.row[i].date.month,tran.row[i].date.day);
                    break;
                case TIME:
                    fprintf(fpW,"%02d:%02d:
%02d",tran.row[i].time.hour,tran.row[i].time.minute,tran.row[i].time.second);
                    break;
                case id_tran:
                    fprintf(fpW,"%ld",tran.row[i].id);
                    break;
                case price_tran:
                    fprintf(fpW,"%.1f",tran.row[i].price);
                    break;
                case quantity:
                    fprintf(fpW,"%d",tran.row[i].quantity);
                    break;
                case barcode_tran:
                    fprintf(fpW,"%ld",tran.row[i].barcode);
                    break;
            }
            fprintf(fpW,"\n");
        }
```

```
    }
    //close file and replace old file with new file
    fclose(fpR);
    fclose(fpW);
    remove(TRANSACTION_DB);
    rename(temp,TRANSACTION_DB);
    return true;
}

bool update_db_user(struct user user){
    FILE* fpR = fopen(USER_DB,"r");
    char temp[30] = USER_DB;
    strcat(temp,".temp");
    FILE* fpW = fopen(temp,"w");
    if(fpR == NULL || fpW == NULL){
        printf("Error in opening file\n");
        return false;
    }
    for(int i=0;i<user.db.row_count;i++){
        if(user.row[i].isdeleted == true){
            continue;
        }
        for(USER j=name;j<=ENDOFUSER;j++){
            char buffer[100];
            do{
                if(unlikely(feof(fpR))){
                    break;
                }
                fgets(buffer, sizeof(buffer),fpR);
                if(buffer[0] == '#'){//catch comment line and print back to new file
                    fputs(buffer,fpW);
                }
            }while(buffer[0] == '#');
            switch(j){
                case name:
                    fprintf(fpW,"%s",user.row[i].name);
                    break;
                case role:
```

```
                fprintf(fpW,"%s",user.row[i].role);
                break;
            case id_user:
                fprintf(fpW,"%ld",user.row[i].id);
                break;
        }
        fprintf(fpW,"\n");
      }
   }
   //close file and replace old file with new file
   fclose(fpR);
   fclose(fpW);
   remove(USER_DB);
   rename(temp,USER_DB);
   return true;
}

bool append_inventory(struct inventory_row* row){
   FILE* fp = fopen(INVENTORY_DB,"a");
   if(fp == NULL){
      printf("Error in opening file\n");
      return false;
   }
   fprintf(fp,"%s\n",row->category);
   fprintf(fp,"%s\n",row->brand);
   fprintf(fp,"%s\n",row->product);
   fprintf(fp,"%.1f\n",row->price);
   fprintf(fp,"%d\n",row->stock);
   fprintf(fp,"%ld\n",row->barcode);
   fclose(fp);
   return true;
}

bool append_transaction_db(struct transaction_row* row){
   FILE* fp = fopen(TRANSACTION_DB,"a");
   if(fp == NULL){
      printf("Error in opening file\n");
      return false;
```

```
    }
    fprintf(fp,"%04d-%02d-%02d\n",row->date.year,row->date.month,row->date.day);
    fprintf(fp,"%02d:%02d:%02d\n",row->time.hour,row->time.minute,row-
>time.second);
    fprintf(fp,"%ld\n",row->id);
    fprintf(fp,"%.1f\n",row->price);
    fprintf(fp,"%d\n",row->quantity);
    fprintf(fp,"%ld\n",row->barcode);
    fclose(fp);
    return true;
}

bool append_user(struct user_row* row){
    FILE* fp = fopen(USER_DB,"a");
    if(fp == NULL){
        printf("Error in opening file\n");
        return false;
    }
    fprintf(fp,"%s\n",row->name);
    fprintf(fp,"%s\n",row->role);
    fprintf(fp,"%ld\n",row->id);
    fclose(fp);
    return true;
}

//checkout db support,

typedef struct cart{//linked list
    struct inventory_row* row;//pointer to the row
    int quantity;//quantity of the item
    struct cart* next;
}cart;



bool append_transaction(struct cart* cart,struct user_row* user){
    FILE* fp = fopen(TRANSACTION_DB,"a");
    if(fp == NULL){
        printf("Error in opening file\n");
```

```
        return false;
    }

    struct Date date = get_date();
    struct Time time = get_time();
    struct cart* temp = cart;
    if(temp == NULL)
        return false;

    do{

        fprintf(fp,"%d-%02d-%02d\n",date.year,date.month,date.day);
        fprintf(fp,"%02d:%02d:%02d\n",time.hour,time.minute,time.second);
        fprintf(fp,"%ld\n",user->id);
        fprintf(fp,"%f\n",temp->row->price);
        fprintf(fp,"%d\n",temp->quantity);
        fprintf(fp,"%ld\n",temp->row->barcode);
        if(temp->next == NULL)
            break;
        temp = temp->next;
    }while(temp->next != NULL);//this while condition just for safety,as it should
already break in the if statement
    fclose(fp);
    return true;
}

bool update_stock_N_checkout(struct cart* cart,struct user_row* user){
    struct inventory invt = read_db_invt();
    struct cart* temp = cart;
    if(temp == NULL)
        return false;
    do{
        for(int i=0;i<invt.db.row_count;i++){

            if(invt.row[i].barcode == temp->row->barcode){
                invt.row[i].stock -= temp->quantity;
                break;
            }
```

```
        }
        if(temp->next == NULL)
            break;
        temp = temp->next;
    }while(temp->next != NULL);//this while condition just for safety,as it should
already break in the if statement
    if(!append_transaction(cart,user)){
        return false;
    }
    if(!update_db_invt(invt)){
        return false;
    }
    return true;
}

//user
struct user_row* get_user(long userid){
    struct user_row* user = (struct user_row*)malloc(sizeof(struct user_row));
    FILE* fp = fopen(USER_DB,"r");
    if(fp == NULL){
        printf("Error in opening file\n");
        return NULL;
    }
    char buffer[100];

    //gets the number of rows in the file
    int row_count = basic_get_row_count(ENDOFUSER,fp);

    fseek(fp,0,SEEK_SET);//reset fp to the beginning of the file

    //read data
    for(int i=0;i<row_count;i++){
        for(USER j=name;j<=ENDOFUSER;j++){
            char buffer[100];
            fgets(buffer, sizeof(buffer),fp);
            if(buffer[0] == '#'){//catch comment line and ignore
                j--;//decrement j to prevent skipping next column
            }else{
```

```
            buffer[strlen(buffer)] = '\0';
            while(buffer[strlen(buffer)-1] == '\n' || (int)buffer[strlen(buffer)-1] == 13){
               buffer[strlen(buffer)-1] = '\0';
            }

            switch(j){
               case name:
                  strcpy(user->name,buffer);
                  break;
               case role:
                  strcpy(user->role,buffer);
                  break;
               case id_user:
                  user->id = atol(buffer);
                  if(user->id == userid){
                     fclose(fp);
                     return user;
                  }
                  break;
            }
         }

      }
   }
   fclose(fp);
   return NULL;
}
```

dateNtime.h

```
//include std lib if they havent been included
#ifndef STD_LIB_H
#define STD_LIB_H
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<stdbool.h>
#include<math.h>
#include<time.h>
#include<ctype.h>
#endif
//DATE
typedef struct Date{
   int day;
   int month;
   int year;
}Date;


struct Date convert_to_date(char* date){
   struct Date d;
   char* token = strtok(date, "-");
   d.year = atoi(token);
   token = strtok(NULL, "-");
   d.month = atoi(token);
   token = strtok(NULL, "-");
   d.day = atoi(token);
   return d;
}

struct Date get_date(){
   struct Date d;
```

```
    time_t rawtime;
    struct tm * timeinfo;
    time(&rawtime);
    timeinfo = localtime(&rawtime);
    d.day = timeinfo->tm_mday;
    d.month = timeinfo->tm_mon + 1;
    d.year = timeinfo->tm_year + 1900;
    return d;
}

//TIME

typedef struct Time{
    int hour;
    int minute;
    int second;
}Time;

struct Time convert_to_time(char* time){
    struct Time t;
    char* token = strtok(time, ":");
    t.hour = atoi(token);
    token = strtok(NULL, ":");
    t.minute = atoi(token);
    token = strtok(NULL, ":");
    t.second = atoi(token);
    return t;
}

struct Time get_time(){
    struct Time t;
    time_t rawtime;
    struct tm * timeinfo;
    time(&rawtime);
    timeinfo = localtime(&rawtime);
    t.hour = timeinfo->tm_hour;
    t.minute = timeinfo->tm_min;
    t.second = timeinfo->tm_sec;
```

```
    return t;
}
```

admin_user.h

```
//include std lib if they havent been included
#ifndef STD_LIB_H
#define STD_LIB_H
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <math.h>
#include <time.h>
#include <ctype.h>
#endif

#ifndef Utils
#define Utils
#include "sorting.h"
#include "utils.h"
#endif // !Utils
//list of functions
#include "inv_control.h"
#include "tran_control.h"
#include "user_control.h"
#include "role_control.h"
struct user_row *prompt_user();
int admin_menu_user_choices(char* name,char* role);

void admin_menu(){
    Cls();
    welcome_message(stdout);
    //the selection menu
    struct user_row *user = prompt_user();
```

```
    if(user == NULL){//invalid user
        return;
    }
    char * Items[] = {
        "Inventory control",
        "Transaction control",
        "User control",
        "Role control",
    };
    char welcome[256];
    sprintf(welcome,"welcome %s(%s)\n",user->name,user->role);
    bool exit = false;
    do{

        switch (choices_selecter(Items,4,welcome))
        {
            case 1:;//action with inventory
                inv_control();
                break;
            case 2://action with transction
                tran_control();
                break;
            case 3://action with user
                user_control();
                break;
            case 4://role management
                role_control();
                break;
            case 5://Exit
                exit = true;
                break;
            default://invalid input ,should not happen,as it is already catch in above
function
                printf("Invalid choice\n");
                break;
        }
    }while(!exit);
```

```
    return;
}

//func for main for admin
struct user_row *prompt_user(){
    long user_id = 0;
    printf("Please tap your card(student card, staff card,etc)(or input the id)(0 to
cancel)\n>");
    scanf("%ld", &user_id);
    if(user_id == 0){
        printf("cancelled\n");
        printf("press any key to continue\n");
        fflush_stdin();
        getchar();
        return NULL;
    }
    struct user_row *user = get_user(user_id);
    Cls();
    if(user == NULL){
        printf("User not found\n");
        printf("press any key to continue\n");
        fflush_stdin();
        getchar();
        return NULL;
    }else if(!is_admin(user->role)){
        printf("You aren't an admin\n");
        printf("press any key to continue\n");
        fflush_stdin();
        getchar();
        return NULL;
    }else{
            printf("Welcome %s %s\n", user->name , user->role);
        printf("press any key to continue\n");
        fflush_stdin();
        getchar();
    }
    return user;
}
```

sorting.h

```
typedef struct Map{
   int key;
   void* value;
}Map;

int compare_decending_str(const void *a, const void *b){
   if((*(struct Map *)b).value == NULL){//To prevent null pointer cause error
      return -1;
   }else if((*(struct Map *)a).value == NULL){
      return 1;

   }
   return strcmp((*(struct Map *)b).value,(*(struct Map *)a).value);
}
int compare_ascending_str(const void *a, const void *b){
   if((*(struct Map *)b).value == NULL){
      return -1;
   }else if((*(struct Map *)a).value == NULL){
      return 1;

   }
   return strcmp((*(struct Map *)a).value,(*(struct Map *)b).value);
}
int compare_decending(const void *a, const void *b){
   if((*(struct Map *)b).value == NULL){//To prevent null pointer cause error
      return -1;
   }else if((*(struct Map *)a).value == NULL){
      return 1;

   }
   return (*(struct Map *)b).value-(*(struct Map *)a).value;
}
```

```
int compare_ascending(const void *a, const void *b){
   if((*(struct Map *)b).value == NULL){
      return -1;
   }else if((*(struct Map *)a).value == NULL){
      return 1;

   }
   return (*(struct Map *)a).value -(*(struct Map *)b).value;
}
```

utils.h

```
#ifndef STD_LIB_H
#define STD_LIB_H
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<stdbool.h>
#include<math.h>
#include<time.h>
#include<ctype.h>
#include <stddef.h>
#endif
#ifndef _WIN32
//for fflush_stdin
#include <stdio_ext.h>
#endif
#define SIZE_OF_PAGE 20
#ifndef likely
#define likely(x) __builtin_expect(!!(x), 1)
#define unlikely(x) __builtin_expect(!!(x), 0)
#endif
void Cls(){
   #ifdef _WIN32
      system("cls");
```

```
   #else
      system("clear");
   #endif
}
void fflush_stdin(){
 #ifdef _WIN32
   fflush(stdin);
 #else
   __fpurge(stdin);
 #endif
}
char * lto_string(long num){
char * str = malloc(sizeof(char) * 100);
sprintf(str,"%ld",num);
return str;
}

char * welcome_message(void * ptr){ //cross compatible

   if(!ptr){//if fp is NULL
     return "><Welcome to the Student Union POS system><\n";
   }
   if(((FILE*)ptr)!=stdout){// if ptr a string or stdout
     sprintf(ptr,"><Welcome to the Student Union POS system><\n");//welcome
message
   }else{
     fprintf(ptr,"><Welcome to the Student Union POS system><\n");//welcome
message
   }
   return (char*)NULL;
}
int choices_selecter(char * Items[],int items,char * welcome_messages){
 int choice = 0;
 do{
      Cls();

   if (welcome_messages !=NULL){
      printf("%s",welcome_messages);
```

```
        }
        printf("Select An Option:\n");
    int i;
    for (i=0; i<items;i++){
      printf("%d. %s\n",i+1,Items[i]);
    }
    printf("%d. Exit\n",i+1);
    printf(">");
        fflush_stdin();
    scanf("%d",&choice);
    if(choice < 1 || choice > items+1){
        printf("Invalid choice...press any key to retry....\n");
        fflush_stdin();
        getchar();
    }
  }while(choice < 1 || choice > items+1);
  return choice;

}

char* prompt_item(char* prompt,bool empty_allowed){
    char* item = malloc(sizeof(char) * 100);
    do{
        printf("%s",prompt);
        fflush_stdin();
        scanf("%[^\n]",item);//input until /n
        if(!empty_allowed&&strcmp(item,"") == 0){//check if temp is not empty
            printf("Invalid input, try again?(y/n)\n");
            char temp[100];
            fflush_stdin();
            scanf("%[^\n]",temp);
            if(strcmp(temp,"n") == 0){
                return NULL;
            }
        }
    }while(!empty_allowed&&strcmp(item,"") == 0);
    return item;
}
```

```
typedef enum{
  INT=1,STRING=2,LONG=3,DOUBLE=4
}INPUT_TYPE;
bool item_inputer(char * varname,INPUT_TYPE type,void * item,bool
empty_allowed){
  char output[1024] = "Please input the ";
  strcat(output,varname);
  strcat(output,": \n>");
  char* temp = prompt_item(output,empty_allowed);
  if(temp == NULL){
        return false;
  }
  switch(type){
    case INT:
      *((int*)item) = atoi(temp);
      break;
    case STRING:
      strcpy(item,temp);
      break;
    case LONG:
      *((long*)item) = atol(temp);
      break;
    case DOUBLE:
      *((double*)item) = atof(temp);
      break;
    default:
      return false;
  }
  return true;
}

void lister(
  void * db,
  struct Map* map,
    int row,
    char * lister_name,
    char * SortItems[],
    int NoSortItems,
```

```
  void (*page_printer)(void*,int,int,struct Map*),
  struct Map* (*sorter)(void *,int),void * (*showitem)(void *,int))
{
 int choice = -1;
  int page = 0;
  int page_size = SIZE_OF_PAGE;
  int total_pages = ceil((double)row / page_size);
  do{
  Cls();
  welcome_message(stdout);
  printf("%s list\n",lister_name);
  printf("0 exit\n");
  for(int i=0;i<NoSortItems*2;i+=2){
    printf("%d sort %s decending\n",i+1,SortItems[i/2]);
    printf("%d sort %s ascending\n",i+2,SortItems[i/2]);
  }
  if(page+1 == total_pages){
     (*page_printer)(db,page*page_size,row,map);
  }else{
     (*page_printer)(db,page*page_size,(page+1)*page_size,map);
  }
  //page control
  int current_page_size = page_size+NoSortItems*2;
  if(page+1 == total_pages){
     current_page_size = row - page*page_size+NoSortItems*2;
  }
  printf("%d next page\n", current_page_size+1);
  printf("%d previous page\n", current_page_size+2);
  printf("%d set page size\n", current_page_size+3);
  printf("%d/%d/%d(page size/page number/total)\n",page_size,
page+1,total_pages);

  bool valid = true;
  do{
     valid = true;
     printf("Please input your choice\n>");
     fflush_stdin();
     scanf("%d",&choice);
```

```
        if(choice <=NoSortItems*2 && choice > 0){
          printf("sorting...\n");
          map = (*sorter)(db,choice);
        }else if(choice == current_page_size+1 ){
          if(page + 1 < total_pages){
            page++;
          }else{
            printf("Already at last page\n");
            valid = false;
          }
        }else if(choice == current_page_size+2){
          if(page > 0){
            page--;
          }else{
            printf("Already at first page\n");
            valid = false;
          }
        }else if(choice == current_page_size+3){
          printf("Enter page size: ");
          fflush_stdin();
          scanf("%d", &page_size);
          total_pages = ceil((double)row / page_size);
        }else if(choice > NoSortItems*2 && choice <= current_page_size){
          if(map == NULL){
            db = (*showitem)(db,choice - NoSortItems*2-1 + page_size*page);
          }else{
            db = (*showitem)(db,map[choice - NoSortItems*2-1 +
page_size*page].key);
          }
        }else if(choice != 0){
          printf("Invalid choice\n");
          valid = false;
        }

    }while(!valid);
    }while(choice != 0);
    return;
}
```

```
//Size to be compressed to, > 3(...), should be some larger number else seeing nothing
char * StringCompression(char * str, int size){
  if(strlen(str) > size){
    str[size-3] = '.';
    str[size-2] = '.';
    str[size-1] = '.';
    str[size] = '\0';
  }
  return str;
}



//check if strcasestr is define in current environment
//if not, define it
#ifndef HAVE_STRCASESTR
char *strcasestr(const char *s, const char *find)
{
char c, sc;
size_t len;

if ((c = *find++) != 0) {
c = tolower((unsigned char)c);
len = strlen(find);
do {
do {
if ((sc = *s++) == 0)
return (NULL);
} while ((char)tolower((unsigned char)sc) != c);
} while (strncasecmp(s, find, len) != 0);
s--;
}
return ((char *)s);
}
#endif
```

inv_control.h

```
//invetory control
void add_item();
void remove_item(struct inventory db,int index);
void update_item(struct inventory db,int index);
void * item_control(void * ddb,int index);
char * show_item_admin(struct inventory db,int index);
void * show_item(void * ddb,int index);
void print_page(void * ddb, int cur, int end,struct Map* map);
struct Map* sortItems(void * ddb, int sort);
void search_item(void * (*showitem)(void *,int));
void inv_control(){
	int choice = 0;
	do{
	char buf[1024];
	welcome_message(buf);
	sprintf(buf+strlen(buf),"Inventory control\n");
	choice = choices_selecter((char*[]){
	   "List(allow all operation include add)",
	      "Add item(shortcut to add item)",
	   "Search item"
	},3,buf);
	switch (choice){
	   case 1:;//list
	      //add a new element with product name new item
	      //for item control
	      struct inventory db = read_db_invt();
	      db.row = (struct inventory_row *)realloc(db.row, sizeof(struct
inventory_row) * (db.db.row_count + 1));
	      db.db.row_count += 1;
	      strcpy(db.row[db.db.row_count - 1].product,"CREATE NEW ITEM");
	      db.row[db.db.row_count - 1].barcode = -1024;//IMPORTANT: -1024 is
reserved for new item
```

```
            char * SortItems[] = {
               "Name",
               "Price",
               "Brand",
               "Category",
            };

lister(&db,NULL,db.db.row_count,"Inventory",SortItems,4,print_page,sortItems,item
_control);
            break;
         case 2://add item
            add_item();
            break;
         case 3:
            search_item(item_control);
            break;
         case 4://exit
            break;
         default://should not happen
            printf("Invalid choice\n");
            break;
      }
      }while(choice != 4);
}

char * show_item_admin(struct inventory db,int index){
      char * output = (char*)malloc(sizeof(char) * 2048);
   sprintf(output,"Product: %s\n", db.row[index].product);
   sprintf(output+strlen(output),"Catergory: %s\n", db.row[index].category);// strlen
to add offset
   sprintf(output+strlen(output),"Brand: %s\n", db.row[index].brand);
   sprintf(output+strlen(output),"Price: $%lf\n", db.row[index].price);
   sprintf(output+strlen(output),"Stock: %d\n", db.row[index].stock);
   sprintf(output+strlen(output),"Barcode: %ld\n",db.row[index].barcode);
   return output;
}
//for normal user
void * show_item(void * ddb,int index){
```

```
    struct inventory db = *((struct inventory*)ddb);
    Cls();
    printf("Product: %s\n", db.row[index].product);
    printf("Catergory: %s\n", db.row[index].category);
    printf("Brand: %s\n", db.row[index].brand);
    printf("Price: $%lf\n", db.row[index].price);
    printf("Stock: %d\n", db.row[index].stock);
    printf("Barcode: %ld\n",db.row[index].barcode);
    printf("Press any key to return to the list\n");
    fflush_stdin();
    getchar();
    return ddb;
}

void print_page(void * ddb, int cur, int end,struct Map* map){
    struct inventory db = *((struct inventory*)ddb);
    printf("%-5s%-15s%-15s%-10s%-10s%-10s%-10s\
n","No.","Product","Brand","Category","Price","Stock","Barcode");
    for (int i = cur; i < end; i++)
    {
        if(map != NULL){
            int index = map[i].key;
            printf("%-5d%-15s%-15s%-10s%-10.2lf%-10d%-10ld\
n",i+9,StringCompression(db.row[index].product,13),StringCompression(db.row[inde
x].brand,13),StringCompression(db.row[index].category,8),db.row[index].price,db.ro
w[index].stock,db.row[index].barcode);
        }
        else{
            printf("%-5d%-15s%-15s%-10s%-10.2lf%-10d%-10ld\
n",i+9,StringCompression(db.row[i].product,13),StringCompression(db.row[i].brand,
13),StringCompression(db.row[i].category,8),db.row[i].price,db.row[i].stock,db.row[i
].barcode);
        }
    }
}

void * item_control(void * ddb,int index){
    struct inventory db = *((struct inventory*)ddb);
```

```
    int choice = 0;
    if(db.row[index].barcode == -1024){//new item
        add_item();
        struct inventory temp = read_db_invt();//reappend new item at back
        temp.row = (struct inventory_row *)realloc(temp.row, sizeof(struct
inventory_row) * (temp.db.row_count + 1));
        temp.db.row_count += 1;
        strcpy(temp.row[temp.db.row_count - 1].product,"CREATE NEW ITEM");
        temp.row[temp.db.row_count - 1].barcode = -1024;//IMPORTANT: -1024 is
reserved for new item
        void * re = malloc(sizeof(temp));
        memcpy(re,&temp,sizeof(temp));
        return re;
    }
    char * item = show_item_admin(db,index);
    char welcome[4096];
    strcpy(welcome,item);
    strcat(welcome,"Operations\n");
    do
    {
            choice = choices_selecter((char*[]){
        "update item",
            "remove item"
        },2,welcome);
        switch (choice)
        {
            case 1:
                update_item(db,index);
                break;
            case 2:
                remove_item(db,index);
                break;
            default:
                break;
        }
    } while (choice != 3&&choice != 2);
    struct inventory temp = read_db_invt();
```

```
    temp.row = (struct inventory_row *)realloc(temp.row, sizeof(struct inventory_row)
* (temp.db.row_count + 1));
    temp.db.row_count += 1;
    strcpy(temp.row[temp.db.row_count - 1].product,"CREATE NEW ITEM");
    temp.row[temp.db.row_count - 1].barcode = -1024;//IMPORTANT: -1024 is
reserved for new item
    void * re = malloc(sizeof(temp));
    memcpy(re,&temp,sizeof(temp));
    return re;
}

void add_item(){
    Cls();
    printf("Add new item\n");
    struct inventory_row *item = (struct inventory_row *)malloc(sizeof(struct
inventory_row));
    if(!item_inputer("name",STRING,&item->product,false)       ||
        !item_inputer("brand",STRING,&item->brand,false)       ||
        !item_inputer("category",STRING,&item->category,false) ||
        !item_inputer("price",DOUBLE,&item->price,false)       ||
        !item_inputer("stock",INT,&item->stock,false)        ||
        !item_inputer("barcode",LONG,&item->barcode,false)
    ){
        free(item);
        return;
    }
    if(item == NULL || !append_inventory(item)){
        printf("Failed to add item\n");
    }else{
        printf("Item added\n");
    }
    printf("press any key to continue\n");
    fflush_stdin();
    getchar();
}

void update_item(struct inventory db,int index){
    char temp[100];
```

```
    printf("Update item(empty value to not change the value)\n");
    printf("Name: %s\n", db.row[index].product);
    printf("Brand: %s\n", db.row[index].brand);
    printf("Catergory: %s\n", db.row[index].category);
    printf("Price: $%lf\n", db.row[index].price);
    printf("Stock: %d\n", db.row[index].stock);
    printf("Barcode: %ld\n",db.row[index].barcode);
    item_inputer("name",STRING,&db.row[index].product,true);
    item_inputer("brand",STRING,&db.row[index].brand,true);
    item_inputer("category",STRING,&db.row[index].category,true);
    item_inputer("stock",DOUBLE,&db.row[index].price,true);
    item_inputer("barcode",LONG,&db.row[index].barcode,true);
    if(!update_db_invt(db)){
        printf("Failed to update item\n");
        exit(1);
    }else{
        printf("Item updated\n");
    }
    printf("press any key to continue\n");
    fflush_stdin();
    getchar();
}

void remove_item(struct inventory db,int index){
    printf("Remove item\n");
    printf("Are you sure you want to remove this item? (y/n)\n");
    char choice;
    fflush_stdin();
    scanf("%c", &choice);
    db.row[index].isdeleted = true;
    if(choice == 'y'){
        if(!update_db_invt(db)){
            printf("Failed to remove item\n");
        }else{
            printf("Item removed\n");
        }
    }else{
        printf("Item not removed\n");
```

```
    }
    printf("press any key to continue\n");
    fflush_stdin();
    getchar();
}

struct Map* sortItems(void * ddb, int sort){
    struct inventory db = *((struct inventory*)ddb);
    struct Map *map = malloc(sizeof(struct Map) * db.db.row_count -1);
    for (int i = 0; i < db.db.row_count-1; i++){
        map[i].key = i;

        switch(sort){
            case 1:
            case 2:
                map[i].value = (void*)db.row[i].product;
                break;

            case 3:
            case 4:;
                double price = db.row[i].price * 100;
                double * price_star = malloc(sizeof(long));
                *price_star = price;
                map[i].value = (void*)price_star;//presume there is no price contain  0.001
                break;
            case 5:
            case 6:
                map[i].value = (void*)db.row[i].brand;
                break;
            case 7:
            case 8:
                map[i].value = (void*)db.row[i].category;
                break;
        }
    }
    switch (sort){
        case 1:
        case 5:
```

```
        case 7:
            qsort(map, db.db.row_count-1, sizeof(struct Map), compare_decending_str);
            break;
        case 2:
        case 6:
        case 8:
            qsort(map, db.db.row_count-1, sizeof(struct Map), compare_ascending_str);
            break;
        case 3:
            qsort(map, db.db.row_count-1, sizeof(struct Map), compare_decending);
            break;
        case 4:
            qsort(map, db.db.row_count-1, sizeof(struct Map), compare_ascending);
            break;
    }
    map = realloc(map, sizeof(struct Map) * db.db.row_count);
    map[db.db.row_count-1].key = db.db.row_count-1;
    return map;
}
```

tran_control.h

```
//tran control
void add_tran();
void print_tran(void * ddb, int cur, int end,struct Map* map);
void * showTran(void * ddb,int index);
struct Map* sortTrans(void * ddb,int choice);

void tran_control(){
    struct transaction tran = read_db_tran();
    int choice;
    do{
        char buf[1024];
        welcome_message(buf);
        sprintf(buf+strlen(buf),"Transaction control\n");
```

```
        choice = choices_selecter((char*[]){
            "List transaction",
                "new transaction"
        },2,buf);
        switch(choice){
            case 1:;
                char * SortItems[] = {
                    "Date&Time",
                    "product(barcode)",
                    "quantity",
                    "total",
                };

lister(&tran,NULL,tran.db.row_count,"Transaction",SortItems,4,print_tran,sortTrans,
showTran);
                break;
            case 2:
                add_tran();
                break;
            default:
                break;
        }
    }while(choice != 3);
}

void print_tran(void * ddb, int cur, int end,struct Map* map){
    struct transaction db = *((struct transaction*)ddb);
    printf("%-5s%-15s%-10s%-10s%-10s%-10s%-10s%-10s\
n","No.","Date","Time","Barcode","ID","Price","Quantity","Total");
    for (int i = cur; i < end; i++)
    {
        if(map != NULL){
            int index = map[i].key;
            double total = db.row[index].price * db.row[index].quantity;
            //reconstuct date and time and print all transaction info out
            char date[11];
            char time[9];
```

```
            sprintf(date,"%02d-%02d-
%04d",db.row[index].date.day,db.row[index].date.month,db.row[index].date.year);
            sprintf(time,"%02d:%02d:
%02d",db.row[index].time.hour,db.row[index].time.minute,db.row[index].time.secon
d);
            printf("%-5d%-15s%-10s%-10ld%-10ld%-10.2lf%-10d%-10.2lf\
n",i+9,date,time,db.row[index].barcode,db.row[index].id,db.row[index].price,db.row[
index].quantity,total);
        }else{
            double total = db.row[i].price * db.row[i].quantity;
            //reconstuct date and time and print all transaction info out
            char date[11];
            char time[9];
            sprintf(date,"%02d-%02d-
%04d",db.row[i].date.day,db.row[i].date.month,db.row[i].date.year);
            sprintf(time,"%02d:%02d:
%02d",db.row[i].time.hour,db.row[i].time.minute,db.row[i].time.second);
            printf("%-5d%-15s%-10s%-10ld%-10ld%-10.2lf%-10d%-10.2lf\
n",i+9,date,time,db.row[i].barcode,db.row[i].id,db.row[i].price,db.row[i].quantity,tota
l);
        }
    }
}
//show trans

struct transaction update_tran(struct transaction db,int index);
struct transaction remove_tran(struct transaction db,int index);

void * showTran(void * ddb,int index){
    struct transaction db = *((struct transaction*)ddb);
    int choice;
    do{
        char buff[1024];
        sprintf(buff,"Transaction detail\n");
        double total = db.row[index].price * db.row[index].quantity;
        //reconstuct date and time and print all transaction info out
        char date[11];
        char time[9];
```

```
        sprintf(date,"%02d-%02d-
%04d",db.row[index].date.day,db.row[index].date.month,db.row[index].date.year);
        sprintf(time,"%02d:%02d:
%02d",db.row[index].time.hour,db.row[index].time.minute,db.row[index].time.secon
d);
        sprintf(buff+strlen(buff),"%-15s%-10s%-10s%-10s%-10s%-10s%-10s\
n","Date","Time","Barcode","ID","Price","Quantity","Total");
        sprintf(buff+strlen(buff),"%-15s%-10s%-10ld%-10ld%-10.2lf%-10d%-10.2lf\
n",date,time,db.row[index].barcode,db.row[index].id,db.row[index].price,db.row[inde
x].quantity,total);
        choice = choices_selecter((char*[]){
            "Edit transaction",
            "Delete transaction",
        },2,buff);
        switch(choice){
            case 1:
                db = update_tran(db,index);
                break;
            case 2:
                db = remove_tran(db,index);
                break;
            case 3:
                break;
            default:
                printf("Invalid Choice\n");
                break;
        }
    }while(choice != 3);
    void * re = malloc(sizeof(db));
    memcpy(re,&db,sizeof(db));
    return re;
}

//tran controls
void add_tran(){
    char* temp;
    struct transaction_row* row = (struct transaction_row*)malloc(sizeof(struct
transaction_row));
```

```
char check;
printf("Use current date time? (y/n): ");
fflush_stdin();
scanf("%c",&check);
if(check == 'y'){
   row->date = get_date();
   row->time = get_time();
}else{
   if( !item_inputer("date:day",INT,&row->date.day,false) ||
      !item_inputer("date:month",INT,&row->date.month,false) ||
      !item_inputer("date:year",INT,&row->date.year,false) ||
      !item_inputer("time:hour",INT,&row->time.hour,false) ||
      !item_inputer("time:minute",INT,&row->time.minute,false) ||
      !item_inputer("time:second",INT,&row->time.second,false)
   ){
      free(row);
      return;
   }
}
if (
   !item_inputer("price",DOUBLE,&row->price,false) ||
   !item_inputer("quantity",INT,&row->quantity,false) ||
   !item_inputer("ID",LONG,&row->id,false) ||
   !item_inputer("barcode",LONG,&row->barcode,false)
){
   free(row);
   return;
}
if(!row || !append_transaction_db(row)){
   printf("Failed to add item\n");
}else{
   printf("Item added\n");
}
printf("press any key to continue\n");
fflush_stdin();
getchar();
```

```
}

struct transaction update_tran(struct transaction db,int index){
   char temp[100];
   printf("Update transaction(empty value to not change the value)\n");
   char date[11],time[9];
   sprintf(date,"%02d-%02d-
%04d",db.row[index].date.day,db.row[index].date.month,db.row[index].date.year);
   sprintf(time,"%02d:%02d:
%02d",db.row[index].time.hour,db.row[index].time.minute,db.row[index].time.secon
d);
   printf("%-15s%-10s%-10s%-10s%-10s%-10s\
n","Date","Time","Barcode","ID","Price","Quantity");
   printf("%-15s%-10s%-10ld%-10ld%-10.2lf%-10d\
n",date,time,db.row[index].barcode,db.row[index].id,db.row[index].price,db.row[inde
x].quantity);

   item_inputer("Date:day",INT,&db.row[index].date.day,true);
   item_inputer("Date:month",INT,&db.row[index].date.month,true);
   item_inputer("Date:year",INT,&db.row[index].date.year,true);
   item_inputer("Time:hour",INT,&db.row[index].time.hour,true);
   item_inputer("Time:minute",INT,&db.row[index].time.minute,true);
   item_inputer("Time:second",INT,&db.row[index].time.second,true);
   item_inputer("Price",DOUBLE,&db.row[index].price,true);
   item_inputer("Quantity",INT,&db.row[index].quantity,true);
   item_inputer("ID",LONG,&db.row[index].id,true);
   item_inputer("Barcode",LONG,&db.row[index].barcode,true);

   if(unlikely(!update_db_tran(db))){
      printf("Failed to update transaction\n");
      exit(1);//exit program to prevent errors
   }else{
      printf("Transaction updated\n");
   }
   printf("press any key to continue\n");
   fflush_stdin();
   getchar();
```

```
    return db;
}

struct transaction remove_tran(struct transaction db,int index){
    db.row[index].isdeleted = true;
    if(unlikely(!update_db_tran(db))){
        printf("Failed to delete transaction\n");
        exit(1);//exit program to prevent errors
    }else{
        printf("Transaction deleted\n");
    }
    printf("press any key to continue\n");
    fflush_stdin();
    getchar();

    return read_db_tran();
}

//sort transaction

struct Map* sortTrans(void * ddb,int choice){
    struct transaction db = *((struct transaction*)ddb);
    struct Map *map = malloc(sizeof(struct Map) * db.db.row_count);
    printf("l");
    for (int i = 0; i < db.db.row_count; i++){
        map[i].key = i;
        switch(choice){
            case 1:
            case 2:;
                long long temp = (long long)db.row[i].time.second+
db.row[i].time.minute*60 + db.row[i].time.hour*3600 + db.row[i].date.day*86400 +
db.row[i].date.month*2592000 + db.row[i].date.year*31104000;
                long long* tempstar = malloc(sizeof(long long));
                *tempstar = temp;
                map[i].value = (void*)tempstar;
                break;

            case 3:
```

```
            case 4:;
                long* tempstar2 = malloc(sizeof(long));
                *tempstar2 = db.row[i].barcode;
                map[i].value = (void*)tempstar2;
                break;
            case 5:
            case 6:;
                int* tempstar3 = malloc(sizeof(int));
                *tempstar3 = db.row[i].quantity;
                map[i].value = (void*)tempstar3;
                break;
            case 7:
            case 8:;
                long temp4 = lround(db.row[i].price * db.row[i].quantity * 10000);//clear
all decimal places
                long* tempstar4 = malloc(sizeof(long));
                *tempstar4 = temp4;
                map[i].value = (void*)tempstar4;
                break;
        }
    }
    switch (choice){
        case 1:
        case 3:
        case 5:
        case 7:
            qsort(map, db.db.row_count, sizeof(struct Map), compare_decending);
            break;
        case 2:
        case 4:
        case 6:
        case 8:
            qsort(map, db.db.row_count, sizeof(struct Map), compare_ascending);
            break;
    }
    return map;
}
```

user_control.h

```
//include std lib if they havent been included
#ifndef STD_LIB_H
#define STD_LIB_H
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <math.h>
#include <time.h>
#include <ctype.h>
#endif

#ifndef Utils
#define Utils
#include "sorting.h"
#include "utils.h"
#endif // !Utils
//list of functions
#include "inv_control.h"
#include "tran_control.h"
#include "user_control.h"
#include "role_control.h"
struct user_row *prompt_user();
int admin_menu_user_choices(char* name,char* role);

void admin_menu(){
    Cls();
    welcome_message(stdout);
    //the selection menu
    struct user_row *user = prompt_user();
    if(user == NULL){//invalid user
        return;
```

```
    }
    char * Items[] = {
        "Inventory control",
        "Transaction control",
        "User control",
        "Role control",
    };
    char welcome[256];
    sprintf(welcome,"welcome %s(%s)\n",user->name,user->role);
    bool exit = false;
    do{

        switch (choices_selecter(Items,4,welcome))
        {
            case 1:;//action with inventory
                inv_control();
                break;
            case 2://action with transction
                tran_control();
                break;
            case 3://action with user
                user_control();
                break;
            case 4://role management
                role_control();
                break;
            case 5://Exit
                exit = true;
                break;
            default://invalid input ,should not happen,as it is already catch in above
function
                printf("Invalid choice\n");
                break;
        }
    }while(!exit);

    return;
}
```

```
//func for main for admin
struct user_row *prompt_user(){
    long user_id = 0;
    printf("Please tap your card(student card, staff card,etc)(or input the id)(0 to
cancel)\n>");
    scanf("%ld", &user_id);
    if(user_id == 0){
        printf("cancelled\n");
        printf("press any key to continue\n");
        fflush_stdin();
        getchar();
        return NULL;
    }
    struct user_row *user = get_user(user_id);
    Cls();
    if(user == NULL){
        printf("User not found\n");
        printf("press any key to continue\n");
        fflush_stdin();
        getchar();
        return NULL;
    }else if(!is_admin(user->role)){
        printf("You aren't an admin\n");
        printf("press any key to continue\n");
        fflush_stdin();
        getchar();
        return NULL;
    }else{
            printf("Welcome %s %s\n", user->name , user->role);
        printf("press any key to continue\n");
        fflush_stdin();
        getchar();
    }
    return user;
}
```

role_control.h

```
//role control
void add_role();
void print_role(struct linkedlist* list,int cur,int end,struct Map* map);
void * showRole(struct linkedlist* list,int index);
struct Map* sortRole(struct linkedlist* list,int choice);

void role_control(){
    int choice = -1;
    struct user db = read_db_user();
    do{
        char buf[1024];
        welcome_message(buf);
        sprintf(buf+strlen(buf),"Role control\n");
        choice = choices_selecter((char*[]){
            "add admin role",
                "list role"
        },2,buf);
        switch (choice){
            case 3:
                break;
            case 1:
                add_role();
                break;
            case 2:;
                char * SortItems[] = {
                    "admin",
                    "role",
                };
                void * list = getLinkedList(role_list_db(),0);

lister(list,NULL,sizeofLinkedlist(list),"Role",SortItems,2,print_role,sortRole,showRol
```

```
e);
            //warning appears but can be ignore due to the fact that they are just pointer,
no difference
            break;
        default:
            break;
    }
  }while(choice != 3);
}


void add_role(){
  char role[100];
  printf("Enter Admin role name: ");
  fflush_stdin();
  scanf("%s", role);
  if(is_admin(role)){//check if role exist in admin file
    printf("Admin role already exist\n");
    printf("press any key to continue\n");
    fflush_stdin();
    getchar();
  }else{
    add_admin(role);
    printf("role added\n");
    printf("press any key to continue\n");
    fflush_stdin();
    getchar();
  }
  return;
}
void print_role(struct linkedlist* list,int cur,int end,struct Map* map){
  printf("%-10s%-10s%-10s\n","No.","Role","Admin?");
  if(map == NULL){
    for(int i = cur; i < end; i++){
      list = getLinkedList(list,i);
      char* name = list->data;
      printf("%-10d%-10s%-10s\n",i-cur +
5,StringCompression(name,8),is_admin(name)?"T":"F");
```

```
        }
    }else{
        for(int i = cur; i < end; i++){
            list = getLinkedList(list,map[i].key);
            char* name = list->data;
            printf("%-10d%-10s%-10s\n",i-cur +
5,StringCompression(name,8),is_admin(name)?"T":"F");
        }
    }
}

void * showRole(struct linkedlist* list,int index){
    int choice;
    list = getLinkedList(list,index);
    char* name = list->data;
    Cls();
    printf("%-10s%-10s\n","Role","Admin?");
    printf("%-10s%-10s\n",name,is_admin(name)?"T":"F");
    choice = choices_selecter((char*[]){"Toggle admin"},1,"Role control\n");
    if(choice == 1){
        if(is_admin(name)){
            remove_admin(name);
        }else{
            add_admin(name);
        }
        printf("Admin role toggled\n");
        printf("Press any key to continue\n");
        fflush_stdin();
        getchar();
    }
    return list;
}


struct Map* sortRole(struct linkedlist* list,int choice){
    struct Map* map = malloc(sizeof(struct Map)*sizeofLinkedlist(list));
    struct linkedlist* cur = getLinkedList(list,0);
    for(int i = 0; i < sizeofLinkedlist(list); i++){
```

```
        switch(choice){
            case 1:
            case 2:
                map[i].key = i;
                bool temp = !is_admin(cur->data);
                bool* tempstar = malloc(sizeof(bool));
                *tempstar = temp;
                map[i].value = tempstar;
                break;
            case 3:
            case 4:
                map[i].key = i;
                map[i].value = cur->data;
                break;
        }
        cur = cur->next;
    }
    switch(choice){
        case 1:
        case 3:
            qsort(map,sizeofLinkedlist(list),sizeof(struct Map),compare_decending_str);
            break;
        case 2:
        case 4:
            qsort(map,sizeofLinkedlist(list),sizeof(struct Map),compare_ascending_str);
            break;
    }
    return map;
}
```

normal_user.h

```
//include std lib if they havent been included
#ifndef STD_LIB_H
#define STD_LIB_H
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<stdbool.h>
#include<math.h>
#include<time.h>
#include<ctype.h>
#endif

#ifndef Utils
#define Utils
#include "sorting.h"
#include "utils.h"
#endif // !Utils
//fucntions for menu
void search_item(void * (*showitem)(void *,int));
void self_help_sale_system();
void print_page(void * ddb, int cur, int end,struct Map* map);
struct Map* sortItems(void * ddb, int sort);

//main of normal user
void normal_menu(){
   int choice = 0;
   do{
      char buf[1024];
      welcome_message(buf);
      sprintf(buf+strlen(buf),"Normal User Menu\n");
            choice = choices_selecter((char*[]){
         "List items",
```

```
            "Search item",
            "Self help sale system"
        },3,buf);
        switch (choice)
        {
            case 1:;//List items
                struct inventory db = read_db_invt();
                char * SortItems[] = {
                    "Name",
                    "Price",
                    "Brand",
                    "Category",
                };
                lister(&db,NULL,db.db.row_count,"List of
items",SortItems,4,print_page,sortItems,show_item);
                break;
            case 2://Search item
                search_item(show_item);
                break;
            case 3://self help sale system
                self_help_sale_system();
                break;
            default://invalid input ,should not happen,as it is already catch in above
function
                break;
        }
    }while(choice != 4);
}

//search items
char* prompt_search();
struct Map* searchItems(struct inventory db, char* searchstr);
void search_item(void * (*showitem)(void *,int)){

    //search for item
    //prompt user to select an item
    //if user selects an item, display the item's details
    struct inventory db = read_db_invt();
```

```
    struct Map* map = NULL;
    do{

    //options
    int breakout = 0;
    breakout = choices_selecter((char*[]){"Search"},1,welcome_message(NULL));
    if(breakout == 2){
        break;
    }
    char* searchstr = prompt_search();
    map = searchItems(db,searchstr);
    if(map[0].value > 0){
        int temp_row = *((int*)map[0].value);
        char * SortItems[] = {
            "Name",
            "Price",
            "Brand",
            "Category",
        };
        lister(&db,map+1,temp_row,"List of
items",SortItems,4,print_page,sortItems,showitem);//ofset map, as it is use to store the
size
    }else{//empty search
        printf("No result found\n");
        printf("Press any key to continue\n");
        fflush_stdin();
        getchar();

    }
    }while(true);//break on top

}


char* prompt_search(){
printf("Enter search string(largest length 100)\n(case insensitive)\nmatches multiple
columns\n>");
char* searchstr = malloc(sizeof(char) * 100);
```

```
fflush_stdin();
scanf("%s", searchstr);
printf("searching...\n");
return searchstr;
}



struct Map* searchItems(struct inventory db, char* searchstr){
struct Map* map = malloc(sizeof(struct Map) * (db.db.row_count+1));
int k = 1;
for (int i = 0; i < db.db.row_count; i++){
map[k].key = i;
if(strcasestr(db.row[i].product,searchstr) != NULL||
strcasestr(db.row[i].brand,searchstr) != NULL||
strcasestr(db.row[i].category,searchstr) != NULL||
strcasestr(lto_string(db.row[i].barcode),searchstr) != NULL
){
map[k].value = (void*)db.row[i].product;
k++;
}
}
int* k_star = malloc(sizeof(int));
*k_star = --k;//decrement and assign to k star;don't use k-- as it will decrement after
the assignment
map[0].value = (void*)k_star;
map[0].key = -1;
return map;
}



//self help

struct cart* scan_barcode(struct cart* cart,struct inventory db);
struct cart* list_cart(struct cart* cart);
void self_help_sale_system(){
```

```
    //scan barcode
    struct inventory db = read_db_invt();
    struct cart* cart = NULL;
    int choice = 0;
    do{

    //options
    int breakout = 0;
    choice = choices_selecter((char*[]){
        "scan barcode",
        "cart/checkout"
    },2,welcome_message(NULL));
    switch (choice)
    {
    case 1:
        cart = scan_barcode(cart,db);
        break;
    case 2:
        cart = list_cart(cart);
        break;
    default:
        break;
    }
    }while(choice!=3);
}

//support for the linked list
struct cart* Cartappend(struct cart* cart,struct inventory_row* row,int quantity){
    if(cart == NULL){
        cart = malloc(sizeof(struct cart));
        cart->row = row;
        cart->quantity = quantity;
        cart->next = NULL;
    }else{
        struct cart* temp = cart;
        while(temp->next != NULL){
            temp = temp->next;
        }
```

```
        temp->next = malloc(sizeof(struct cart));
        temp->next->row = row;
        temp->next->quantity = quantity;
        temp->next->next = NULL;
    }

    return cart;
}

struct cart* Cartremove(struct cart* cart,int index){
    if (cart == NULL){
        return cart;
    }else{
        struct cart* temp = cart;
        if (index == 0){
            cart = cart->next;
            free(temp);
            return cart;
        }else{
            for (int i = 0; i < index-1; i++){
                temp = temp->next;
            }
            struct cart* new = temp->next->next;
            free(temp->next);
            temp->next = new;
            return cart;
        }
    }
}
struct cart* Cartupdate(struct cart* cart,int index,int quantity){
    if (cart == NULL){
        return cart;
    }else{
        struct cart* temp = cart;
        if (index == 0){
            cart->quantity = quantity;
            return cart;
        }else{
```

```
            for (int i = 0; i < index-1; i++){
                temp = temp->next;
            }
            temp->next->quantity = quantity;
            return cart;
        }
    }
}

struct inventory_row* Cartget(struct cart* cart,int index){
    if (cart == NULL){
        return NULL;
    }else{
        struct cart* temp = cart;
        if (index == 0){
            return cart->row;
        }else{
            for (int i = 0; i < index-1; i++){
                temp = temp->next;
            }
            return temp->next->row;
        }
    }
}

int Cartqty(struct cart* cart,int index){
    if (cart == NULL){
        return 0;
    }else{
        struct cart* temp = cart;
        if (index == 0){
            return cart->quantity;
        }else{
            for (int i = 0; i < index-1; i++){
                temp = temp->next;
            }
            return temp->next->quantity;
        }
```

```
    }
}

//scan barcode
struct inventory_row* find_barcode(struct inventory db,long barcode);
long prompt_barcode();
struct cart* scan_barcode(struct cart* cart,struct inventory db){

    long barcode = prompt_barcode();
    printf("matching...\n");
    struct inventory_row* row = find_barcode(db,barcode);
    if(row != NULL){
        int quantity = 0;
        char choice = 'n';
        do{
            Cls();
            printf("product: %s\n",row->product);
            printf("price: %.1f\n",row->price);
            printf("Enter quantity(0 to cancel)\n>");\
            fflush_stdin();
            scanf("%d", &quantity);
            fflush_stdin();
            if(quantity == 0){
                printf("cancelled\n");
                printf("press any key to continue\n");
                fflush_stdin();
                getchar();
                return cart;
            }
            printf("Are you sure you want to add this item to cart?(y/n)\n>");
            fflush_stdin();
            scanf("%c", &choice);
            if(row->stock - quantity < 0 && choice == 'y'){
                printf("WARNING:not enough stock\n");
                printf("Are you sure the quantity is correct?\n(y to ignore and continue/n)\
n>");
                fflush_stdin();
                scanf("%c", &choice);
```

```
            }
        }while(choice != 'y');
        if(quantity != 0){
            cart = Cartappend(cart,row,quantity);
            printf("added to cart\n");
        }
    }else{//empty search
        printf("Unable to match or invalid input\n");

    }
    printf("press any key to continue\n");
    fflush_stdin();
    getchar();
    return cart;
}


long prompt_barcode(){
    printf("Please scan the qr code(or input the barcode) ");
    long barcode;
    fflush_stdin();
    scanf("%ld", &barcode);
    return barcode;
}

struct inventory_row* find_barcode(struct inventory db,long barcode){
    struct inventory_row row;
    for (int i = 0; i < db.db.row_count; i++){
        if(db.row[i].barcode == barcode){

            return &db.row[i];
        }
    }
    return (struct inventory_row*)NULL;
}


//list cart
```

```
struct cart* cart_control(struct cart* cart,int index);
struct cart* checkout(struct cart* cart);
struct cart* list_cart(struct cart* cart){
        int choice = 0;
    do{
        Cls();
        welcome_message(stdout);
        double total_price = 0;
        int i=1;
        printf("0 exit\n");
        if(cart == NULL){
            printf("Cart is empty\n");
        }else{
            struct cart* temp = cart;
            while(temp != NULL){//show cart
                int qty = temp->quantity;
                double price = temp->row->price * qty;
                total_price += price;
                printf("%d product:%s\n",i,temp->row->product);
                printf("  price(per qty): %.1f\n",temp->row->price);//space to align
                printf("  price(total): %.1f\n",price);
                printf("  quantity: %d\n",qty);
                printf("<------------------------>\n");
                temp = temp->next;
                i++;
            }
            printf("\n<------------------------>\n");
            printf("Total price: $%.1f\n",total_price);
            printf("<------------------------>\n");

            printf("\n%d CHECKOUT\n",i);
            printf("%d Clear Cart\n",i+1);
        }
        do{
            printf("input the corresponding value for more action\n>");
            fflush_stdin();
            scanf("%d", &choice);
            if(choice >0 && choice < i){
```

```
            cart = cart_control(cart,choice);
        }else if(choice == i && cart != NULL){
            cart = checkout(cart);
        }else if(choice == i+1 && cart != NULL){
            cart = NULL;
            printf("Cart cleared\n");
        }
    }while(choice <0 || choice > i+1);
  }while(choice != 0);
  return cart;
}
struct cart* update_cart(struct cart* cart,int index);
struct cart* cart_control(struct cart* cart,int index){
  int choice = 0;
  struct inventory_row* row = Cartget(cart,index-1);
  int quantity = Cartqty(cart,index-1);
  do{
    char welcome[256];
    welcome_message(welcome);
    sprintf(welcome+strlen(welcome),"product: %s\n",row->product);
    sprintf(welcome+strlen(welcome),"price: %.1f\n",row->price);
    sprintf(welcome+strlen(welcome),"quantity: %d\n",quantity);
        choice = choices_selecter((char*[]){
      "remove",
        "edit quantity"
    },2,welcome);

      if(choice == 1){
        cart = Cartremove(cart,index-1);
        printf("Remove successful\n");
        printf("press any key to continue\n");
        fflush_stdin();
        getchar();
      }else if(choice == 2){
        cart = update_cart(cart,index);
        printf("Update successful\n");
        printf("press any key to continue");
        fflush_stdin();
```

```
            getchar();
        }
    }while(choice != 1 || choice != 2 || choice !=3 );
    return cart;
}

struct cart* update_cart(struct cart* cart,int index){
    Cls();
    welcome_message(stdout);
    char choice;
    do{
        int quantity = 0;
        printf("enter the new quantity: ");
        fflush_stdin();
        scanf("%d", &quantity);
        printf("Are you sure you want to update this item?(y/n/e to exit)");
        fflush_stdin();
        scanf("%c", &choice);
        if(choice == 'y'){

            if(quantity != 0){
                cart = Cartupdate(cart,index-1,quantity);
            }else if (quantity == 0){
                cart = Cartremove(cart,index-1);
            }
        }
    }while(choice != 'y' && choice != 'e');
    return cart;
}



//Checkout

struct cart* checkout(struct cart* cart){
    Cls();
    welcome_message(stdout);
```

```
    long user_id = 0;
    printf("Please tap your card(student card, staff card,etc)(or input the id)(0 to
cancel)\n>");
    scanf("%ld", &user_id);
    if(user_id == 0){
        printf("cancelled\n");
        printf("press any key to continue\n");
        fflush_stdin();
        getchar();
        return cart;
    }
    struct user_row* user = get_user(user_id);


    if(user == NULL){
        printf("Fail to indentify your card or error occurs\n");
        printf("press any key to continue\n");
        fflush_stdin();
        getchar();
        return cart;
    }else{
        printf("Welcome %s(%s)\n",user->name,user->role);
        printf("Are you sure you want to checkout?(y/n)\n>");
        char choice;
        fflush_stdin();
        scanf("%c", &choice);
        if(choice == 'y'){
            if(update_stock_N_checkout(cart,user)){//catch err
                printf("Checkout successful\n");
                printf("press any key to continue\n");
                fflush_stdin();
                getchar();
                return NULL;
            }else{
                printf("Checkout failed(files maybe corrputed)\n");
                printf("press any key to continue\n");
                fflush_stdin();
                getchar();
```

```
            return cart;
        }
    }else{
        printf("cancelled\n");
        printf("press any key to continue\n");
        fflush_stdin();
        getchar();
        return cart;
    }
  }


  return cart;
}
```