**SOURCE CODE**

#include <iostream>

#include <windows.h>

#include <fstream>

#include <iomanip>

#include <string>

#include <cstring>

#include <cctype>

#include <conio.h>

using namespace std;

const int MAX\_SIZE = 20;

void run(int, struct Order rd[]);

void topping\_menu(string st);

void pizza(string st);

void drink\_menu(string st);

void size\_menu();

void sub\_orderMenu();

void food\_menu(string st);

int get\_option(string, string);

void update\_menu();

void sale\_typeMenu();

void sub\_order\_menu(struct Order rd[], int, int&, string);

void sub\_menu(struct Order rd[], int, int&, int&, string);

void sub\_topping\_menu(struct Order rd[], int, int&, int&, string);

void sub\_drink\_menu(struct Order rd[], int, int&, int&, string);

void main\_menu();

void sub\_size\_menu(struct Order rd[], string, int, int&, int&, string);

void reciept\_detail(struct Order rd[], string, float, int, int&, string);

void get\_detail(struct Order rd[], string, float, int, int&, string);

void reciept(struct Order rd[], int, int);

void all\_order\_view(struct Order rd[], int, int);

void get\_total\_kh(struct Order rd[], int, float);

void get\_vat(struct Order rd[], int, float);

void get\_net\_amt(struct Order rd[], int, float);

void reciept();

float get\_amount(int, float);

float get\_total(float, float);

int get\_numbers(string);

void order\_view(struct Order rd[], int, int);

int sub\_search(int, int);

void show\_message(string);

void display(struct Order rd[], int, int);

void sub\_order\_update\_menu(struct Order rd[], int, int, string, int&);

void sub\_delete(struct Order rd[], int, int&, int);

void order\_type\_menu(struct Order rd[], int);

string getString(string, string);

void get\_date(struct Order rd[], int);

void search\_order(struct Order rd[], int, int);

void delete\_order\_menu(struct Order rd[], int&);

void order\_update\_menu(struct Order rd[], int, float, float, float);

void sort\_order(struct Order rd[], int);

void swap(struct Order&, struct Order&);

bool deletes(struct Order rd[], int, int&, char);

void sub\_update\_menu(struct Order rd[], int, float, float, string);

void update(struct Order rd[], int i, int k, float price);

void get\_character(string st);

void payment\_update(struct Order rd[], int, float, float);

float check\_price(string);

bool validat\_option(string);

void repet\_show(string, string);

void search\_menu();

bool validat\_number(string);

bool validat\_id(string);

bool validat\_staff(string);

bool validat\_phoneNumber(string);

void gotoxy(short, short);

void setcolor(int);

void front\_display(int, int);

bool check\_string(string);

bool check\_character(char, int&, int&, int&);

string get\_name(int, int);

string get\_password(int, int);

bool check\_passwordInput(char, int&, int&, int&, int);

void clear\_text(int, int);

void intro\_changePassword();

void change\_password(struct login& account);

void writefile(struct login account);

void to\_account(int, struct Order rd[]);

void get\_detail(struct Order rd[], float, float, int, int);

void view(struct Order rd[], int, int);

int search(struct Order rd[], int, string, string);

void swap1(struct Order rd[], int, int);

void get\_inform(struct Order rd[], int, string);

void intro\_reciept\_header();

bool verify\_newPassword(struct login& account, string, string, string);

bool verify(struct login account, string, string, int, int);

string get\_usernameInput(int, int);

string get\_passwordInput(int, int, int);

void login\_menu();

// Structure Date

struct login

{

string password;

string username;

};

login get\_account()

{

login account;

show\_message("\tEnter username: ");

account.username = get\_usernameInput(23, 0);

show\_message("\tEnter password: ");

account.password = get\_passwordInput(24, 2, 24);

return account;

}

struct Date

{

int hour;

int minute;

int month;

int year;

int day;

};

// Structure Inform

struct Inform

{

string staff;

int table\_numbers;

string id;

string address;

string type\_sale;

string phone\_number;

};

// Structure Order

struct Order

{

float amount[MAX\_SIZE];

float total\_usd = 0;

float total\_kh = 0;

float vat = 0;

float net\_amt = 0;

int n[MAX\_SIZE];

string name[MAX\_SIZE];

int qty[MAX\_SIZE];

float retail\_price[MAX\_SIZE];

// Declaring variablesName of struct Date and struct Inform

struct Date date;

struct Inform inform;

};

// Main function

int main()

{

struct Order rd[MAX\_SIZE];

struct Order get\_rd;

int k = 0;

int i = 0;

ifstream loadFile;

loadFile.open("Customer.bat", ios::in | ios::binary);

while (loadFile.read((char\*)(&get\_rd), sizeof(get\_rd)))

{

rd[k] = get\_rd;

k++;

}

loadFile.close();

to\_account(k, rd);

return 0;

}

void to\_account(int k, struct Order rd[])

{

int option;

login account;

login getAccount;

enum OPTION

{

EXIT = 0,

REGISTER,

LOGIN,

CHANGE\_PASSWORD

};

do

{

do {

system("cls");

login\_menu();

option = get\_option("login", " ");

} while (option > 3);

switch (option)

{

case OPTION::REGISTER: {

system("cls");

account = get\_account();

writefile(account);

cout << endl;

show\_message("\tRegister successfully!!!\n");

}break;

case OPTION::LOGIN:

{

string username;

string password;

bool result;

ifstream readFile;

readFile.open("login.bat", ios::in | ios::binary);

while (readFile.read((char\*)&getAccount, sizeof(getAccount)))

{

account = getAccount;

}

readFile.close();

do {

system("cls");

front\_display(20, 7);

username = get\_name(20, 7);

password = get\_password(20, 7);

result = verify(account, username, password, 20, 7);

setcolor(15);

} while (result != true);

run(k, rd);

}

break;

case OPTION::CHANGE\_PASSWORD:

{

ifstream readFile;

string username;

string password;

bool result;

readFile.open("login.bat", ios::in | ios::binary);

while (readFile.read((char\*)&getAccount, sizeof(getAccount)))

{

account = getAccount;

}

readFile.close();

change\_password(account);

writefile(account);

do {

system("cls");

front\_display(20, 7);

username = get\_name(20, 7);

password = get\_password(20, 7);

result = verify(account, username, password, 20, 7);

setcolor(15);

get\_character("press any key to continue...");

} while (result != true);

run(k, rd);

}

break;

case OPTION::EXIT:

{

char ch;

cout << "\tAare you sure to exit (y/n): ";

cin >> ch;

if (ch == 'y' || ch == 'Y')

{

exit(0);

}

else {

option = -1;

cin.ignore();

}

}

break;

}

\_getch();

} while (option != 0);

}

void run(int k, struct Order rd[])

{

// Declaring Variables

int i = 0;

i = k;

int j;

//int n[MAX\_SIZE];

const float VAT\_PERCENTAGE = 0.1;

const int VAT = 10;

const float KH = 4100;

int option\_type;

enum MAIN\_MENU

{

EXIT = 0, ORDER, VIEW\_ORDER, UPDATE\_ORDER, SEARCH\_ORDER, DELETE\_ORDER, SORT\_ORDER, SAVE, LOAD

};

do

{

system("cls");

main\_menu();

option\_type = get\_option("main\_menu", "");

switch (option\_type)

{

case MAIN\_MENU::ORDER:

{

j = 0;

sub\_order\_menu(rd, i, j, "sub\_orderMenu");

get\_detail(rd, KH, VAT\_PERCENTAGE, i, j);

rd[i].n[i] = j;

i++;

}break;

case MAIN\_MENU::VIEW\_ORDER:

{

view(rd, i, VAT);

}break;

case MAIN\_MENU::SEARCH\_ORDER:

{

view(rd, i, VAT);

search\_order(rd, i, VAT);

}break;

case MAIN\_MENU::UPDATE\_ORDER:

{

view(rd, i, VAT);

order\_update\_menu(rd, i, KH, VAT\_PERCENTAGE, VAT);

}break;

case MAIN\_MENU::DELETE\_ORDER:

{

view(rd, i, VAT);

delete\_order\_menu(rd, i);

}break;

case MAIN\_MENU::SORT\_ORDER:

{

cout << "\t=======SORT MENU=======" << endl;

cout << "\tBY TOTAL\_AMOUNT" << endl;

sort\_order(rd, i);

show\_message("\tSort successfully!!!\n");

}break;

case MAIN\_MENU::SAVE:

{

ofstream saveFile;

saveFile.open("Customer.bat", ios::out | ios::binary);

for (int k = 0; k < i; k++)

{

saveFile.write((char\*)(&rd[k]), sizeof(rd[k]));

}

saveFile.close();

show\_message("\tFile save successfully!!!\n");

}break;

case MAIN\_MENU::LOAD:

{

show\_message("\tfile load successfully!!!\n");

}break;

case MAIN\_MENU::EXIT:

{

char st;

cout << "\tAre you sure to exit (Y/N)? ";

cin >> st;

if (st == 'y' || st == 'Y')

{

exit(0);

}

else

{

option\_type = -1;

cin.ignore();

}

}break;

default:

{

show\_message("\tInvalid!!!\n");

cin.ignore();

}break;

}

get\_character("\tPress any key to counitue...");

} while (option\_type != 0);

}

void sub\_order\_menu(struct Order rd[], int i, int& j, string st)

{

enum SUB\_ORDER\_MENU

{

BACK\_TO\_MAIN\_MENU = 0, ORDER, VIEW, UPDATE, SEARCH, DELETES

};

int option\_order;

int k;

do

{

system("cls");

sub\_orderMenu();

option\_order = get\_option("sub\_orderMenu", "");

switch (option\_order)

{

case SUB\_ORDER\_MENU::ORDER:

{

sub\_menu(rd, i, j, option\_order, st);

}break;

case SUB\_ORDER\_MENU::VIEW:

{

intro\_reciept\_header();

order\_view(rd, j, i);

get\_character("\tPress any key to counitue...");

}break;

case SUB\_ORDER\_MENU::SEARCH:

{

int neutron;

intro\_reciept\_header();

order\_view(rd, j, i);

neutron = get\_numbers("\tPlease input neutron(No): ");

k = sub\_search(neutron, j);

if (k != -1)

{

show\_message("\tSearch is found!!!\n");

intro\_reciept\_header();

display(rd, k, i);

}

else

{

show\_message("\tSearch no found!!!\n");

}

get\_character("\tPress any key to counitue...");

}break;

case SUB\_ORDER\_MENU::UPDATE:

{

intro\_reciept\_header();

order\_view(rd, j, i);

sub\_order\_update\_menu(rd, i, j, "update", option\_order);

get\_character("\tPress any key to counitue...");

}break;

case SUB\_ORDER\_MENU::DELETES:

{

int neutron;

char key;

intro\_reciept\_header();

order\_view(rd, j, i);

neutron = get\_numbers("\tPlease input neutron(No): ");

k = sub\_search(neutron, j);

if (k != -1)

{

show\_message("\tSearch is found!!!\n");

show\_message("\tAre you sure to delete (Y/N)? ");

cin >> key;

if (key == 'y' || key == 'Y')

{

rd[i].total\_usd = rd[i].total\_usd - rd[i].amount[k];

sub\_delete(rd, k, j, i);

show\_message("\tDelete successfully!!!\n");

}

}

else

{

show\_message("\tSearch not found!!!\n");

}

get\_character("\tPress any key to counitue...");

cin.ignore();

}break;

case SUB\_ORDER\_MENU::BACK\_TO\_MAIN\_MENU: {}break;

default:

{

show\_message("\tInvalid!!!\n");

cin.ignore();

get\_character("\tPress any key to counitue...");

}

}

} while (option\_order != 0);

}

void sub\_menu(struct Order rd[], int i, int& j, int& option\_order, string st)

{

enum FOOD\_MENU

{

BACK\_TO\_SUB\_ORDER\_MENU = 0, PIZZA, DRINKS, BACK\_TO\_MAIN\_MENU

};

int option\_menu;

do

{

if (st == "order\_update" || st == "update")

{

do

{

system("cls");

food\_menu(st);

option\_menu = get\_option("food\_menu", st);

} while (option\_menu > 2);

}

else

{

do

{

system("cls");

food\_menu(st);

option\_menu = get\_option("food\_menu", st);

} while (option\_menu > 3);

}

switch (option\_menu)

{

case FOOD\_MENU::PIZZA:

{

sub\_topping\_menu(rd, i, j, option\_menu, st);

}break;

case FOOD\_MENU::DRINKS:

{

sub\_drink\_menu(rd, i, j, option\_menu, st);

}break;

case FOOD\_MENU::BACK\_TO\_MAIN\_MENU:

{

option\_order = 0;

option\_menu = 0;

}break;

case FOOD\_MENU::BACK\_TO\_SUB\_ORDER\_MENU:

{

if (st == "update")

{

option\_order = 0;

}

}break;

default:

{

show\_message("\tInvalid!!!\n");

get\_character("\tPress any key to counitue...");

}break;

}

} while (option\_menu != 0);

}

void sub\_topping\_menu(struct Order rd[], int i, int& j, int& option\_menu, string st)

{

enum TOPPING\_MENU

{

BACK\_TO\_FOOD\_MENU = 0, CHICKEN, SEAFOOD, SAUSAGE, BACK\_TO\_SUB\_ORDER\_MENU

};

int option\_topping;

do

{

system("cls");

if (st == "order\_update")

{

do

{

system("cls");

pizza(st);

option\_topping = get\_option("topping\_menu", st);

} while (option\_topping > 3);

st = "update";

}

else

{

do

{

system("cls");

pizza(st);

option\_topping = get\_option("topping\_menu", st);

} while (option\_topping > 5);

}

switch (option\_topping)

{

case TOPPING\_MENU::CHICKEN:

{

sub\_size\_menu(rd, "Chicken", i, j, option\_topping, st);

}break;

case TOPPING\_MENU::SAUSAGE:

{

sub\_size\_menu(rd, "Sausage", i, j, option\_topping, st);

}break;

case TOPPING\_MENU::SEAFOOD:

{

sub\_size\_menu(rd, "Seafood", i, j, option\_topping, st);

}break;

case TOPPING\_MENU::BACK\_TO\_FOOD\_MENU:

{

}break;

case TOPPING\_MENU::BACK\_TO\_SUB\_ORDER\_MENU:

{

option\_menu = 0;

option\_topping = 0;

}break;

default:

{

show\_message("\tInvalid!!!\n");

get\_character("\tPress any key to counitue...");

}break;

}

} while (option\_topping != 0);

}

void sub\_size\_menu(struct Order rd[], string topping\_name, int i, int& j, int& option\_topping, string st)

{

enum SIZE\_MENU

{

BACK\_TO\_TOPPING\_MENU = 0, LARGE, MEDIUM, SMALL, BACK\_TO\_FOOD\_MENU,

};

int option\_size;

float price;

string productName;

do

{

system("cls");

size\_menu();

option\_size = get\_option("size\_menu", "");

switch (option\_size)

{

case SIZE\_MENU::LARGE:

{

productName = ("L-" + topping\_name);

price = check\_price(productName);

get\_detail(rd, productName, price, i, j, st);

}break;

case SIZE\_MENU::MEDIUM:

{

productName = ("M-" + topping\_name);

price = check\_price(productName);

get\_detail(rd, productName, price, i, j, st);

}break;

case SIZE\_MENU::SMALL:

{

productName = ("S-" + topping\_name);

price = check\_price(productName);

get\_detail(rd, productName, price, i, j, st);

}break;

case SIZE\_MENU::BACK\_TO\_FOOD\_MENU:

{

option\_topping = 0;

option\_size = 0;

}

break;

case SIZE\_MENU::BACK\_TO\_TOPPING\_MENU: {}break;

default:

{

show\_message("\tInvalid!!!\n");

}break;

}

get\_character("\tPress any key to counitue...");

} while (option\_size != 0);

}

void sub\_drink\_menu(struct Order rd[], int i, int& j, int& option\_menu, string st)

{

enum DRINK\_MENU

{

BACK\_TO\_FOOD\_MENU = 0, STING, COCA\_COLA, SPRITE, DRINKING\_WATER, BACK\_TO\_SUB\_ORDER\_MENU

};

int option\_drink;

string productName;

float price;

do

{

system("cls");

if (st == "order\_update")

{

do

{

system("cls");

drink\_menu(st);

option\_drink = get\_option("drink\_menu", st);

} while (option\_drink > 4);

}

else

{

do

{

system("cls");

drink\_menu(st);

option\_drink = get\_option("drink\_menu", st);

} while (option\_drink > 6);

}

switch (option\_drink)

{

case DRINK\_MENU::COCA\_COLA:

{

productName = "Coca\_Cola";

price = check\_price(productName);

get\_detail(rd, productName, price, i, j, st);

}break;

case DRINK\_MENU::DRINKING\_WATER:

{

productName = "Drinking\_Water";

price = check\_price(productName);

get\_detail(rd, productName, price, i, j, st);

}break;

case DRINK\_MENU::SPRITE:

{

productName = "Sprite";

price = check\_price(productName);

get\_detail(rd, productName, price, i, j, st);

}break;

case DRINK\_MENU::STING:

{

productName = "Sting";

price = check\_price(productName);

get\_detail(rd, productName, price, i, j, st);

}break;

case DRINK\_MENU::BACK\_TO\_FOOD\_MENU: {}break;

case DRINK\_MENU::BACK\_TO\_SUB\_ORDER\_MENU:

{

option\_drink = 0;

option\_menu = 0;

}break;

default:

{

show\_message("\tInvalid!!!\n");

}break;

}

get\_character("\tPress any key to counitue...");

} while (option\_drink != 0);

}

void sub\_update\_menu(struct Order rd[], int i, float kh, float vat\_percentage, string st)

{

int option;

int k;

int j = rd[i].n[i];

enum UPDATE\_MENU

{

NAME = 1, QTY, QTY\_AND\_NAME

};

if (st != "order\_update")

{

system("cls");

}

do

{

update\_menu();

option = get\_option("sub\_update\_menu", "");

} while (option > 3 || option == 0);

switch (option)

{

case UPDATE\_MENU::NAME:

{

int neutron;

neutron = get\_numbers("\tPlease input neutron(No): ");

k = sub\_search(neutron, j);

if (k != -1)

{

show\_message("\tSearch is found!!!\n");

sub\_menu(rd, i, k, option, "order\_update");

payment\_update(rd, i, kh, vat\_percentage);

}

else

{

show\_message("\tSearch not found!!!\n");

}

}break;

case UPDATE\_MENU::QTY:

{

int neutron;

float price = 0;

neutron = get\_numbers("\tPlease input neutron(No): ");

k = sub\_search(neutron, j);

if (k != -1)

{

show\_message("\tSearch is found!!!");

price = check\_price(rd[i].name[k]);

update(rd, i, k, price);

payment\_update(rd, i, kh, vat\_percentage);

show\_message("\tUpdate successfully!!!\n");

}

else

{

show\_message("\tSearch not found!!!\n");

}

}break;

case UPDATE\_MENU::QTY\_AND\_NAME:

{

int neutron;

float price = 0;

neutron = get\_numbers("\tPlease input neutron(No): ");

k = sub\_search(neutron, j);

if (k != -1)

{

show\_message("\tSearch is found!!!\n");

price = check\_price(rd[i].name[k]);

update(rd, i, k, price);

sub\_menu(rd, i, k, option, "order\_update");

payment\_update(rd, i, kh, vat\_percentage);

}

else

{

show\_message("\tSearch not found!!!\n");

}

}break;

}

}

void sub\_order\_update\_menu(struct Order rd[], int i, int j, string st, int& option\_order)

{

int option;

int k;

enum UPDATE\_MENU

{

BACK\_TO\_SUB\_ORDER\_MENU = 0, UPDATE\_NAME, UPDATE\_QTY, UPDATE\_QTYAND\_NAME

};

do

{

update\_menu();

option = get\_option("sub\_update\_menu", "");

} while (option > 3);

switch (option)

{

case UPDATE\_MENU::UPDATE\_NAME:

{

int neutron;

neutron = get\_numbers("\tPlease input neutron(No): ");

k = sub\_search(neutron, j);

if (k != -1)

{

show\_message("\tSearch is found!!!\n");

sub\_menu(rd, i, k, option, st);

}

else

{

show\_message("\tSearch not found!!!\n");

}

}break;

case UPDATE\_MENU::UPDATE\_QTY:

{

int neutron;

float price = 0;

neutron = get\_numbers("\tPlease input neutron(No): ");

k = sub\_search(neutron, j);

if (k != -1)

{

show\_message("\tSearch is found!!!\n");

price = check\_price(rd[i].name[k]);

update(rd, i, k, price);

show\_message("\tUpdate successfully!!!\n");

}

else

{

show\_message("\tSearch not found!!!\n");

}

}break;

case UPDATE\_MENU::UPDATE\_QTYAND\_NAME:

{

int neutron;

float price = 0;

neutron = get\_numbers("\tPlease input neutron(No): ");

k = sub\_search(neutron, j);

if (k != -1)

{

show\_message("\tSearch is found!!!\n");

price = check\_price(rd[i].name[k]);

update(rd, i, k, price);

sub\_menu(rd, i, k, option, st);

}

else

{

show\_message("\tSearch not found!!!\n");

}

}break;

case UPDATE\_MENU::BACK\_TO\_SUB\_ORDER\_MENU: {}break;

}

}

void order\_update\_menu(struct Order rd[], int i, float kh, float vat\_percentage, float vat)

{

int updateOption;

enum UPDATE\_BY

{

BY\_ID = 1, BY\_PHONENUMBER = 2

};

do

{

cout << "\t==========UPDATE MENU==========\n" << endl;

search\_menu();

updateOption = get\_option("update\_menu", "");

} while (updateOption > 2);

switch (updateOption)

{

case UPDATE\_BY::BY\_ID:

{

string id;

bool found = false;

id = getString("\tPlease iput id: ", "id");

for (int k = 0; k < i; k++)

{

if (rd[k].inform.id == id)

{

system("cls");

show\_message("\tSearch is found!!!\n");

all\_order\_view(rd, k, vat);

sub\_update\_menu(rd, k, kh, vat\_percentage, "order\_update");

found = true;

break;

}

}

if (found == false)

{

show\_message("\tSearch not found!!!\n");

}

}break;

case UPDATE\_BY::BY\_PHONENUMBER:

{

string phone;

bool found = false;

phone = getString("\tPlease input phone number: ", "phoneNumber");

for (int k = 0; k < i; k++)

{

if (rd[k].inform.phone\_number == phone)

{

system("cls");

show\_message("\tSearch is found!!!\n");

all\_order\_view(rd, k, vat);

sub\_update\_menu(rd, k, kh, vat\_percentage, "order\_update");

found = true;

break;

}

}

if (found == false)

{

show\_message("\tSearch not found!!!\n");

}

}break;

}

}

void order\_type\_menu(struct Order rd[], int i)

{

int option\_type;

enum SALTTPYE

{

DINE\_IN = 1, TAKEAWAY = 2, DELIVERY = 3

};

do

{

system("cls");

sale\_typeMenu();

option\_type = get\_option("sale\_typeMenu", "");

} while (option\_type > 3 || option\_type == 0);

switch (option\_type)

{

case SALTTPYE::DINE\_IN:

{

get\_inform(rd, i, "DINE IN");

}break;

case SALTTPYE::TAKEAWAY:

{

get\_inform(rd, i, "TAKE AWAY");

}break;

case SALTTPYE::DELIVERY:

{

get\_inform(rd, i, "DELIVERY");

}break;

}

}

void topping\_menu(string st)

{

cout << "\t=========TOPPING MENU=========" << endl;

cout << "\t1.Chicken" << endl;

cout << "\t2.Seafood" << endl;

cout << "\t3.Sausage" << endl;

if (st == "order\_update")

{

cout << "\t0.Back to FOOD MENU" << endl;

cout << "\tPlease choose one option(1-3):";

}

else

{

cout << "\t4.Back to SUB ORDER MENU" << endl;

cout << "\t0.Back to FOOD MENU" << endl;

cout << "\tPlease choose one option(1-4):";

}

}

void size\_menu()

{

cout << "\t===========SIZE MENU===========" << endl;

cout << "\t1.Large" << endl;

cout << "\t2.Medium" << endl;

cout << "\t3.Small" << endl;

cout << "\t4.Back to FOOD MENU" << endl;

cout << "\t0.Back to TOPPING MENU" << endl;

cout << "\tPleae choose one option(1-4):";

}

void drink\_menu(string st)

{

cout << "\t==========DRINKS MENU==========" << endl;

cout << "\t1.Sting" << endl;

cout << "\t2.Coca Cola" << endl;

cout << "\t3.Sprite" << endl;

cout << "\t4.Drinking Water" << endl;

if (st == "order\_update")

{

cout << "\t0.Back to FOOD MENU" << endl;

cout << "\tPlease choose one option(1-4):";

}

else

{

cout << "\t5.Back to SUB ORDER MENU" << endl;

cout << "\t0.Back to FOOD MENU" << endl;

cout << "\tPlease choose one option(1-5):";

}

}

void sub\_orderMenu()

{

cout << "\t=======SUB ORDER MENU=======" << endl;

cout << "\t1.Order" << endl;

cout << "\t2.View order" << endl;

cout << "\t3.Update order" << endl;

cout << "\t4.Search order" << endl;

cout << "\t5.Delete order" << endl;

cout << "\t0.Back to MAIN MENU" << endl;

cout << "\tPlease choose one option(1-5): ";

}

void food\_menu(string st)

{

cout << "\t=====FOOD MENU=====" << endl;

cout << "\t1.Pizza" << endl;

cout << "\t2.Drinks" << endl;

if (st == "order\_update")

{

cout << "\t0.Back to MAIN MENU" << endl;

cout << "\tPlease choose one option(1-2):";

}

else if (st == "update")

{

cout << "\t0.Back to SUB ORDER MENU" << endl;

cout << "\tPlease choose one option(1-2):";

}

else

{

cout << "\t3.Back to MAIN MENU" << endl;

cout << "\t0.Back to SUB ORDER MENU" << endl;

cout << "\tPlease choose one option(1-3):";

}

}

void main\_menu()

{

cout << "\t===========MAIN MENU===========" << endl;

cout << "\t1.Order" << endl;

cout << "\t2.View order" << endl;

cout << "\t3.Update order" << endl;

cout << "\t4.Search order" << endl;

cout << "\t5.Delete order" << endl;

cout << "\t6.Sort order" << endl;

cout << "\t7.Save order to a file" << endl;

cout << "\t8.Load order from a file" << endl;

cout << "\t0.Exit" << endl;

cout << "\tPlease choose one option (1-8): ";

}

void delete\_order\_menu(struct Order rd[], int& i)

{

int deleteOption;

enum DELETED

{

BY\_ID = 1, BY\_PHONENUMBER = 2

};

do

{

cout << "\t=========DELETE MENU=========\n" << endl;

search\_menu();

deleteOption = get\_option("delete\_menu", "");

} while (deleteOption > 2);

switch (deleteOption)

{

case DELETED::BY\_ID:

{

string id;

int k;

char ch;

id = getString("\tPlease input id to delete: ", "id");

k = search(rd, i, "id", id);

if (k > -1) {

show\_message("\tSearch is found!!!\n");

show\_message("\tAre you sure to delete (Y/N)? ");

cin >> ch;

if (deletes(rd, k, i, ch)) {

show\_message("\tDelete successfully!!!\n");

}

}

else show\_message("\tSearch not found!!!\n");

}

break;

case DELETED::BY\_PHONENUMBER:

{

string phoneNumber;

int k;

char ch;

phoneNumber = getString("\tPlease input id to delete: ", "phoneNumber");

k = search(rd, i, "phoneNumber", phoneNumber);

if (k > -1) {

show\_message("\tSearch is found!!!\n");

show\_message("\tAre you sure to delete (Y/N)?");

cin >> ch;

if (deletes(rd, k, i, ch)) {

show\_message("\tDelete successfully!!!\n");

}

}

else show\_message("\tSearch not found!!!\n");

}break;

}

}

void search\_order(struct Order rd[], int i, int vat)

{

enum SEARCH

{

BY\_ID = 1, BY\_PHONENUMBER = 2

};

int searchOption;

do

{

show\_message("\t=========SEARCH MENU=========\n");

search\_menu();

searchOption = get\_option("search\_menu", "");

} while (searchOption > 2);

switch (searchOption)

{

case SEARCH::BY\_ID:

{

string id;

int k;

id = getString("\tPlease input id: ", "id");

k = search(rd, i, "id", id);

if (k > -1) {

show\_message("\tSearch is found!!!\n");

all\_order\_view(rd, k, vat);

}

else show\_message("\tSearch not found!!!\n");

}break;

case SEARCH::BY\_PHONENUMBER:

{

string phoneNumber;

int k;

phoneNumber = getString("\tPlease input phone number: ", "phoneNumber");

k = search(rd, i, "phoneNumber", phoneNumber);

if (k > -1) {

show\_message("\tSearch is found!!!\n");

all\_order\_view(rd, k, vat);

}

else show\_message("\tSearch not found!!!\n");

}break;

}

}

void reciept()

{

cout << endl;

cout << right << setw(76) << "The pizza resturant: #107, Phoum 4 ANZ Road" << endl;

cout << right << setw(67) << "Sangkat Boeung Kakti 2," << endl;

cout << right << setw(72) << "Khan Toul Kork Phnom Penh, 120408" << endl;

cout << right << setw(71) << "Call & Pick up : 088 50 60 697" << endl;

cout << right << setw(80) << "-------------------------------------------------" << endl;

}

void reciept(struct Order rd[], int i, int vat)

{

cout << right << setw(61) << "Receiption " << endl;

cout << right << setw(80) << "-------------------------------------------------" << endl;

cout << right << setw(38) << "Slip: "

<< "000000000C92000000" << i << endl;

cout << right << setw(38) << "Staff: " << rd[i].inform.staff << endl;

cout << right << setw(38) << "Date: " << rd[i].date.day << "/" << rd[i].date.month << "/" << rd[i].date.year - 2000 << " " << rd[i].date.hour << ":" << rd[i].date.minute << endl;

if (rd[i].inform.type\_sale == "TAKE AWAY")

{

cout << right << setw(38) << "ID: " << rd[i].inform.id << endl;

cout << right << setw(47) << "PHONE NUMBER: " << rd[i].inform.phone\_number << endl;

}

else if (rd[i].inform.type\_sale == "DELIVERY")

{

cout << right << setw(38) << "ID: " << rd[i].inform.id << endl;

cout << right << setw(47) << "PHONE NUMBER: " << rd[i].inform.phone\_number << endl;

cout << right << setw(41) << "Adderess: " << rd[i].inform.address << endl;

}

else

{

cout << right << setw(35) << "ID: " << rd[i].inform.id << endl;

cout << right << setw(38) << "Table: " << rd[i].inform.table\_numbers << endl;

}

cout << right << setw(43) << "Sales Type: " << left << setw(7) << rd[i].inform.type\_sale << endl;

cout << right << setw(80) << "-------------------------------------------------" << endl;

cout << right << setw(64) << "\*\* PRINT BILL \*\* " << endl;

cout << right << setw(80) << "-------------------------------------------------" << endl;

cout << right << setw(80) << "No. Description Amount " << endl;

cout << endl;

for (int l = 0; l < rd[i].n[i]; l++)

{

if (l < 10)

{

cout << right << setw(32) << "0" << l + 1;

cout << right << setw(7 + rd[i].name[l].length()) << rd[i].name[l] << endl;

cout << right << setw(38) << rd[i].qty[l] << " pcs \* " << left << setw(15) << rd[i].retail\_price[l];

cout << right << setw(15) << rd[i].amount[l] << " $" << left << setw(10) << endl;

}

else

{

cout << right << setw(32) << l + 1;

cout << right << setw(7 + rd[i].name[l].length()) << rd[i].name[l] << endl;

cout << right << setw(38) << rd[i].qty[l] << " pcs \* " << left << setw(15) << rd[i].retail\_price[l];

cout << right << setw(15) << rd[i].amount[l] << " $" << left << setw(10) << endl;

}

}

cout << endl;

cout << right << setw(80) << "-------------------------------------------------" << endl;

cout << right << setw(80) << "-------------------------------------------------" << endl;

cout << right << setw(40) << "Total USD" << left << setw(20) << right << setw(35) << rd[i].total\_usd << " $" << left << setw(12) << endl;

cout << right << setw(40) << "Total KHR" << left << setw(20) << right << setw(35) << fixed << setprecision(2) << rd[i].total\_kh << " R" << left << setw(12) << endl;

cout << right << setw(80) << "-------------------------------------------------" << endl;

cout << right << setw(80) << " VAT% Net.Amt VAT Amount" << endl;

cout << right << setw(32) << "$" << left << setw(2) << " " << vat;

cout << left << setw(7) << " " << left << setw(8) << rd[i].net\_amt << left << setw(8) << " " << left << setw(8) << rd[i].vat << right << setw(13) << rd[i].total\_usd << endl;

cout << right << setw(80) << "-------------------------------------------------" << endl;

cout << right << setw(71) << "Thank you, See you again soon!" << endl;

}

void reciept\_detail(struct Order rd[], string product\_name, float price, int i, int& j, string st)

{

if (st == "update")

{

rd[i].name[j] = product\_name;

rd[i].retail\_price[j] = price;

rd[i].total\_usd = rd[i].total\_usd - rd[i].amount[j];

rd[i].amount[j] = get\_amount(rd[i].qty[j], price);

rd[i].total\_usd = get\_total(rd[i].total\_usd, rd[i].amount[j]);

}

else

{

rd[i].name[j] = product\_name;

rd[i].retail\_price[j] = price;

rd[i].qty[j] = get\_numbers(st);

rd[i].amount[j] = get\_amount(rd[i].qty[j], price);

rd[i].total\_usd = get\_total(rd[i].total\_usd, rd[i].amount[j]);

j++;

}

}

void update\_menu()

{

show\_message("\t=========UPDATE MENU=========\n");

cout << "\t1.Update name" << endl;

cout << "\t2.Update qty" << endl;

cout << "\t3.Update name and qty" << endl;

cout << "\t0.Back" << endl;

cout << "\tPlease choose one option(1-3): ";

}

void sale\_typeMenu()

{

cout << "\t========SALE TYPE MENU========" << endl;

cout << "\t1.DINE IN" << endl;

cout << "\t2.Take Away" << endl;

cout << "\t3.Delivery" << endl;

cout << "\tPlease choose one option(1-3): ";

}

void search\_menu()

{

cout << "\t1.BY ID" << endl;

cout << "\t2.By PhoneNumber" << endl;

cout << "\t0.Back" << endl;

cout << "\tPlease choose one option(1-2): ";

}

void writefile(struct login account)

{

ofstream writeFile;

writeFile.open("login.bat", ios::out | ios::binary);

writeFile.write(reinterpret\_cast<char\*>(&account), sizeof(account));

writeFile.close();

}

void change\_password(struct login& account)

{

bool flag;

string currentPassword;

string newPassword;

string confirmPassword;

do

{

system("cls");

intro\_changePassword();

gotoxy(20, 9);

clear\_text(28, 9);

currentPassword = get\_passwordInput(20, 10, 20);

gotoxy(20, 12);

clear\_text(28, 12);

newPassword = get\_passwordInput(20, 13, 20);

gotoxy(20, 15);

clear\_text(26, 15);

confirmPassword = get\_passwordInput(20, 16, 20);

flag = verify\_newPassword(account, currentPassword, newPassword, confirmPassword);

\_getch();

} while (flag != true);

}

void gotoxy(short x, short y)

{

COORD pos = { x, y };

SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), pos);

}

void setcolor(int Color)

{

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), Color);

}

bool verify(struct login account, string username, string password, int x, int y)

{

if (password[0] == 0)

{

gotoxy(x, y + 6);

setcolor(12);

show\_message("Please input your password...\n");

\_getch();

}

if (username == account.username && password == account.password)

{

gotoxy(x, y + 6);

setcolor(2);

show\_message("Successfully!!!\n");

\_getch();

return true;

}

else

{

gotoxy(x, y + 6);

setcolor(12);

show\_message("Sorry,please try again!!!\n");

\_getch();

return false;

}

}

string get\_name(int x, int y)

{

string name;

bool flag;

do

{

clear\_text(x + 13, y + 1);

gotoxy(x, y + 1);

getline(cin, name);

if (check\_string(name))

{

flag = true;

}

else

{

gotoxy(x, y + 8);

setcolor(12);

show\_message("Please input your name...");

flag = false;

\_getch();

system("cls");

front\_display(x, y);

}

} while (!flag);

return name;

}

bool check\_string(string name)

{

if (name[0] == NULL)

{

return false;

}

return true;

}

string get\_password(int x, int y)

{

char password[20] = { 0 };

char str;

bool flag;

int i = 0;

gotoxy(x, y + 5);

clear\_text(x + 13, y + 5);

do

{

if ((str = \_getch()))

{

if (str == 13)

{

break;

}

else if (check\_character(str, x, y, i))

{

password[i] = str;

i++;

flag = true;

}

else

{

flag = true;

}

}

} while (flag);

password[i] = '\0';

return password;

}

void front\_display(int x, int y)

{

setcolor(15);

setcolor(3);

gotoxy(x, 5);

cout << ("LOGIN");

setcolor(15);

gotoxy(x, y);

cout << ("Username");

gotoxy(x, y + 1);

setcolor(8);

cout << ("Enter username");

setcolor(15);

gotoxy(x, y + 3);

cout << ("Password");

gotoxy(x, y + 5);

setcolor(8);

cout << ("Enter Password");

gotoxy(x, y + 1);

}

void clear\_text(int x, int y)

{

if (\_getch() || \_getch() == 8)

{

for (int i = 20; i < 34; i++)

{

gotoxy(x--, y);

show\_message(" ");

}

}

}

bool check\_character(char str, int& x, int& y, int& i)

{

if (str == 8)

{

if (x > 20)

{

gotoxy(x--, y + 5);

show\_message(" ");

i--;

}

else if (x == 20)

{

gotoxy(x--, y + 5);

show\_message(" ");

x++;

}

return false;

}

else

{

gotoxy(x, y + 5);

show\_message("\*");

x++;

return true;

}

}

bool check\_passwordInput(char str, int& x, int& y, int& i, int x1)

{

if (str == 8)

{

if (x > x1)

{

gotoxy(x--, y - 1);

show\_message(" ");

i--;

}

else if (x == x1)

{

gotoxy(x--, y - 1);

show\_message(" ");

x++;

}

return false;

}

else

{

gotoxy(x, y - 1);

show\_message("\*");

x++;

return true;

}

}

void intro\_changePassword()

{

setcolor(15);

setcolor(3);

gotoxy(20, 5);

cout << ("Change password");

setcolor(15);

gotoxy(20, 8);

cout << ("Current password");

setcolor(8);

gotoxy(20, 9);

cout << "Password";

setcolor(15);

gotoxy(20, 11);

cout << ("New password");

setcolor(8);

gotoxy(20, 12);

cout << "Password";

setcolor(15);

gotoxy(20, 14);

cout << ("Confirm new password");

setcolor(8);

gotoxy(20, 15);

cout << "Cofirm";

}

void get\_detail(struct Order rd[], float kh, float vat\_percentage, int i, int j) {

order\_type\_menu(rd, i);

get\_total\_kh(rd, i, kh);

get\_net\_amt(rd, i, vat\_percentage);

get\_vat(rd, i, vat\_percentage);

}

void view(struct Order rd[], int i, int vat) {

for (int k = 0; k < i; k++)

{

all\_order\_view(rd, k, vat);

cout << endl;

cout << endl;

}

}

int search(struct Order rd[], int i, string str, string st) {

if (str == "id") {

for (int k = 0; k < i; k++)

{

if (rd[k].inform.id == st)

{

return k;

}

}

}

else {

for (int k = 0; k < i; k++)

{

if (rd[k].inform.phone\_number == st)

{

return k;

}

}

}

return -1;

}

float check\_price(string topping)

{

// Pizza

// Small Prices

const float S\_CHICKEN\_PRICE = 8.99;

const float S\_SEAFOOD\_PRICE = 8.99;

const float S\_SAUSAGE\_PRICE = 8.99;

// Large Prices

const float L\_CHICKEN\_PRICE = 19.99;

const float L\_SEAFOOD\_PRICE = 19.99;

const float L\_SAUSAGE\_PRICE = 17.85;

// Medium Prices

const float M\_CHICKEN\_PRICE = 13.99;

const float M\_SEAFOOD\_PRICE = 13.99;

const float M\_SAUSAGE\_PRICE = 12.28;

// Drinks Prices

const float STING\_PRICE = 1.99;

const float COCA\_COLA\_PRICE = 2.00;

const float SPRITE\_PRICE = 1.50;

const float DRINKING\_WATER\_PRICE = 1.20;

if (topping == "L-Chicken")

{

return L\_CHICKEN\_PRICE;

}

else if (topping == "M-Chicken")

{

return M\_CHICKEN\_PRICE;

}

else if (topping == "S-Chicken")

{

return S\_CHICKEN\_PRICE;

}

else if (topping == "L-Sausage")

{

return L\_SAUSAGE\_PRICE;

}

else if (topping == "M-Sausage")

{

return M\_SAUSAGE\_PRICE;

}

else if (topping == "S-Sausage")

{

return S\_SAUSAGE\_PRICE;

}

else if (topping == "L-Seafood")

{

return L\_SEAFOOD\_PRICE;

}

else if (topping == "M-Seafood")

{

return M\_SEAFOOD\_PRICE;

}

else if (topping == "S-Seafood")

{

return S\_SEAFOOD\_PRICE;

}

else if (topping == "Coca\_Cola") {

return COCA\_COLA\_PRICE;

}

else if (topping == "Drinking\_Water") {

return DRINKING\_WATER\_PRICE;

}

else if (topping == "Sprite") {

return SPRITE\_PRICE;

}

else if (topping == "Sting") {

return STING\_PRICE;

}

}

void repet\_show(string option\_type, string str)

{

show\_message("\tError input,please try again\n");

get\_character("\tpress any key to continue...\n");

if (option\_type == "main\_menu")

{

system("cls");

main\_menu();

}

else if (option\_type == "food\_menu")

{

system("cls");

food\_menu(str);

}

else if (option\_type == "sub\_orderMenu")

{

system("cls");

sub\_orderMenu();

}

else if (option\_type == "topping\_menu")

{

system("cls");

pizza(str);

}

else if (option\_type == "size\_menu")

{

system("cls");

size\_menu();

}

else if (option\_type == "drink\_menu")

{

system("cls");

drink\_menu(str);

}

else if (option\_type == "sub\_update\_menu")

{

cout << endl;

update\_menu();

}

else if (option\_type == "sale\_typeMenu")

{

system("cls");

sale\_typeMenu();

}

else if (option\_type == "search\_menu")

{

cout << endl;

show\_message("\t=========SEARCH MENU=========\n");

search\_menu();

}

else if (option\_type == "delete\_menu")

{

cout << endl;

show\_message("\t=========DELETE MENU=========\n");

search\_menu();

}

else if (option\_type == "update\_menu")

{

cout << endl;

show\_message("\t==========UPDATE MENU==========\n");

search\_menu();

}

else {

system("cls");

login\_menu();

}

}

void update(struct Order rd[], int i, int k, float price)

{

rd[i].qty[k] = get\_numbers("\tEnter qty: ");

rd[i].total\_usd = rd[i].total\_usd - rd[i].amount[k];

rd[i].amount[k] = get\_amount(rd[i].qty[k], price);

rd[i].total\_usd = get\_total(rd[i].total\_usd, rd[i].amount[k]);

}

void sub\_delete(struct Order rd[], int k, int& j, int i)

{

for (int l = k; l < j; l++)

{

swap1(rd, i, l);

}

j = j - 1;

}

void swap1(struct Order rd[], int i, int l) {

rd[i].name[l] = rd[i].name[l + 1];

rd[i].qty[l] = rd[i].qty[l + 1];

rd[i].retail\_price[l] = rd[i].retail\_price[l + 1];

rd[i].amount[l] = rd[i].amount[l + 1];

}

string getString(string st, string str)

{

string get;

bool found = false;

do

{

cout << st;

cin.sync();

getline(cin, get);

if (str == "staffName") {

found = validat\_staff(get);

if (!found)

{

show\_message("\tstaff name must be 3 character contain letter and latin,Please try again!!!\n");

get\_character("\tPress any key to continue...\n");

system("cls");

}

}

else if (str == "phoneNumber") {

found = validat\_phoneNumber(get);

if (!found)

{

show\_message("\tphone number must be 8 character contain number ,Please try again!!!\n");

get\_character("\tPress any key to continue...\n");

}

}

else if (str == "address") {

found = true;

}

else if (str == "id") {

found = validat\_id(get);

if (!found)

{

show\_message("\tID must be 6 character contain number,Please try again!!!\n");

get\_character("\tPress any key to continue...\n");

}

}

} while (!found);

return get;

}

bool validat\_id(string st)

{

if (st[0] == NULL)

{

return false;

}

else if (st.length() < 6)

{

return false;

}

for (int i = 0; i < st.length(); i++)

{

if ((st[i] >= '0' && st[i] <= '9'))

{

continue;

}

else

{

return false;

}

}

return true;

}

bool validat\_staff(string st)

{

if ((st[0] == NULL) || (!(st[0] >= 'A' && st[0] <= 'Z')))

{

return false;

}

else if (st.length() < 3)

{

return false;

}

for (int i = 0; i < st.length(); i++)

{

if ((st[i] >= 65 && st[i] <= 122) || st[i] == ' ')

{

continue;

}

else

{

return false;

}

}

return true;

}

bool validat\_phoneNumber(string st)

{

if (st[0] == NULL)

{

return false;

}

else if (st.length() < 8)

{

return false;

}

for (int i = 0; i < st.length(); i++)

{

if ((st[i] >= '0' && st[i] <= '9'))

{

continue;

}

else

{

return false;

}

}

return true;

}

void order\_view(struct Order rd[], int n, int i)

{

for (int k = 0; k < n; k++)

{

display(rd, k, i);

}

cout << endl;

}

void display(struct Order rd[], int k, int i)

{

if (k < 10)

{

cout << right << setw(32) << "0" << k + 1 << left << setw(16) << " ";

cout << left << setw(18) << rd[i].name[k];

cout << right << setw(7) << rd[i].qty[k] << " \* pcs" << endl;

}

else

{

cout << right << setw(32) << k + 1 << left << setw(16) << " ";

cout << left << setw(18) << rd[i].name[k];

cout << right << setw(7) << rd[i].qty[k] << " \* pcs" << endl;

}

cout << endl;

}

void show\_message(string st)

{

cout << st;

}

int sub\_search(int neutron, int j)

{

for (int k = 0; k < j; k++)

{

if (neutron - 1 == k)

{

return k;

}

}

return -1;

}

int get\_numbers(string st)

{

int number;

string str;

bool found;

do

{

show\_message(st);

cin.sync();

getline(cin, str);

found = validat\_number(str);

if (found)

{

number = stoi(str);

}

else

{

show\_message("\tError input,please try again\n");

get\_character("\tpress any key to continue...\n");

cout << endl;

}

} while (!found);

return number;

}

bool validat\_number(string st)

{

if (st[0] == NULL)

{

return false;

}

for (int i = 0; i < st.length(); i++)

{

if ((st[i] >= '0' && st[i] <= '9'))

{

continue;

}

else

{

return false;

}

}

return true;

}

void get\_total\_kh(struct Order rd[], int i, float kh)

{

rd[i].total\_kh = rd[i].total\_usd \* kh;

}

void get\_vat(struct Order rd[], int i, float percentage)

{

rd[i].vat = rd[i].total\_usd \* percentage;

}

void get\_net\_amt(struct Order rd[], int i, float percentage)

{

rd[i].net\_amt = rd[i].total\_usd - (percentage \* rd[i].total\_usd);

}

int get\_option(string option\_type, string str)

{

int opTion;

string st;

bool found;

do

{

cin.sync();

getline(cin, st);

found = validat\_option(st);

if (found)

{

opTion = stoi(st);

}

else

{

repet\_show(option\_type, str);

}

} while (!found);

return opTion;

}

bool validat\_option(string st)

{

if (st[0] == NULL)

{

return false;

}

for (int i = 0; i < st.length(); i++)

{

if (st[i] >= '0' && st[i] <= '9')

{

continue;

}

else

{

return false;

}

}

return true;

}

float get\_amount(int qty, float price)

{

return qty \* price;

}

float get\_total(float total, float amount)

{

return total + amount;

}

void pizza(string st)

{

topping\_menu(st);

}

void all\_order\_view(struct Order rd[], int i, int vat)

{

cout << endl;

reciept();

reciept(rd, i, vat);

}

void swap(struct Order& rd1, struct Order& rd2)

{

struct Order temp;

temp = rd1;

rd1 = rd2;

rd2 = temp;

}

void sort\_order(struct Order rd[], int i)

{

for (int k = 0; k < i - 1; k++)

{

for (int l = k; l < i; l++)

{

if (rd[k].total\_usd < rd[l].total\_usd)

{

swap(rd[k], rd[l]);

}

}

}

}

bool deletes(struct Order rd[], int k, int& i, char ch)

{

if (ch == 'y' || ch == 'Y')

{

for (int l = k; l < i; l++)

{

swap(rd[l], rd[l + 1]);

}

i = i - 1;

return true;

}

else

{

return false;

}

}

void get\_date(struct Order rd[], int i)

{

time\_t now = time(NULL);

tm\* ltm = localtime(&now);

rd[i].date.hour = (1 + ltm->tm\_hour) - 1;

rd[i].date.minute = (1 + ltm->tm\_min) - 1;

rd[i].date.month = (1 + ltm->tm\_mon);

rd[i].date.year = (1 + ltm->tm\_year + 1900) - 1;

rd[i].date.day = (1 + ltm->tm\_mday) - 1;

}

void get\_character(string st)

{

show\_message(st);

\_getch();

}

void payment\_update(struct Order rd[], int i, float kh, float vat\_percentage)

{

get\_total\_kh(rd, i, kh);

get\_net\_amt(rd, i, vat\_percentage);

get\_vat(rd, i, vat\_percentage);

}

string get\_usernameInput(int x, int y)

{

string name;

bool flag;

do

{

gotoxy(x, y - 1);

getline(cin, name);

if (check\_string(name))

{

flag = true;

}

else

{

gotoxy(x, y + 2);

setcolor(12);

show\_message("Please input your name...");

flag = false;

\_getch();

system("cls");

}

} while (!flag);

return name;

}

string get\_passwordInput(int x, int y, int x1)

{

char password[20] = { 0 };

char str;

bool flag;

int i = 0;

gotoxy(x, y - 1);

do

{

if ((str = \_getch()))

{

if (str == 13)

{

break;

}

else if (check\_passwordInput(str, x, y, i, x1))

{

password[i] = str;

i++;

flag = true;

}

else

flag = true;

}

} while (flag);

password[i] = '\0';

return password;

}

void get\_inform(struct Order rd[], int i, string type) {

rd[i].inform.type\_sale = type;

rd[i].inform.staff = getString("\tInput staff name: ", "staffName");

rd[i].inform.id = getString("\tInput id: ", "id");

if (type == "DINE IN") {

rd[i].inform.table\_numbers = get\_numbers("\tInput table: ");

}

else if (type == "TAKE AWAY") {

rd[i].inform.phone\_number = getString("\tInput phone number: ", "phoneNumber");

}

else {

rd[i].inform.phone\_number = getString("\tInput phone number: ", "phoneNumber");

rd[i].inform.address = getString("\tInput address: ", "address");

}

get\_date(rd, i);

system("cls");

}

void intro\_reciept\_header() {

reciept();

cout << right << setw(80) << "No. Description Qty" << endl;

}

void get\_detail(struct Order rd[], string productName, float price, int i, int& j, string st) {

if (st == "update")

{

reciept\_detail(rd, productName, price, i, j, st);

show\_message("\tUpdate successfully!!!\n");

}

else if (st == "order\_update")

{

reciept\_detail(rd, productName, price, i, j, "update");

show\_message("\tUpdate successfully!!!\n");

st = "order\_update";

}

else

{

reciept\_detail(rd, productName, price, i, j, "\tEnter qty: ");

}

}

bool verify\_newPassword(struct login& account, string currentPassword, string newPassword, string confirmPassword) {

if (currentPassword != account.password)

{

setcolor(12);

gotoxy(20, 16);

show\_message("Currentpassword incorrect!!!");

return false;

}

else if (newPassword != confirmPassword)

{

setcolor(12);

gotoxy(20, 16);

show\_message("Password not match!!!");

return false;

}

else if ((currentPassword == account.password) && (newPassword == confirmPassword))

{

setcolor(2);

gotoxy(20, 16);

show\_message("Password updated!!!");

account.password = newPassword;

return true;

}

else

{

setcolor(12);

gotoxy(20, 16);

show\_message("Invalid!!!");

return false;

}

}

void login\_menu() {

cout << "\t===== WELCOME TO THE PIZZA RESTRAURANT =====\n" << endl;

cout << "\t1.Register" << endl;

cout << "\t2.LOGIN" << endl;

cout << "\t3.CHANGE PASSWORD" << endl;

cout << "\t0.EXIT" << endl;

cout << "\tPlease choice one option:";

}