School of Computing and Information Technology, University of Technology Jamaica

CMP1025: Programming 2

Jonathan C. Blackwood - 2306822

Chevannese M. Ellis – 2301109

Gabrielle Tucker – 2206681

Shanoi Walker-Gayle - 1808177

Occurrence: UN1 and UE2

Oneil Charles

Mrs Bowen Mighty

Due Date: April 12, 2024

Table of Contents

[Report 3](#_Toc163943905)

[Pseudocode 7](#_Toc163943906)

[C Source Code 28](#_Toc163943907)

[Declaration of Authorship 68](#_Toc163943908)

[Shanoi Walker Gayle 68](#_Toc163943909)

[Gabrielle Tucker 69](#_Toc163943910)

[Chevannese Ellis 70](#_Toc163943911)

[Jonathan Blackwood 71](#_Toc163943912)

# Report

|  |  |  |
| --- | --- | --- |
| Group Member | Activity | Description |
| **Jonathan Blackwood** | Enums | Identified the importance of implementing enums such as PREFERENCE, PAYMENT\_TYPE, FUEL\_TYPE, REPORT\_TYPE, CustType to make the program more readable when navigating through the various menus. |
|  | LOGIN Function | Implemented login feature that inputs a string and denies access if the string does not match the system password. |
|  | PROMPT\_INT\_RANGE | Established a function that takes in a range of options, checks, reads, and validates that a variable is within the specified option range. It improves the efficiency of the program and reduces tremendous lines of code. |
|  | Global Variables | Identified the essence of global variables, for they make the program easier to implement the various functionalities without redeclaring variable types. |
|  | Random Numbers/Characters | Identified and implemented functions that would randomly assign values to a variable. These would be needed to generate customer receipt. The prototype function is lub\_req(). |
|  | Retrieve current date | The get\_current\_date() function which will return the current date as a string. |
|  | Structure | Identified and created ChargeCustomer structure to maintain the records of charge customers. |
| **Chevannese Ellis** | Random Generator | Identified and implemented a function that would randomly assign values to a variable. These would be needed to generate license plate number and preference respectively which are generate\_license and generate\_preference. |
|  | Structure | Identified and created Customer structure to maintain the records of cash customers. |
|  | Global Variables | Implemented global variables onto the system to make ease of declaring variables and passing them though functions. |
|  | Calculating Prototype Functions | Identified and implemented several functions that would convert a unit type to another unit type and other essential calculating functions. The prototype functions are as follows: litre, gallon, gct, convert\_gallon\_litres, accumulate\_litre, accumulate\_litres2, accumulate\_fuel, accumulate\_fuel2. |
|  | Receipt Prototypes | Identified and implemented functions that reads the collected data from user and displays a user-friendly receipt to user. These prototype functions are receipt1 and receipt2. |
|  | Core Function Prototypes | Implemented and modified existing System Functionalities function which contains other functions to allow user to perform any of the eight options in the program. The prototype functions are as follows: serve\_customer, add\_charge\_customer, update\_charge\_customer, remove\_charge\_customer, payment\_charge\_customer, refuel\_tank, generate\_report. |
|  | Retrieve Charge Customer Data | Identified the importance of implementing search functions to compare if the value inputted by user matches the record on Charge Customer. These prototype functions are search\_id and search\_reps. |
| **Gabby Tucker** | Add Charge Customer | Identified and implemented a feature that allows a new charge customer to be added on the system. |
|  | Existing Charge Customer | Identified and implemented a feature that allows users to view customers that are registered on the system. |
|  | Update Charge Customer | Identified and implemented a feature that allows admin to edit a single field, or all fields related to an existing charge customer. |
| **Shanoi Walker-Gayle** | Pseudocode | Identified and implemented the pseudocode to demonstrate how the program should be like. |
|  | Refuel Tank | Identified and implemented a feature that allows the admin to refuel the fuel quantity. |

# Pseudocode

START

DECLARE SIZE = 100 AS CONSTANT INTEGER

DECLARE chargelist = 3, serve\_list = 0, overall\_list = 0 AS INTEGER

DECLARE AS current\_charge\_id = 4 AS INTEGER

DECLARE AS chargecount AS INTEGER

DECLARE cust\_type, option, cod\_type, fuel, lub, cashcount = 0, charge\_count = 0 AS INTEGER

DECLARE money AS FLOAT

DECLARE fuel\_types[][] = {"None", "E10-87", "E10-90", "Diesel"} AS CONSTANT STRING

DECLARE lubricant\_types[][] = {"None", "5W-30", "5W-40", "15W-40", "SAE-40"} AS CONSTANT STRING

DECLARE preference\_method[][] = {"DEPOSIT", "MAXIMUM"} AS CONSTANT STRING

DECLARE lub\_price[6] = {0, 2900, 3500, 3600, 2100} AS CONSTANT STRING

DECLARE fuel\_per\_litre[5] = {0, 184.90, 193.60, 182.30} AS CONSTANT STRING

DECLARE AS Constant Float min\_money\_card = 1000.00, min\_litre\_cash = 2.00

DECLARE license[7] AS CHARACTER

DECLARE reps, id, item\_count, current\_customer AS INTEGER

DECLARE min\_deposit = 10000, max\_litres = 3785.41, min\_cheque = 5000 AS CONSTANT FLOAT

DECLARE type\_pay AS INTEGER

DECLARE payment, payment\_litre AS FLOAT

DECLARE field\_option, field\_type AS INTEGER

DECLARE FUEL\_TANK\_CAPACITY[5] = {0, 75708.23, 113562.35, 94635.29} AS CONSTANT STRING

DECLARE refuel\_per\_gallon[5] = {0, 165.64, 171.74, 159.38} AS CONSTANT STRING

DECLARE CURRENT\_FUEL\_TANK\_CAPACITY[5] = {0, 75708.23, 113562.35, 94635.29} AS STRING

DECLARE refuel\_amount AS FLOAT

DECLARE report\_option AS INTEGER

DECLARE FUEL\_SALES[5] = {0, 0, 0, 0} AS STRING

DECLARE TOTAL\_LITRES = 0 AS FLOAT

DECLARE SALES\_GRAND\_TOTAL[4] = {0, 0, 0} AS STRING

FUNCTION luq\_req() RETURN CHARACTER

CONSTANT CHARACTER lub[2] = {'Y','N'}

RETURN lub [RANDOMLY 'Y' OR 'N']

END FUNCTION

FUNCTION generate\_preference() RETURN CHARACTER

CONSTANT CHARACTER pre[2] = {'D','M'}

RETURN pre[RANDOMLY 'Y' OR 'N']

END FUNCTION

FUNCTION generate\_preference\_output(client)

DECLARE response = generate\_preference AS INTEGER

PRINT "Generating Preference: ", response

IF response == 'D' THEN

PRINT "Preference Deposit has been assigned to customer"

preference\_number = DEPOSIT

preference = preference\_method[0]

ELSE IF response == 'M' THEN

PRINT "Preference Maximum litres has been assigned to customer"

preference\_number = MAXIMUM

preference = preference\_method[0]

ENDIF

ENDIF

FUNCTION get\_current\_date() RETURN STRING

CHARACTER date[11]

CONSTANT clock = time("")

time = localtime(CLOCK)

strftime (date, 11, CURRENT\_DATE, time)

RETURN DATE

FUNCTION accumulate\_fuel\_sales() RETURN FLOAT

DO CASE fuel

CASE 1 RETURN FUEL\_SALES[1] += money

CASE 2 RETURN FUEL\_SALES[2] += money

CASE 3 RETURN FUEL\_SALES[3] += money

ENDCASE

END FUNCTION

FUNCTION accumulate\_litres() RETURN FLOAT

RETURN TOTAL\_LITRES += litre()

END FUNCTION

FUNCTION convert\_gallon\_litres(FLOAT X) RETURN FLOAT

RETURN X \* 3.785

END FUNCTION

FUNCTION gct() RETURN FLOAT

RETURN lub\_price[lub] \* 0.16

END FUNCTION

FUNCTION litre() RETURN FLOAT

DOCASE fuel

CASE 1 RETURN money / 184.9

CASE 2 RETURN money / 193.6

CASE 3 RETURN money / 182.3

ENDCASE

END FUNCTION

FUNCTION login() RETURN BOOLEAN

BEGIN

CONST PASSWORD = "bruh"

DECLARE input AS STRING

READ input

RETURN INPUT != PASSWORD

END FUNCTION

FUNCTION search\_id (client ,INTEGER key) RETURN INTEGER

FOR INTEGER i = 0 TO SIZE STEP 1

IF id == key THEN

RETURN i

ENDIF

ENDFOR

RETURN -1

END FUNCTION

FUNCTION search\_id(client AS ARRAY OF ChargeCustomer, key AS INTEGER) AS INTEGER

DECLARE i AS INTEGER

FOR INTEGER i = 0 TO SIZE STEP 1

IF clienct[i].id == key THEN

RETURN i

ENDIF

ENDFOR

RETURN -1

END FUNCTION

FUNCTION search\_reps(client, INTEGER key) RETURN INTEGER

FOR INTEGER i = 0 TO SIZE STEP 1

IF representatives[i] == key THEN

RETURN i

ENDIF

ENDFOR

RETURN -1

END FUNCTION

FUNCTION add\_charge\_customer()

PRINTF "Enter admin password: "

IF NOT login() THEN

PRINT "Incorrect password"

ELSE

PRINT "Login successful"

PRINT "New customer has been automatically assigned to id " + current\_charge\_id

id = current\_charge\_id

PRINT "Enter business name: "

READ name

reps = prompt\_int\_range("Enter the number of representatives", 1, 5, "Number must be from 1 to 5")

representatives = reps

FOR INTEGER i = 0 To reps STEP 1

generate\_license(client, i)

END FOR

generate\_preference\_output(client)

IF preference\_number == DEPOSIT Then

PRINT "Enter the initial money: $"

READ initial\_money

WHILE initial\_money < min\_deposit

PRINT "The minimum deposit is $" min\_deposit

PRINT "Enter the initial money: $"

READ initial\_money

END WHILE

Else IF preference\_number == MAXIMUM THEN

PRINT "Enter the initial litres: "

READ initial\_litres

While initial\_litres > max\_litres

PRINT "The maximum litres is $" + max\_litres

PRINT "Enter the initial money: $"

READ initial\_litres

END WHILE

END IF

PRINT "New customer has been successfully added to the system"

chargelist+=1

current\_charge\_id+=1

END IF

END FUNCTION

FUNCTION generate\_license(client, INTEGER i)

CHARACTER string[20]

FOR INTEGER i = 0 TO 4 STEP 1

string[i] = 48 + (RANDOM INTEGER % 57)

WHILE string[i] <48 OR string[i] > 57

string[i] = 48 + (RANDOM INTEGER % 57)

ENDWHILE

ENDFOR

FOR INTEGER i = 4 TO 6 STEP 1

string[i] = 65 + (RANDOM INTEGER % 90)

WHILE string[i] <65 OR string[i] > 90

string[i] = 65 + (RANDOM INTEGER % 90)

ENDWHILE

ENDFOR

licenses[i] = string

PRINT "",string "has been assigned to license number",i + 1

END FUNCTION

FUNCTION generate\_report(client)

PRINT "Enter admin password"

IF NOT login()

PRINT "Incorrect password"

ELSE

PRINT "Login successfully"

report\_option = prompt\_int\_range("Enter the number of representatives",report\_option, 1, 5, "Number must be from 1 to 5")

PRINT ""

DOCASE report\_option

CASE 1

PRINT "Stateline Gas Station Sales Report"

PRINT "Total Cash Customers: ", cashcount

PRINT "Total Charge Customers: ",chargecount

PRINT "Total Cash Amount: $",SALES\_GRAND\_TOTAL[1]

PRINT "Total Charge Amount: $", SALES\_GRAND\_TOTAL[2]

PRINT "Grand Total: $", SALES\_GRAND\_TOTAL[1] + SALES\_GRAND\_TOTAL[2]

CASE 2

PRINT "Stateline Gas Station Charge Report"

FOR INTEGER i = 0 TO 3 STEP 1

PRINT "Customer id: ", id[i]

PRINT "Customer name: ", name[i]

PRINT "Initial Deposit: $", initial\_money[i]

PRINT "Balance Remaining: $", initial\_money - money

PRINT "Initial Litres Amount: ", initial\_litres

PRINT "Amount Owned: ",

PRINT "Amount Paid: ",

PRINT "Balance Remaining: ", balance

PRINT ""

CASE 3

PRINT "Stateline Gas Station Fuel Report"

PRINT "Total Fuel Sales: $", FUEL\_SALES[1] + FUEL\_SALES[2] + FUEL\_SALES[3]

PRINT "E10-87 Sales: $", FUEL\_SALES[1]

PRINT "E10-90 Sales: $", FUEL\_SALES[2]

PRINT "Diesel Sales: $", FUEL\_SALES[3]

PRINT "Total Litres Purchased: ", TOTAL\_LITRES

ENDCASE

END FUNCTION

void prompt\_int\_range(const char \*message, int \*option

FUNCTION prompt\_int\_range (CONSTANT CHARACTER message, INTEGER option, INTEGER min, INTEGER max, CONSTANT CHARACTER error)

WHILE 1

PRINT "",message,":"

READ option

IF option >= min AND option <= max

BREAK

ELSE IF error != ""

PRINT "", error

ELSE

PRINT "Invalid option"

ENDIF

ENDIF

ENDIF

END FUNCTION

FUNCTION receipt()

PRINT ""

PRINT "Stateline Gas Station"

PRINT "Date of transaction: ", get\_current\_date()

PRINT "Customer Type: ", cust\_type

PRINT "Item count: ", item\_count

PRINT "==============================="

PRINT "Fuel Type: ", fuel\_type

PRINT "Cost: $", fuel\_amount

PRINT "Lubricant Type: ", lubricant

PRINT "Cost: $", lubricant\_amount

PRINT "==============================="

PRINT "Sub-Total: $", sub\_total

PRINT "GCT: $", gct\_amount

PRINT "Grand Total: $", grand\_total

PRINT "Cash: $", grand\_total

PRINT "Amount Tendered: $", money

PRINT "Change/Amount Owned: $", change

END FUNCTION

FUNCTION refuel\_tank\_tank()

PRINT "Enter admin password: "

IF NOT login()

PRINT "Incorrect password"

ELSE

PRINT "Login successful"

fuel = prompt\_int\_range("Enter fuel type (E10—87 = 1, E10-90 = 2, Diesel = 3)", fuel, 1, 3);

DO CASE fuel

CASE 1 refuel\_tank\_output()

CASE 2 refuel\_tank\_output()

CASE 3 refuel\_tank\_output()

ENDCASE

ENDIF

END FUNCTION

FUNCTION refuel\_tank\_output()

PRINT "Current Tank Capacity: "",CURRENT\_FUEL\_TANK\_CAPACITY[fuel], "litres"

PRINT "Enter the refuel amount in gallons or Press 0 to exit: "

READ refuel\_amount

WHILE CURRENT\_FUEL\_TANK\_CAPACITY[fuel] + convert\_gallon\_to\_litres((gallon())) > FUEL\_TANK\_CAPACITY[fuel] OR refuel\_amount < refuel\_per\_gallon[fuel]

IF refuel\_amount == 0

BREAK

ENDIF

PRINT "Higher than $",refuel\_amount,"is required"

PRINT "Enter refuel amount in gallons or Press 0 to Exit"

READ refuel\_amount

CURRENT\_FUEL\_TANK\_CAPACITY[fuel] += convert\_gallon\_to\_litres(gallon())

PRINT "Tank has been successfully refueled"

PRINT "Current Tank Capacity: ",CURRENT\_FUEL\_TANK\_CAPACITY,"litres"

END FUNCTION

FUNCTION MAIN()

WHILE 1

PRINT "1 - Serve Customer"

PRINT "2 - Add Charge Customer"

PRINT "3 - Update Charge Customer"

PRINT "4 - Delete Charge Customer"

PRINT "5 - Make Payment to Charge Account"

PRINT "6 - Refuel Tank"

PRINT "7 - Generate Report"

PRINT "8 - Exit"

PRINT "Enter a option (1-8):"

DOCASE option

CASE 1

IF overall\_list < SIZE

CALL serve\_customer(cust,client)

ELSE

PRINT "The system has reached max capacity"

ENDIF

CASE 2

IF overall\_list < SIZE

CALL add\_charge\_customer(client)

ELSE

PRINT "The system has reached max capacity"

ENDIF

CASE 3

CALL update\_customer(cust,client)

CASE 4

CALL remove\_charge\_customer(client)

CASE 5

CALL payment\_charge\_customer(client)

CASE 6

CALL refuel\_tank()

CASE 7

CALL generate\_report(client)

CASE 8

CALL saveToFileCash()

CALL savetoFileCharge()

CALL savetoFileCharge2()

ENDCASE

ENDWHILE

STOP

# C Source Code

/\*

Names of Group Members:

Occurence: UN1

Tutorial Teacher: Oniel Charles

Chevannese Ellis 2301109

Jonathan Blackwood 2306822

Gabby Tucker 2206681

Occurrences: UE2

Tutorial Teacher: Mrs. Aleisa Bowen-Mighty

Shanoi Walker-Gayle 1808177

Description: Stateline Gas Station Programming 2 Major Project

The default password to access admin features is set to bruh

\*/

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <time.h>

#include <ctype.h>

#define SIZE 100

//Declarations of Enums

enum CustType

{

CASH = 1, CHARGE = 2

};

enum FUEL\_TYPE

{

E10\_87 = 1, E10\_90 = 2, DIESEL = 3

};

enum PREFERENCE

{

DEPOSIT = 0, MAXIMUM = 1

};

enum PAYMENT\_TYPE

{

CARDCASH = 1, CHEQUE = 2

};

enum REPORT\_TYPE

{

SALES\_REPORT = 1, CHARGE\_REPORT = 2, FUEL\_REPORT = 3

};

//Declarations of Structs

typedef struct {

char cust\_type [6];

char fuel\_type[9];

float fuel\_amount;

float money;

float change;

char lubricant[6];

float lubricant\_amount;

float gct\_amount;

float sub\_total;

float grand\_total;

}Customer;

typedef struct {

int id;

char name[51];

char licenses[5][7];

int representatives;

char fuel\_type[9];

char preference[9];

char lubricant[6];

float lubricant\_amount;

float fuel\_amount;

int preference\_number;

float money;

float current\_money;

float initial\_money;

float litres;

float current\_litres;

float initial\_litres;

float gct\_amount;

float sub\_total;

float grand\_total;

float change;

}ChargeCustomer;

//Global Variables that manages the list

int chargelist = 3, serve\_list = 0, overall\_list = 0;

int current\_charge\_id = 4;

int chargecount;

//Global Variables of Serve Customer

int cust\_type, option, cod\_type, fuel, lub, cashcount = 0, charge\_count = 0;

float money;

const char fuel\_types[][7] = {"None","E10-87","E10-90","Diesel"};

const char lubricant\_types[][7] = {"None","5W-30","5W-40","15W-40","SAE-40"};

const char preference\_method[][8] = {"DEPOSIT","MAXIMUM"};

const float lub\_price[6] = {0, 2900, 3500, 3600, 2100};

const float fuel\_per\_litre[5] = {0, 184.90, 193.60, 182.30};

const float min\_money\_card = 1000.00, min\_litre\_cash = 2.00;

//Global Variables of Add Charge Customer

char license [7];

int reps, id, item\_count, current\_customer;

const float min\_deposit = 10000, max\_litres = 3785.41, min\_cheque = 5000;

//Global Variables of Make Payment to Charge Customer

int type\_pay;

float payment;

float payment\_litre;

//Global Variables that Update Charge Customer

int field\_option, field\_type;

//Global Variables that Refuel Tank

const float FUEL\_TANK\_CAPACITY[5] = {0, 75708.23, 113562.35, 94635.29};

const float refuel\_per\_gallon [5] = {0, 165.64, 171.74, 159.38};

float CURRENT\_FUEL\_TANK\_CAPACITY [5] = {0, 75708.23, 113562.35, 94635.29};

float refuel\_amount;

//Global Variables of Generate Report

int report\_option;

float FUEL\_SALES[5] = {0, 0, 0, 0};

float TOTAL\_LITRES = 0;

float SALES\_GRAND\_TOTAL[4] = {0, 0, 0};

//ProtoType Functions

char luq\_req();

char generate\_preference();

const char \*get\_current\_date();

float accumulate\_fuel\_sales();

float accumulate\_fuel\_sales2();

float accumulate\_litres();

float accumulate\_litres2();

float convert\_gallon\_to\_litres(float x);

float gallon();

float gct();

float litre();

int login();

int search\_id(ChargeCustomer \*client, int key);

int search\_reps(ChargeCustomer \*client, int key);

void add\_charge\_customer(ChargeCustomer \*ptrclient);

void payment\_charge\_customer(ChargeCustomer \*ptrclient);

void deposit\_serve(ChargeCustomer \*ptrclient);

void create\_default\_customers(ChargeCustomer \*ptrclient);

void prompt\_int\_range(const char \*message, int \*option, int min, int max, const char \*error);

void generate\_license(ChargeCustomer \*ptrcust, int i);

void generate\_preference\_output(ChargeCustomer \*ptrclient);

void generate\_report(ChargeCustomer \*ptrclient);

void receipt1(Customer \*ptrcust);

void receipt2(ChargeCustomer \*ptrclient);

void refuel\_tank();

void refuel\_tank\_output();

void remove\_charge\_customer(ChargeCustomer \*ptrclient);

void saveToFileCash(FILE \*fcashcust, Customer \*ptr);

void saveToFileCharge(FILE \*fchargecust, ChargeCustomer \*ptclient);

void saveToFileCharge2(FILE \*fchargecust2, ChargeCustomer \*ptclient);

void serve\_customer(Customer \*ptrcust, ChargeCustomer \*ptrclient);

void update\_charge\_customer(ChargeCustomer \*ptrclient);

int main()

{

FILE \*fcashcust;

FILE \*fchargecust;

FILE \*fchargecust2;

fcashcust = fopen("CashCustomerData.txt","a");

fchargecust = fopen("ChargeCustomerData.txt", "w");

fchargecust2 = fopen("List\_of\_representatives.txt", "w");

srand(time(NULL));

Customer customer[SIZE];

Customer \*ptrcust = customer;

ChargeCustomer client[SIZE];

ChargeCustomer \*ptrclient = client;

create\_default\_customers(ptrclient);

while (1) {

printf("1 - Serve Customer\n"

"2 - Add Charge Customer\n"

"3 - Update Charge Customer\n"

"4 - Delete Charge Customer\n"

"5 - Make Payment to Charge Account\n"

"6 - Refuel Tank\n"

"7 - Generate Report\n"

"8 - Exit\n"

"Enter a option (1 - 8): ");

scanf("%d", &option);

switch (option)

{

case 1:

if(overall\_list < SIZE)

{

serve\_customer(ptrcust,ptrclient);

}

else

{

puts("The system has reached max capacity for the day");

}

break;

case 2:

if (overall\_list < SIZE)

{

add\_charge\_customer(ptrclient);

}

else

{

puts("The system has reached max capacity for the day");

}

break;

case 3:

update\_charge\_customer(ptrclient);

break;

case 4:

remove\_charge\_customer(ptrclient);

break;

case 5:

payment\_charge\_customer(ptrclient);

break;

case 6:

refuel\_tank();

break;

case 7:

generate\_report(ptrclient);

break;

case 8:

saveToFileCash(fcashcust,ptrcust);

saveToFileCharge(fchargecust,ptrclient);

saveToFileCharge2(fchargecust2,ptrclient);

return 0;

break;

default:

puts("Invalid Option");

}

puts("");

}

return 0;

}

char lub\_req()

{

const char lub[2] = {'Y','N'};

return lub [rand() % 2];

}

char generate\_preference()

{

const char pre[2] = {'D','M'};

return pre[rand() % 2];

}

//Generates a response based on generate\_preference()

void generate\_preference\_output(ChargeCustomer \*ptrclient)

{

int response = generate\_preference();

printf("Generating Preference: %c\n",response);

if (response == 'D')

{

puts("Preference Deposit has been assigned to customer");

ptrclient[current\_customer].preference\_number = DEPOSIT;

strcpy(ptrclient[current\_customer].preference, preference\_method[DEPOSIT]);

}

else if (response == 'M')

{

puts("Preference Maximum litres has been assigned to customer");

ptrclient[current\_customer].preference\_number = MAXIMUM;

strcpy(ptrclient[current\_customer].preference,preference\_method[DEPOSIT]);

}

}

//Returns the current date as a string

const char \*get\_current\_date() {

static char date[11];

const time\_t clock = time(NULL);

struct tm \*time = localtime(&clock);

strftime(date, 11, "%d/%m/%Y", time);

return date;

}

//Accumulates each fuel type sales

float accumulate\_fuel\_sales2()

{

switch(fuel)

{

case E10\_87:

return FUEL\_SALES[E10\_87] += payment\_litre \* fuel\_per\_litre[fuel];

case E10\_90:

return FUEL\_SALES[E10\_90] += payment\_litre \* fuel\_per\_litre[fuel];

case DIESEL:

return FUEL\_SALES[DIESEL] += payment\_litre \* fuel\_per\_litre[fuel];

}

}

//Accumulates each fuel type sales on maximum litres charge customers

float accumulate\_fuel\_sales()

{

switch(fuel)

{

case E10\_87:

return FUEL\_SALES[E10\_87] += money;

case E10\_90:

return FUEL\_SALES[E10\_90] += money;

case DIESEL:

return FUEL\_SALES[DIESEL] += money;

}

}

float accumulate\_litres()

{

return TOTAL\_LITRES += litre();

}

float accumulate\_litres2()

{

return TOTAL\_LITRES += payment\_litre;

}

float convert\_gallon\_to\_litres(float x)

{

return x \* 3.785;

}

//Converts the refuel\_amount into gallons based on the fuel types

float gallon()

{

switch (fuel)

{

case E10\_87: return refuel\_amount / 165.64;

case E10\_90: return refuel\_amount / 171.74;

case DIESEL: return refuel\_amount / 159.38;

}

}

float gct()

{

return lub\_price[lub] \* 0.16;

}

float litre()

{

switch (fuel)

{

case E10\_87: return money / 184.9;

case E10\_90: return money / 193.6;

case DIESEL: return money / 182.3;

}

}

//Validates user to grant access to edit charge customer or view sales

int login() {

const char \*PASSWORD = "bruh";

char input[51];

scanf("%s", input);

return !strcmp(input, PASSWORD);

}

int search\_id (ChargeCustomer \*client, int key)

{

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

{

if (client[i].id == key)

{

return i;

}

}

return -1;

}

int search\_reps(ChargeCustomer \*client, int key)

{

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

{

if (client[i].representatives == key)

{

return i;

}

}

return -1;

}

void add\_charge\_customer(ChargeCustomer \*ptrclient)

{

printf("Enter admin password: ");

if (!login()) {

printf("Incorrect password\n");

}

else

{

puts("Login successful");

printf("New customer has been automatically assigned to id %d\n",current\_charge\_id);

ptrclient[chargelist].id = current\_charge\_id;

printf("Enter business name: ");

scanf("%s", ptrclient[chargelist].name);

prompt\_int\_range("Enter the number of representatives", &reps, 1, 5, "Number must be from 1 to 5");

ptrclient[chargelist].representatives = reps;

for (int i = 0; i < reps; i += 1)

{

generate\_license(ptrclient,i);

}

generate\_preference\_output(ptrclient);

if (ptrclient[current\_customer].preference\_number == DEPOSIT)

{

printf("Enter the initial money: $");

scanf("%f",&ptrclient[current\_customer].initial\_money);

while (ptrclient[current\_customer].initial\_money < min\_deposit)

{

printf("The minimum deposit is $%.2f\n",min\_deposit);

printf("Enter the initial money: $");

scanf("%f",&ptrclient[current\_customer].initial\_money);

}

}

else if(ptrclient[current\_customer].preference\_number == MAXIMUM)

{

printf("Enter the initial litres: ");

scanf("%f",&ptrclient[current\_customer].initial\_litres);

while (ptrclient[current\_customer].initial\_litres > max\_litres)

{

printf("The maximum litres is %.2f\n", max\_litres);

printf("Enter the initial litres: ");

scanf("%f",&ptrclient[current\_customer].initial\_litres);

}

}

puts("New customer has been successfully added to the system");

chargelist++;

current\_charge\_id++;

}

}

void create\_default\_customers(ChargeCustomer \*ptrclient) {

ChargeCustomer DEFAULT[3] = {

{ 1, "TANKWELDLTD", {"5786KW","0578PQ","7777TY"},3,"N/A","DEPOSIT","N/A", 0, 0, DEPOSIT, 0, 100000, 100000, 0, 0, 0, 0, 0, 0, 0},

{ 2, "LEADINGEDGELTD", {"6588HH", "5436RK"}, 2, "N/A", "MAXIMUM", "N/A" , 0, 0, MAXIMUM, 0, 0, 0, 0, 50, 50, 0, 0, 0, 0},

{ 3, "DERRIMONTRADINGLTD", {"3378JQ"}, 1, "N/A", "DEPOSIT", "N/A", 0, 0, DEPOSIT, 0, 50000, 50000, 0, 0, 0, 0, 0, 0, 0}

};

ptrclient[0] = DEFAULT[0];

ptrclient[1] = DEFAULT[1];

ptrclient[2] = DEFAULT[2];

}

void deposit\_serve(ChargeCustomer \*ptrclient)

{ puts("Serving Preference Deposit");

prompt\_int\_range("Enter fuel type (E10—87 = 1, E10-90 = 2, Diesel = 3)", &fuel, 1, 3, NULL);

strcpy(ptrclient[current\_customer].fuel\_type,fuel\_types[fuel]);

printf("Current Deposit Amount: $%.2f\n",ptrclient[current\_customer].current\_money);

printf("Enter the amount of fuel needed or Press 0 to Exit: $");

scanf("%f", &money);

while(ptrclient[current\_customer].current\_money - money < 0 || CURRENT\_FUEL\_TANK\_CAPACITY[fuel] - litre() < 0)

{

printf("Money entered has either exceeded Current deposit amount or Current Tank Capactity\n");

printf("Enter a lower amount or Press 0 to Exit");

scanf("%f",&money);

}

accumulate\_fuel\_sales();

accumulate\_litres();

CURRENT\_FUEL\_TANK\_CAPACITY[fuel] -= litre();

ptrclient[current\_customer].current\_money -= money;

item\_count = 1;

ptrclient[current\_customer].fuel\_amount = money;

ptrclient[current\_customer].money = ptrclient[current\_customer].fuel\_amount;

int response = lub\_req();

printf("Lubricant Request? %c\n", response);

//initialises lub to 0 once the loop starts again to not cause errors

ptrclient[current\_customer].lubricant\_amount = lub\_price[0];

if (response == 'Y') {

prompt\_int\_range("Enter the lubricant type (5W-30 = 1, 5W-40 – 2, 15W-40 – 3, SAE – 40 - 4)", &lub, 1, 4, NULL);

strcpy(ptrclient[current\_customer].lubricant,lubricant\_types[lub]);

while (1)

{

printf("Enter the lubricant amount or Press 0 to Exit: $");

scanf("%f",&ptrclient[current\_customer].lubricant\_amount);

while(ptrclient[current\_customer].current\_money - ptrclient[current\_customer].lubricant\_amount < 0)

{

printf("Money has exceeded Current deposit amount\n");

printf("Enter a lower amount or Press 0 to Exit");

scanf("%f",&ptrclient[current\_customer].lubricant\_amount);

}

if(ptrclient[current\_customer].lubricant\_amount >= lub\_price[lub] || ptrclient[current\_customer].lubricant\_amount == 0)

break;

else

printf("Invalid. Please enter a higher amount than $%.2f\n",ptrclient[current\_customer].lubricant\_amount);

}

ptrclient[current\_customer].current\_money -= ptrclient[current\_customer].lubricant\_amount;

ptrclient[current\_customer].money += ptrclient[current\_customer].lubricant\_amount;

//lubricant amount gets reassigned to its rightful value

ptrclient[current\_customer].lubricant\_amount = lub\_price[lub];

ptrclient[current\_customer].gct\_amount = gct();

item\_count += 1;

}

else

{

strcpy(ptrclient[current\_customer].lubricant,lubricant\_types[lub]);

}

ptrclient[current\_customer].sub\_total = ptrclient[current\_customer].fuel\_amount + ptrclient[current\_customer].lubricant\_amount;

ptrclient[current\_customer].grand\_total = ptrclient[current\_customer].sub\_total + ptrclient[current\_customer].gct\_amount;

SALES\_GRAND\_TOTAL[CHARGE] += ptrclient[current\_customer].grand\_total;

ptrclient[current\_customer].change = ptrclient[current\_customer].money - ptrclient[current\_customer].grand\_total;

receipt2(ptrclient);

chargecount++;

}

void generate\_license(ChargeCustomer \*ptrclient, int i)

{

char string[20];

for (int i = 0; i < 4; i++)

{

string[i] = 48 + (rand() % 57);

while (string[i] <48 || string[i] > 57)

{

string[i] = 48 + (rand() % 57);

}

}

for (int i = 4; i <6; i++)

{

string[i] = 65 + (rand() % 90);

while (string[i] <65 || string[i] > 90)

{

string[i] = 65 + (rand() % 90);

}

}

strcpy(ptrclient[chargelist].licenses[i],string);

printf("%s has been assigned to license number %d\n",string,i + 1);

}

void generate\_report(ChargeCustomer \*ptrclient)

{

printf("Enter admin password: ");

if (!login()) {

printf("Incorrect password\n");

}

else

{

puts("Login successful");

prompt\_int\_range("Which report would you like to view? (1 = Sales report, 2 = Charge Customers report, 3 = Fuel report)", &report\_option, 1, 3, NULL);

puts("");

switch (report\_option)

{

case SALES\_REPORT:

puts("Stateline Gas Station Sales Report");

printf("Total Cash Customers: %d\n",cashcount);

printf("Total Charge Customers: %d\n", chargecount);

printf("Total Cash Amount: $%.2f\n", SALES\_GRAND\_TOTAL[CASH]);

printf("Total Charge Amount: $%.2f\n",SALES\_GRAND\_TOTAL[CHARGE]);

printf("Grand Total: $%.2f\n", SALES\_GRAND\_TOTAL[CASH] + SALES\_GRAND\_TOTAL[CHARGE]);

break;

case CHARGE\_REPORT:

puts("Stateline Gas Station Charge Report");

for (int i = 0; i < 3; i++)

{

printf("Customer id: %d\n", ptrclient[i].id);

printf("Customer name: %s\n", ptrclient[i].name);

printf("Initial Deposit Amount: $%.2f\n",ptrclient[i].initial\_money);

printf("Balance Remaining: $%.2f\n", ptrclient[i].initial\_money - ptrclient[i].money);

printf("Initial Litres Amount: %.2f\n", ptrclient[i].initial\_litres);

printf("Litres Remaining: %.2f\n", ptrclient[i].initial\_litres - ptrclient[i].litres);

printf("Amount Owned: \n");

printf("Amount Paid: \n");

printf("Balance Remaining: \n");

puts("");

}

break;

case FUEL\_REPORT:

puts("Stateline Gas Station Fuel Report");

printf("Total Fuel Sales: $%.2f\n",FUEL\_SALES[E10\_87] + FUEL\_SALES[E10\_90] + FUEL\_SALES[DIESEL]);

printf("E10-87 Sales: $%.2f\n", FUEL\_SALES[E10\_87]);

printf("E10-90 Sales: $%.2f\n", FUEL\_SALES[E10\_90]);

printf("Diesel Sales: $%.2f\n", FUEL\_SALES[DIESEL]);

printf("Total Litres Purchased: %.2f litres\n", TOTAL\_LITRES);

break;

}

}

}

void maximum\_litre\_serve(ChargeCustomer \*ptrclient)

{

puts("Serving Preference Maximum Litres");

prompt\_int\_range("Enter fuel type (E10—87 = 1, E10-90 = 2, Diesel = 3)", &fuel, 1, 3, NULL);

strcpy(ptrclient[current\_customer].fuel\_type,fuel\_types[fuel]);

printf("Current Litres Amount: %.2f\n",ptrclient[current\_customer].current\_litres);

printf("Enter the amount of litres needed: ");

scanf("%f", &payment\_litre);

while(ptrclient[current\_customer].current\_litres - payment\_litre < 0 || payment\_litre < 1 || payment\_litre > CURRENT\_FUEL\_TANK\_CAPACITY[fuel])

{

printf("Litre has either exceeded or not meet above 0 litres\n");

printf("Enter amount of litres: ");

scanf("%f",&payment\_litre);

}

accumulate\_litres2();

accumulate\_fuel\_sales2();

CURRENT\_FUEL\_TANK\_CAPACITY[fuel] -= payment\_litre;

ptrclient[current\_customer].current\_litres -= payment\_litre;

item\_count = 1;

ptrclient[current\_customer].fuel\_amount = payment\_litre \* fuel\_per\_litre[fuel];

ptrclient[current\_customer].money = payment\_litre \* fuel\_per\_litre[fuel];

int response = lub\_req();

printf("Lubricant Request? %c\n", response);

//initialises lub to 0 once the loop starts again to not cause errors

ptrclient[current\_customer].lubricant\_amount = lub\_price[0];

if (response == 'Y') {

prompt\_int\_range("Enter the lubricant type (5W-30 = 1, 5W-40 – 2, 15W-40 – 3, SAE – 40 - 4)", &lub, 1, 4, NULL);

strcpy(ptrclient[current\_customer].lubricant,lubricant\_types[lub]);

while (1)

{

printf("Enter the lubricant amount: ");

scanf("%f",&payment\_litre);

while(ptrclient[current\_customer].current\_litres - payment\_litre < 0 || payment\_litre < 1)

{

printf("Litre entered either has exceeded current litres in account or less than Mimimum litre\n");

printf("Enter a lower amount or Press 0 to Exit: ");

scanf("%f",&payment\_litre);

}

if(payment\_litre \* lub\_price[lub] >= lub\_price[lub])

break;

else

puts("Invalid");

}

ptrclient[current\_customer].current\_litres -= payment\_litre;

ptrclient[current\_customer].money += payment\_litre \* lub\_price[lub];

//lubricant amount gets reassigned to its rightful value

ptrclient[current\_customer].lubricant\_amount = lub\_price[lub];

ptrclient[current\_customer].gct\_amount = gct();

item\_count += 1;

}

else

{

strcpy(ptrclient[current\_customer].lubricant,lubricant\_types[lub]);

}

ptrclient[current\_customer].sub\_total = ptrclient[current\_customer].fuel\_amount + ptrclient[current\_customer].lubricant\_amount;

ptrclient[current\_customer].grand\_total = ptrclient[current\_customer].sub\_total + ptrclient[current\_customer].gct\_amount;

SALES\_GRAND\_TOTAL[CHARGE] += ptrclient[current\_customer].grand\_total;

ptrclient[current\_customer].change = ptrclient[current\_customer].money - ptrclient[current\_customer].grand\_total;

receipt2(ptrclient);

chargecount++;

}

void payment\_charge\_customer(ChargeCustomer \*ptrclient)

{

printf("Enter admin password: ");

if (!login()) {

printf("Incorrect password\n");

}

else

{

puts("Login successful");

printf("Enter customer id to make payment: ");

scanf("%d",&id);

if (search\_id(ptrclient,id) != -1)

{

puts("Customer has been found");

current\_customer = search\_id(ptrclient,id);

if (ptrclient[current\_customer].preference\_number == DEPOSIT)

{

puts("Preference Type: Deposit");

prompt\_int\_range("Enter payment type (Cash/Card = 1 or Cheque = 2)",&type\_pay,1, 2, NULL);

printf("Enter payment amount or Press 0 to Exit: ");

scanf("%f",&payment);

while (payment < min\_money\_card && type\_pay == CARDCASH)

{

if (payment == 0)

{

break;

}

printf("The Mimimum payment cash/card amount is $%.2f\n",min\_money\_card);

printf("Enter payment amount or Press 0 to exit: $");

scanf("%f",&payment);

}

while(payment < min\_cheque && type\_pay == CHEQUE)

{

if (payment == 0)

{

break;

}

printf("The Mimimum payment cash/card amount is $%.2f\n",min\_cheque);

printf("Enter payment amount or Press 0 to exit: $");

scanf("%f",&payment);

}

ptrclient[current\_customer].current\_money += payment;

}

else if (ptrclient[current\_customer].preference\_number == MAXIMUM)

{

printf("Enter the litres needed: ");

scanf("%f",&payment\_litre);

while(payment\_litre > max\_litres)

{

printf("The max litres is %.2f\n",max\_litres);

printf("Enter the litres needed: ");

scanf("%f",&payment\_litre);

}

ptrclient[current\_customer].current\_litres += payment\_litre;

}

else

{

puts("Invalid id. Customer was not found");

}

}

}

}

void prompt\_int\_range(const char \*message, int \*option, int min, int max, const char \*error) {

while (1) {

printf("%s: ", message);

scanf("%d", option);

if (\*option >= min && \*option <= max)

break;

else if (error != NULL)

printf("%s\n", error);

else

printf("Invalid option\n");

}

}

void receipt1(Customer \*ptrcust)

{

puts("");

puts("Stateline Gas Station");

printf("Date of transaction: %s\n", get\_current\_date());

printf("Customer Type: %s\n", ptrcust[serve\_list].cust\_type);

printf("Item count: %d\n",item\_count);

printf("===============================\n");

printf("Fuel Type: %s\n",ptrcust[serve\_list].fuel\_type);

printf("Cost: $%.2f\n",ptrcust[serve\_list].fuel\_amount);

printf("Lubricant Type: %s\n",ptrcust[serve\_list].lubricant);

printf("Cost: $%.2f\n",ptrcust[serve\_list].lubricant\_amount);

printf("===============================\n");

printf("Sub-Total: $%.2f\n",ptrcust[serve\_list].sub\_total);

printf("GCT: $%.2f\n",ptrcust[serve\_list].gct\_amount);

printf("Grand Total: $%.2f\n\n",ptrcust[serve\_list].grand\_total);

printf("Cash: $%.2f\n",ptrcust[serve\_list].grand\_total);

printf("Amount Tendered: $%.2f\n",ptrcust[serve\_list].money);

printf("Change/Amount Owned: $%.2f\n",ptrcust[serve\_list].change);

}

void receipt2(ChargeCustomer \*ptrclient)

{

puts("");

puts("Stateline Gas Station");

printf("Date of transaction: %s\n", get\_current\_date());

printf("Customer Type: Charge\n");

printf("Item count: %d\n",item\_count);

printf("===============================\n");

printf("Fuel Type: %s\n",ptrclient[current\_customer].fuel\_type);

printf("Cost: $%.2f\n",ptrclient[current\_customer].fuel\_amount);

printf("Lubricant Type: %s\n",ptrclient[current\_customer].lubricant);

printf("Cost: $%.2f\n",ptrclient[current\_customer].lubricant\_amount);

printf("===============================\n");

printf("Sub-Total: $%.2f\n",ptrclient[current\_customer].sub\_total);

printf("GCT: $%.2f\n",ptrclient[current\_customer].gct\_amount);

printf("Grand Total: $%.2f\n\n",ptrclient[current\_customer].grand\_total);

printf("Cash: $%.2f\n",ptrclient[current\_customer].grand\_total);

printf("Amount Tendered: $%.2f\n",ptrclient[current\_customer].money);

printf("Change/Amount Owned: $%.2f\n",ptrclient[current\_customer].change);

}

void refuel\_tank()

{

printf("Enter admin password: ");

if (!login()) {

printf("Incorrect password\n");

}

else

{

puts("Login successful");

prompt\_int\_range("Enter fuel type (E10—87 = 1, E10-90 = 2, Diesel = 3)", &fuel, 1, 3, NULL);

switch (fuel)

{

case E10\_87:

refuel\_tank\_output();

break;

case E10\_90:

refuel\_tank\_output();

break;

case DIESEL:

refuel\_tank\_output();

break;

}

}

}

void refuel\_tank\_output()

{

printf("Current Tank Capacity: %.2f litres\n",CURRENT\_FUEL\_TANK\_CAPACITY[fuel]);

printf("Enter the refuel amount in gallons or Press 0 to exit: ");

scanf("%f",&refuel\_amount);

while (CURRENT\_FUEL\_TANK\_CAPACITY[fuel] + convert\_gallon\_to\_litres((gallon())) > FUEL\_TANK\_CAPACITY[fuel] || refuel\_amount < refuel\_per\_gallon[fuel])

{

if(refuel\_amount == 0)

{

break;

}

printf("Higher than $%.2f is required.\n",refuel\_per\_gallon[fuel] \*2);

printf("Enter refuel amount in gallons or Press 0 to exit: ");

scanf("%f",&refuel\_amount);

}

CURRENT\_FUEL\_TANK\_CAPACITY[fuel] += convert\_gallon\_to\_litres(gallon());

puts("Tank has been successfully refueled");

printf("Current Tank Capacity: %.2f litres\n", CURRENT\_FUEL\_TANK\_CAPACITY[fuel]);

}

void remove\_charge\_customer(ChargeCustomer \*ptrclient)

{

printf("Enter admin password: ");

if (!login()) {

printf("Incorrect password\n");

}

else

{

puts("Login successful");

printf("Enter customer id number to delete: ");

scanf("%d",&id);

if (search\_id(ptrclient,id) != -1)

{

puts("Customer has been found");

current\_customer = search\_id(ptrclient,id);

ptrclient[current\_customer] = (ChargeCustomer) {

.id = 0,

.licenses = {0},

.representatives = 0,

.fuel\_type = {0},

.preference = {0},

.lubricant = {0},

.lubricant\_amount = 0,

.fuel\_amount = 0,

.preference\_number = 0,

.money = 0,

.current\_money = 0,

.initial\_money = 0,

.litres = 0,

.current\_litres = 0,

.initial\_litres = 0,

.gct\_amount = 0,

.sub\_total = 0,

.grand\_total = 0,

.change = 0

};

puts("Customer fields has been deleted from system");

}

else

{

puts("Invalid id. Customer was not found");

}

}

}

void serve\_customer(Customer \*ptrcust, ChargeCustomer \*ptrclient)

{

char input[30];

item\_count = 0;

prompt\_int\_range("Enter a customer type (1 - Cash, 2 - Charge)", &cust\_type, 1, 2, NULL);

if (cust\_type == CASH)

{

strcpy(ptrcust[serve\_list].cust\_type,"Cash");

ptrcust[serve\_list].gct\_amount = 0;

prompt\_int\_range("Enter the payment type (1 - Cash or 2 - Card)", &cod\_type, 1, 2, NULL);

prompt\_int\_range("Enter fuel type (E10—87 = 1, E10-90 = 2, Diesel = 3)", &fuel, 1, 3, NULL);

strcpy(ptrcust[serve\_list].fuel\_type,fuel\_types[fuel]);

printf("Enter the amount of fuel needed: $");

scanf("%f", &money);

while (cod\_type == 1 && litre() <min\_litre\_cash || litre() > CURRENT\_FUEL\_TANK\_CAPACITY[fuel]) {

puts("");

if(litre() > CURRENT\_FUEL\_TANK\_CAPACITY[fuel])

{

printf("Amount entered exteeds %s current tank capacity\n",fuel\_types[fuel]);

}

printf("Mimimum of higher than $%.2f is required.\n",fuel\_per\_litre[fuel] \*2);

printf("A minimum of 2 litres is required.\nPlease enter money or Press 0 to Exit: $");

scanf("%f",&money);

if(money == 0)

{

break;

}

}

while (cod\_type == 2 && (money < min\_money\_card || litre() > CURRENT\_FUEL\_TANK\_CAPACITY[fuel]))

{

puts("");

if(money == 0)

{

break;

}

if(litre() > CURRENT\_FUEL\_TANK\_CAPACITY[fuel])

{

printf("Amount entered exteeds %s current tank capacity\n",fuel\_types[fuel]);

}

printf("The minimum amount of money is $1000. Please enter money or Press 0 to Exit: $");

scanf("%f",&money);

}

accumulate\_fuel\_sales();

accumulate\_litres();

CURRENT\_FUEL\_TANK\_CAPACITY[fuel] -= litre();

item\_count = 1;

ptrcust[serve\_list].fuel\_amount = money;

ptrcust[serve\_list].money += ptrcust[serve\_list].fuel\_amount;

int response = lub\_req();

//Randomly generates yes or no

printf("Lubricant Request? %c\n", response);

//initialises lub to 0 once the loop starts again to not cause errors

ptrcust[serve\_list].lubricant\_amount = lub\_price[0];

if (response == 'Y') {

prompt\_int\_range("Enter the lubricant type (5W-30 = 1, 5W-40 – 2, 15W-40 – 3, SAE – 40 - 4)", &lub, 1, 4, NULL);

strcpy(ptrcust[serve\_list].lubricant,lubricant\_types[lub]);

while (1)

{

printf("Enter the lubricant amount: $");

scanf("%f",&ptrcust[serve\_list].lubricant\_amount);

if(ptrcust[serve\_list].lubricant\_amount >= lub\_price[lub])

break;

else

printf("Invalid. Please enter a higher amount than $%.2f\n",ptrcust[serve\_list].lubricant\_amount);

}

ptrcust[serve\_list].money += ptrcust[serve\_list].lubricant\_amount;

//lubricant amount gets reassigned to its rightful value

ptrcust[serve\_list].lubricant\_amount = lub\_price[lub];

ptrcust[serve\_list].gct\_amount = gct();

item\_count += 1;

}

else

{

strcpy(ptrcust[serve\_list].lubricant,lubricant\_types[lub]);

}

ptrcust[serve\_list].sub\_total = ptrcust[serve\_list].fuel\_amount + ptrcust[serve\_list].lubricant\_amount;

ptrcust[serve\_list].grand\_total = ptrcust[serve\_list].sub\_total + ptrcust[serve\_list].gct\_amount;

SALES\_GRAND\_TOTAL[CASH] += ptrcust[serve\_list].grand\_total;

ptrcust[serve\_list].change = ptrcust[serve\_list].money - ptrcust[serve\_list].grand\_total;

receipt1(ptrcust);

serve\_list++;

cashcount++;

overall\_list++;

}

else

{

if (cust\_type == CHARGE)

{

printf("Please enter customer id: ");

scanf("%d",&id);

if (search\_id(ptrclient,id) != -1)

{

current\_customer = search\_id(ptrclient,id);

puts("Customer has been found");

printf("Enter number of representatives: ");

scanf("%d",&reps);

if (search\_reps(ptrclient,reps) !=-1 )

{

puts("Number of representatives has been confirmed");

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

{

printf("Enter license number %d: ",j+1);

scanf("%s",license);

while (strcmp(license,ptrclient[current\_customer].licenses[j])!=0)

{

printf("Invalid license number %d\n",j+1);

printf("Enter correct license number: ");

scanf("%s",license);

}

}

if(ptrclient[current\_customer].preference\_number == DEPOSIT)

{

deposit\_serve(ptrclient);

}

else if(ptrclient[current\_customer].preference\_number == MAXIMUM)

{

maximum\_litre\_serve(ptrclient);

}

overall\_list++;

}

else

{

puts("Number of representatives does not match the system records\n"

"Re-enter and try again");

}

}

else

{

puts("Customer not found");

}

}

}

}

void update\_charge\_customer(ChargeCustomer \*ptrclient)

{

printf("Enter admin password: ");

if (!login()) {

printf("Incorrect password\n");

}

else

{

puts("Login successful");

printf("Enter customer id number to update: ");

scanf("%d",&id);

if (search\_id(ptrclient,id) != -1)

{

current\_customer = search\_id(ptrclient,id);

puts("Customer has been found");

prompt\_int\_range("Which field would you like to edit?\n1 - All fields\n2 - Single field\nEnter option", &field\_option, 1,2,NULL);

switch(field\_option)

{

case 1:

printf("Input new business name of customer: ");

scanf("%s",ptrclient[current\_customer].name);

prompt\_int\_range("Enter number of representatives: ", &reps, 1, 5, NULL);

ptrclient[current\_customer].representatives = reps;

generate\_preference\_output(ptrclient);

puts("Customer field has been successfully edited");

break;

case 2:

prompt\_int\_range("1 - Customer name\n2 - Number of representatives\n3-Preference\nEnter a single field to edit", &field\_type, 1 ,3, NULL );

if (field\_type == 1)

{

printf("Enter new business name: ");

scanf("%s",ptrclient[current\_customer].name);

}

else

{

if(field\_type == 2)

{

prompt\_int\_range("Enter new number of representatives (1 - 5): ",&ptrclient[current\_customer].representatives, 1, 5, NULL);

}

else

{

generate\_preference\_output(ptrclient);

}

}

puts("Customer field has been successfully edited");

break;

}

}

else

{

puts("Invalid ID. Customer not found");

}

}

}

void saveToFileCash(FILE \*fcashcust, Customer \*ptrcust) {

for (int i = 0; i < cashcount; i++) {

fprintf(fcashcust, "%5s %9s %9s %12.2f %5.2f %12.2f %12.2f %12.2f\n",

ptrcust->cust\_type, ptrcust->fuel\_type, ptrcust->lubricant, ptrcust->sub\_total, ptrcust->gct\_amount,

ptrcust->grand\_total, ptrcust->money, ptrcust->change);

ptrcust++;

}

fclose(fcashcust);

}

void saveToFileCharge(FILE \*fchargecust, ChargeCustomer \*ptrclient) {

fprintf(fchargecust, "%4s %20s %4s %10s %12s %10s\n","ID","Customer Name","Reps","Preference","Amount $", "Litre");

for (int i = 0; i < chargelist; i++) {

fprintf(fchargecust, "%4d %20s %4d %10s %12.2f %10.2f\n",

ptrclient->id, ptrclient->name, ptrclient->representatives, ptrclient->preference,

ptrclient->current\_money, ptrclient->current\_litres);

ptrclient++;

}

fclose(fchargecust);

}

void saveToFileCharge2(FILE \*fchargecust2, ChargeCustomer \*ptrclient) {

fprintf(fchargecust2,"%4s %6s %6s %6s %6s %6s\n", "ID", "Rep#1", "Rep#2", "Rep#3", "Rep#4", "Rep#5");

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

{

fprintf(fchargecust2,"%4d %6s %6s %6s %6s %6s\n", ptrclient[i].id,

ptrclient[i].licenses[0], ptrclient[i].licenses[1],ptrclient[i].licenses[2], ptrclient[i].licenses[3], ptrclient[i].licenses[4]);

}

fclose(fchargecust2);

}

# Declaration of Authorship

## Shanoi Walker Gayle



## Gabrielle Tucker



## Chevannese Ellis



## Jonathan Blackwood

