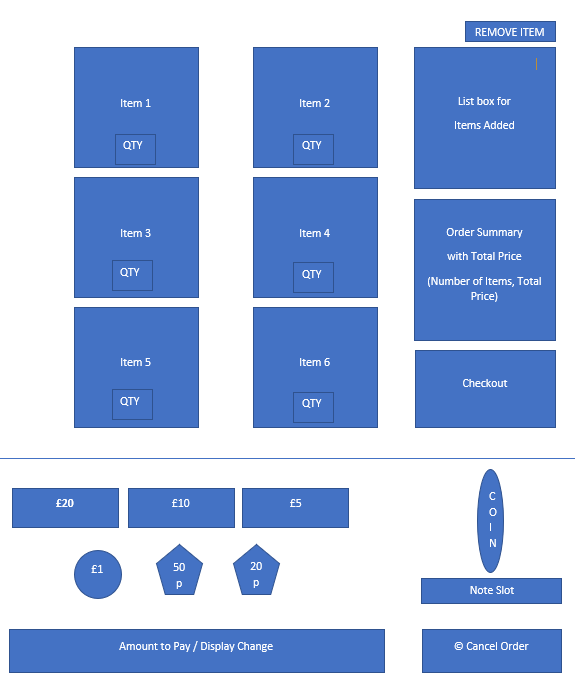
## Plan 1

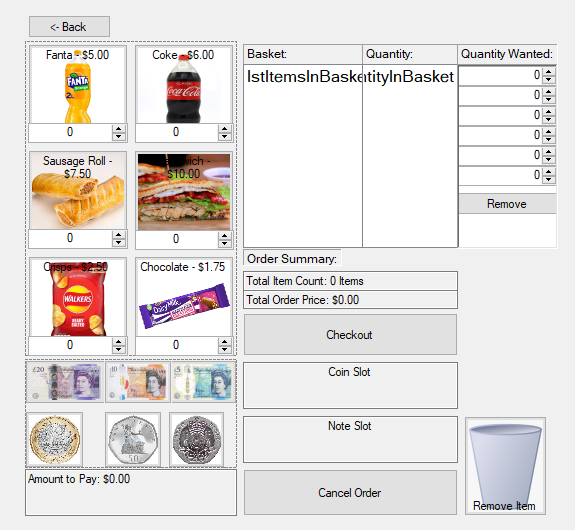
This plan shows the initial design concept, I created two plans for this project allowing me to complete the basic objectives and afterwards explore adding some individual functionality. Working this way ensures I have a complete project to hand in but allows me to expand my project to be a better application and push my skills.

From my initial design I have kept it extremely simple, keeping much of the design similar to the existing Cover sheet design. I have made a couple of alterations though as I feel there were a couple of bare bones features missing, for starters a remove item button to allow customers to change their minds about products and a back button which allows customers to return to the order screen after pressing check out for final item additions. Upon clicking the Remove Button, a prompt will ask the user to click on the item they wish to remove from their basket, it will then confirm their choice allowing them to return to select a new item if they selected an incorrect item, once confirmed the item will be removed, this was updated in version two of my code to be drag and drop. I added a note slot to my payment section as I felt it was strange to be inputting notes into a coin slot, small quality of life update which makes the program feel more user-friendly, this has been set up so once a coin or note is selected the opposite slot will be disabled. This lays out the initial plan for this program, once I have this completed I will look at version 2 and how we can improve this further.



## Plan 2 – Improved

The second variation of my plan includes a variation on how I planned to remove items from the order, I wanted the customer to have more control over how many items should be removed in case they didn’t want to remove all. This system allows the customer to select which item to remove using drag and drop to the trash can which makes the side menu visible, only the matching index numeric up down will show to make it easier to understand after the customer has adjusted the value they press the remove button which will update the form with new values. I also added a second text box to show the amount of change due, this way enabled me to swap the visibility when the order was paid for.



## Test Table

A lot of testing was undertaken during the development of my program, which has meant my end testing has been quite simple as I knew all features worked.

Quantity 0 on item selection

Graphical user interface, application

Description automatically generated

Pressing cancel before any items added to order

Graphical user interface, application

Description automatically generated

Pressing check out before any items added to order

Graphical user interface, application

Description automatically generated

Dragging item to trash before adding to order = No drag drop operation allowed

Removing items in each index – Works perfectly

Removing singular quantities – Works perfectly

Dragging money without pressing checkout = No drag drop operation allowed

Dragging a coin to note slot = No drag drop operation allowed

Dragging a note to coin slot = No drag drop operation allowed

Pressing back button and then commencing with order – Works perfectly

Removing all items – Works perfectly

Write to file – Works perfectly

## Source Code:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Globalization;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.IO;

namespace Check\_Out\_Ver.\_1.\_0

{

public partial class Vending\_Machine : Form

{

public Vending\_Machine()

{

InitializeComponent();

}

private readonly string[,] itemCatalogue = new string[6, 2] { { "Fanta", "5.00" }, { "Coke", "6.00" }, // Item catalogue in a 2 dimensional array to store item name and item price

{ "Sausage Roll", "7.50" }, { "Sandwich", "10.00" }, { "Crisps", "2.50" }, { "Chocolate", "1.75" } };

private string itemName, tempString = ""; // Variables to store item name and a temporary string to store substrings while we change values

private double itemPrice = 0.00, orderTotal = 0.00, tenderDue = 0.00, tenderPaid = 0.00, changeDue = 0.00; // All monetary variables to handle prices and order total

private int itemQuantity = 0, previousItemQuantity = 0, totalItemQuantity = 0, orderCount = 0, index = 0; // All quantity variables to handle number of items and an order count variable

private bool orderActive = true; // Bool variable to create while true loop for order creation

private readonly List<string> orderList = new List<string>(); // List to create an orderList which will store order number, item name, item quantity, item price and total order price

private void FindItem(string itemChoice) // Function to find the item button pressed

{

itemQuantity = 0; // Resets item quantity to stop items being added while having no quantity because item quantity still has previous value

if (itemChoice == "btnItem\_1") // If selector checks the name of the button pressed and sorts that to corressponding item

{

itemName = itemCatalogue[0, 0]; // Finds and stores the name of the item from item catalogue

itemPrice = Convert.ToDouble(itemCatalogue[0, 1]); // Finds, converts to double and stores the price of the item from item catalogue

if (nudItem1Quantity.Value != 0) // Check the value of Item Quantity Numeric Up Down and checks value is not 0

{

itemQuantity = (int)nudItem1Quantity.Value; // Send to item quantity and convert from decimal to int

nudItem1Quantity.Value = 0; // Reset numeric up down value to 0

}

}

else if (itemChoice == "btnItem\_2")

{

itemName = itemCatalogue[1, 0];

itemPrice = Convert.ToDouble(itemCatalogue[1, 1]);

if (nudItem2Quantity.Value != 0)

{

itemQuantity = (int)nudItem2Quantity.Value;

nudItem2Quantity.Value = 0;

}

}

else if (itemChoice == "btnItem\_3")

{

itemName = itemCatalogue[2, 0];

itemPrice = Convert.ToDouble(itemCatalogue[2, 1]);

if (nudItem3Quantity.Value != 0)

{

itemQuantity = (int)nudItem3Quantity.Value;

nudItem3Quantity.Value = 0;

}

}

else if (itemChoice == "btnItem\_4")

{

itemName = itemCatalogue[3, 0];

itemPrice = Convert.ToDouble(itemCatalogue[3, 1]);

if (nudItem4Quantity.Value != 0)

{

itemQuantity = (int)nudItem4Quantity.Value;

nudItem4Quantity.Value = 0;

}

}

else if (itemChoice == "btnItem\_5")

{

itemName = itemCatalogue[4, 0];

itemPrice = Convert.ToDouble(itemCatalogue[4, 1]);

if (nudItem5Quantity.Value != 0)

{

itemQuantity = (int)nudItem5Quantity.Value;

nudItem5Quantity.Value = 0;

}

}

else if (itemChoice == "btnItem\_6")

{

itemName = itemCatalogue[5, 0];

itemPrice = Convert.ToDouble(itemCatalogue[5, 1]);

if (nudItem6Quantity.Value != 0)

{

itemQuantity = (int)nudItem6Quantity.Value;

nudItem6Quantity.Value = 0;

}

}

else

{

MessageBox.Show("Option Selected is not valid."); // Will never run, is here as a crash preventer

}

}

private void CreateOrder(string itemChoice) // Function to create order passing item choice as string

{

do // Do while loop to keep adding items until order active becomes false

{

FindItem(itemChoice); // Calls find item function with item choice being passed as string to select which item was pressed

orderTotal += itemPrice \* itemQuantity; // Increases the order total by item price multiplied by item quantity

tenderDue = orderTotal; // Sets tender due to order total after the increase is more performance friendly only performing the calculation once

totalItemQuantity += itemQuantity; // Increases total item quantity by item quantity

DuplicateFinder(); // Calls duplicate finder function to ensure that any item added isn't already in the order

DisplayValues(totalItemQuantity, orderTotal, tenderDue); // Call Display Values function passing total item quantity, order total and tender due to Update the listboxes across the form with new values

break; // break out of the while loop

}

while (orderActive); // While order active is true

}

private void DuplicateFinder() // Function to check for duplicate items being added to array

{

if (orderList.Contains("Item Name: " + itemName)) // If statement checks if order list contains item name

{

int quantityPosition = orderList.IndexOf("Item Name: " + itemName) + 1; // Gets the position of item quantity by finding the index of item name and increasing by one

string quantityString = orderList[quantityPosition]; // Gets the quantity string by pulling index just found from orderlist

int quantityValue = Convert.ToInt32(quantityString.Substring(15)); // Creates a substring from the 15th value and converts to an integer to get the item quantity value

itemQuantity += quantityValue; // Increase item quantity by the existing quantity just pulled from order list

orderList[quantityPosition] = (Convert.ToString("Item Quantity: " + itemQuantity)); // Selects the quantity position index in orderlist and sets the new quantity value

quantityPosition = lstItemsInBasket.Items.IndexOf(itemName); // Finding the quantity position in quantity list box by finding the item name in items in basket listbox

lstQuantityInBasket.Items.RemoveAt(quantityPosition); // Remove the previous quantity value in quantity list box at the quantity position index

lstQuantityInBasket.Items.Insert(quantityPosition, itemQuantity); // Inserts the new quantity at the quantity position index

}

else if (itemQuantity > 0) // Else if statement checks if item quantity is greater than 0 this will be used for new items

{

orderList.Add(Convert.ToString("Item Name: " + itemName)); // Adds Item name to order list and converts value to string

orderList.Add(Convert.ToString("Item Quantity: " + itemQuantity)); // Adds Item quantity to order list and converts value to string

orderList.Add(Convert.ToString(("Item Price: " + itemPrice))); // Adds Item price to order list and converts value to string

lstItemsInBasket.Items.Add(itemName); // Adds item name to items in basket listbox

lstQuantityInBasket.Items.Add(itemQuantity); // Adds item quantity to quantity in basket listbox

}

else // Else handles exceptions when there is no quantity

{

MessageBox.Show("Item Quantity Insufficient, Please Increase Value."); // Sends a message box to the user telling them quantity is insufficient and to increase values

}

}

private void RemoveItem(string itemName, int itemQuantity) // Function for removing items from order list

{

lblQuantityToRemove.Visible = true;

pnlQuantityToRemove.Visible = true;

btnCompleteItemRemoval.Visible = true;

if (lstItemsInBasket.SelectedIndex == 0) // If statements detects which index has been selected

{

lstQuantityInBasket.SelectedIndex = 0; // Selects the corresponding index in quantity in basket listbox

string value = lstQuantityInBasket.SelectedItem.ToString(); // Creates a string called value and stores the current quantity of item in the index

nudQuantityToRemoveItem1.Value = Convert.ToDecimal(value); // Converts value from string to decimal and sets the numeric up down value to match

nudQuantityToRemoveItem1.Visible = true; // Makes the numeric up down visible to user

}

else if (lstItemsInBasket.SelectedIndex == 1)

{

lstQuantityInBasket.SelectedIndex = 1;

string value = lstQuantityInBasket.SelectedItem.ToString();

nudQuantityToRemoveItem2.Value = Convert.ToDecimal(value);

nudQuantityToRemoveItem2.Visible = true;

}

else if (lstItemsInBasket.SelectedIndex == 2)

{

lstQuantityInBasket.SelectedIndex = 2;

string value = lstQuantityInBasket.SelectedItem.ToString();

nudQuantityToRemoveItem3.Value = Convert.ToDecimal(value);

nudQuantityToRemoveItem3.Visible = true;

}

else if (lstItemsInBasket.SelectedIndex == 3)

{

lstQuantityInBasket.SelectedIndex = 3;

string value = lstQuantityInBasket.SelectedItem.ToString();

nudQuantityToRemoveItem4.Value = Convert.ToDecimal(value);

nudQuantityToRemoveItem4.Visible = true;

}

else if (lstItemsInBasket.SelectedIndex == 4)

{

lstQuantityInBasket.SelectedIndex = 4;

string value = lstQuantityInBasket.SelectedItem.ToString();

nudQuantityToRemoveItem5.Value = Convert.ToDecimal(value);

nudQuantityToRemoveItem5.Visible = true;

}

else if (lstItemsInBasket.SelectedIndex == 5)

{

lstQuantityInBasket.SelectedIndex = 5;

string value = lstQuantityInBasket.SelectedItem.ToString();

nudQuantityToRemoveItem6.Value = Convert.ToDecimal(value);

nudQuantityToRemoveItem6.Visible = true;

}

ResetTimer(); // Calls function reset timer as action has been taken

}

private void MoneySelection(string moneySelection) // Function to select money sends button text as the value selected

{

if (moneySelection == "20") // If selection checks if the selected value is equal to 20

{

tenderPaid += 20; // Increases tender paid by 20

}

else if (moneySelection == "10")

{

tenderPaid += 10;

}

else if (moneySelection == "5")

{

tenderPaid += 5;

}

else if (moneySelection == "1")

{

tenderPaid += 1;

}

else if (moneySelection == "50p")

{

tenderPaid += 0.50;

}

else if (moneySelection == "20p")

{

tenderPaid += 0.20;

}

else

{

MessageBox.Show("Sorry that Value doesn't exist."); // Else statement handles exceptions with an error message

}

}

private void OrderPayment(string moneySelection) // Function to handle order payment passing money selection from button text

{

do // Do while loop while order total is less than or equal to tender paid

{

MoneySelection(moneySelection); // Calls money selection function passing money selection from button text to the function

if (tenderPaid < orderTotal) // If statement checks if tender paid is less than order total

{

txtAmountToPay.Visible = true; // Amount to pay becomes visible if it wasn't already (exception handler as default is true)

txtChange.Visible = false; // Change due visibility set to false if it wasn't already (exception handler as default is false)

tenderDue = orderTotal - tenderPaid; // Tender due set to order total minus tender paid

tempString = txtAmountToPay.Text.Substring(0, 16) + FormatNumbers(tenderDue); // Creates substring storing text from amount to pay, tender due value passed through format numbers function and stored in tempstring

txtAmountToPay.Clear(); // Clear the text box ready to append

txtAmountToPay.AppendText(tempString); // Append temp string containing new total amount to pay to text box

}

else if (orderTotal <= tenderPaid) // Else if statement checks if order total is less than or equal to tenderpaid

{

txtAmountToPay.Visible = false; // Makes amount to pay invisible

txtChange.Visible = true; // Makes change due visible

changeDue = tenderPaid - orderTotal; // Calculates how much change is due to the customer by minusing tender paid from order total and storing in change due

tempString = txtChange.Text.Substring(0, 13) + FormatNumbers(changeDue); // Creates substring storing text from change text, change due value passed through format numbers function and stored in tempstring

txtChange.Clear(); // Clear the text box ready to append

txtChange.AppendText(tempString); // Append temp string containing new change amount due to text box

CompleteOrder(); // Calls Complete order function this is to add order number and order total to order list after full payment has been received

}

break; // Break out of the do while loop

}

while (orderTotal <= tenderPaid); // While order total is less than or equal to tender paid means this loop runs while there is still more money to pay

}

private string FormatNumbers(double inputNumber) // Function to format numbers to two decimal places passing the numbers in as a double

{

string formattedNumber = String.Format("{0:0.00}", inputNumber); // Creates a string to store formated number in and formats input number to two decimal places

return formattedNumber; // Return formatted number to the program

}

private void DisplayValues(int totalItemQuantity, double orderTotal, double tenderDue) // Function to display and update values across the form in listboxes

{

tempString = txtTotalItemCount.Text.Substring(0, 18) + totalItemQuantity + " Items"; // Creates substring storing text from total item count, total item count value and storing in tempstring

txtTotalItemCount.Clear(); // Clear the text box ready to append

txtTotalItemCount.AppendText(tempString); // Append temp string containing total item count to text box

tempString = txtTotalOrderPrice.Text.Substring(0, 20) + FormatNumbers(orderTotal); // Creates substring storing text from total order price, total order price value passed through format numbers function and storing in tempstring

txtTotalOrderPrice.Clear(); // Clear the text box ready to append

txtTotalOrderPrice.AppendText(tempString); // Append temp string containing total order price to text box

tempString = txtAmountToPay.Text.Substring(0, 16) + FormatNumbers(tenderDue); // Creates substring storing text from amount to pay, tender due value passed through format numbers function and storing in tempstring

txtAmountToPay.Clear(); // Clear the text box ready to append

txtAmountToPay.AppendText(tempString); // Append temp string containing total amount to pay to text box

}

private void CancelOrder() // Function to cancel orders when butto pressed

{

string message = "Do you want to cancel this order?"; // Cancellation message displayed in message box

string title = "Cancel Order"; // Message box title

MessageBoxButtons buttons = MessageBoxButtons.YesNo; // We will use a yes/no message box for cancelling orders

DialogResult result = MessageBox.Show(message, title, buttons); // Displays the message box storing result from key press in result

switch (result) // Switch statement to control yes or no used this for the break if no was pressed

{

case DialogResult.Yes: // If yes pressed clears all values and then breaks out of the switch statement

ClearValues();

break;

case DialogResult.No: // If no pressed breaks out of the switch statement

break;

}

}

private void ClearValues() // Function to clear all values in the form

{

totalItemQuantity = 0; itemQuantity = 0; orderTotal = 0; tenderDue = 0; tenderPaid = 0; changeDue = 0; // Resets all monetary and quantity variables

orderList.Clear(); lstItemsInBasket.Items.Clear(); lstQuantityInBasket.Items.Clear(); // Clears all list boxes and order list using clear method

tempString = txtAmountToPay.Text.Substring(0, 16) + "0.00"; // Creates substring storing text from amount to pay, amount to pay value and stored in tempstring

txtAmountToPay.Clear(); // Clear the text box ready to append

txtAmountToPay.AppendText(tempString); // Append temp string containing new amount to pay to text box

tempString = txtChange.Text.Substring(0, 13) + "0.00"; // Creates substring storing text change due, change due value and stored in tempstring

txtChange.Clear(); // Clear the text box ready to append

txtChange.AppendText(tempString); // Append temp string containing new amount of change due to text box

tempString = txtTotalItemCount.Text.Substring(0, 18) + totalItemQuantity + " Items"; // Creates substring storing text from total item count, total item count value and stored in tempstring

txtTotalItemCount.Clear(); // Clear the text box ready to append

txtTotalItemCount.AppendText(tempString); // Append temp string containing new total item count to text box

tempString = txtTotalOrderPrice.Text.Substring(0, 20) + "0.00"; // Creates substring storing text from total order price, total order price value and stored in tempstring

txtTotalOrderPrice.Clear(); // Clear the text box ready to append

txtTotalOrderPrice.AppendText(tempString); // Append temp string containing new order total price to text box

tempString = ""; // Clears temp string value

orderActive = true; // Sets order active to true allowing an order to continue after cancel pressed

pnlMoneySelection.Enabled = false; // Disables the Money panel in case they were in checkout screen

pnlItemSelection.Enabled = true; // Enables Item selection panel in case they were in checkout screen

btnBackToOrder.Visible = false; // Makes the back button invisible

txtAmountToPay.Visible = true; // Makes amount to pay visible

txtChange.Visible = false; // Makes change due invisible

}

private void CompleteOrder() // Function to complete order adding order number and order value at the beginning and end of order list then calling send to file function

{

if (changeDue > 0) // If statement checks if change due is greater than 0

{

MessageBox.Show("Order is Complete, Thank You and Don't Forget Your $" + FormatNumbers(changeDue) + " Change."); // Displays a thank you message along with how much change they received

}

else // Else statement if no change is due to the customer

{

MessageBox.Show("Order is Complete, Thank You and See You Soon"); // Displays a order confirmation message and says thank you see you soon

}

orderCount++; // Increments order count by one

orderList.Insert(0, "Order Number: " + orderCount); // Inserts order number at the beginning of order list

orderList.Add("Order Total: $" + FormatNumbers(orderTotal)); // Adds order total to the end of order list

orderList.Add("Change Given: $" + FormatNumbers(changeDue) + Environment.NewLine); // Adds change due and a line of separation between orders in the output file for easier recognition of new orders

FileOutput(); // Call file output function to handle sending the order list to the Order History file

ClearValues(); // Call clear values function to clear all values ready for the next order

}

private void FileOutput() // Function to handle all file outputs

{

string path = @"c:\Users\conno\Desktop\OrderHistory.txt"; // Path to the file being used, hard coded because I wanted this on my desktop not in amongst project files

for (int i = 0; i < orderList.Count; i++) // For loop to iterate through order list while i is less than count of order list

{

File.AppendAllText(path, orderList[i].ToString() + Environment.NewLine); // Append all text method used as I wanted the file to be created if it didn't exist already

} // Uses i as index value to iterate through order list appending to file and adding new lines for each item added

}

private void ItemClick(object sender, EventArgs e) // Function to handle items being clicked

{

Button button = sender as Button; // Creates object of same button class and typecasts the button object to local object to access it properties

CreateOrder(button.Name); // Calls create order function passing local button object name as item choice parameter

ResetTimer(); // Resets timer as action has been taken

}

private void BtnMoney20\_MouseDown(object sender, MouseEventArgs e) // All mouse down events for money being dragged

{

tempString = "20"; // Sets temp string value to 20 this acts as a check for coin or note

btnMoney20.DoDragDrop(btnMoney20.Text, DragDropEffects.Move); // Do drag drop event sending button text value as sender and using Move drag drop effect

}

private void BtnMoney10\_MouseDown(object sender, MouseEventArgs e)

{

tempString = "10";

btnMoney10.DoDragDrop(btnMoney10.Text, DragDropEffects.Move);

}

private void BtnMoney5\_MouseDown(object sender, MouseEventArgs e)

{

tempString = "5";

btnMoney5.DoDragDrop(btnMoney5.Text, DragDropEffects.Move);

}

private void BtnMoney1\_MouseDown(object sender, MouseEventArgs e)

{

tempString = "1";

btnMoney1.DoDragDrop(btnMoney1.Text, DragDropEffects.Move);

}

private void BtnMoney50p\_MouseDown(object sender, MouseEventArgs e)

{

tempString = "50p";

btnMoney50p.DoDragDrop(btnMoney50p.Text, DragDropEffects.Move);

}

private void BtnMoney20p\_MouseDown(object sender, MouseEventArgs e)

{

tempString = "20p";

btnMoney20p.DoDragDrop(btnMoney20p.Text, DragDropEffects.Move);

}

private void CoinDragEnter(object sender, DragEventArgs e) // Coin slot check

{

if (tempString == "20" || tempString == "10" || tempString == "5") // If statement checks if temp string holds the value of a note using or

{

e.Effect = DragDropEffects.None; // Changes drag drop effect to none to show a note can't be inserted into the coin slot

}

else if (tempString == "1" || tempString == "50p" || tempString == "20p") // If statement checks if temp string holds the value of a coin using or

{

e.Effect = DragDropEffects.Move; // Changes drag drop effect to move to show a coin can be inserted and to allow the drag drop to take place

}

ResetTimer(); // Resets timer as action taken

}

private void NoteDragEnter(object sender, DragEventArgs e) // Note slot check

{

if (tempString == "20" || tempString == "10" || tempString == "5") // If statement checks if temp string holds the value of a note using or

{

e.Effect = DragDropEffects.Move; // Changes drag drop effect to move to show a note can be inserted and to allow the drag drop to take place

}

else if (tempString == "1" || tempString == "50p" || tempString == "20p") // If statement checks if temp string holds the value of a coin using or

{

e.Effect = DragDropEffects.None; // Changes drag drop effect to none to show a coin can't be inserted into the note slot

}

ResetTimer(); // Resets timer as action taken

}

private void MoneyDragDrop(object sender, DragEventArgs e) // Function to handle DoDragDrop events

{

string moneyValue = (string)e.Data.GetData(DataFormats.Text); // Gets the value from sender and stores in money value

OrderPayment(moneyValue); // Calls the Order payment function and passes money value as money selection parameter

ResetTimer(); // Resets timer as action taken

}

private void BtnCheckout\_Click(object sender, EventArgs e) // Function to handle checkout button click

{

do // Do while loop always running as value doesn't change

{

if (orderList.Count >= 1) // If statement checks if order list count is greater than or equal to one

{

orderActive = false; // Sets order active to false to stop order screen

pnlMoneySelection.Enabled = true; // Enables money selection panel

pnlItemSelection.Enabled = false; // Disables item selection panel

txtCoinSlot.Enabled = true; // Enables the coin slot

txtNoteSlot.Enabled = true; // Enables the note slot

btnBackToOrder.Visible = true; // Enables the back button

}

else // Exception handler if no items in basket there is nothing to check out

{

MessageBox.Show("No items in basket. Add items to proceed to Check-out."); // Displays message box asking for items to be added before proceeding to check out screen

break; // Breaks out of do while loop to enable user to add items

}

}

while (false);

ResetTimer(); // Resets the timer as action taken

}

private void BtnCancelOrder\_Click(object sender, EventArgs e) // Function to handle cancel button click

{

do // Do while loop always running as value doesn't change

{

if (orderList.Count >= 1) // If statement check order list count is greater than or equal to 1

{

CancelOrder(); // Calls cancel order function

}

else // Else exception handler no items in basket so there is no order to cancel

{

MessageBox.Show("No Order Has Been Created. Add Some Items To Begin An Order."); // Displays message box saying no order has been created, add some items to begin order

break; // Break out of do while loop to enable user to add items

}

}

while (false);

ResetTimer(); // Reset timer as action taken

}

private void BtnBackToOrder\_Click(object sender, EventArgs e) // Function to handle back button click

{

pnlItemSelection.Enabled = true; // Enable the item selection panel

pnlMoneySelection.Enabled = false; // Disable the money selection panel

orderActive = true; // Set order active to true

btnBackToOrder.Visible = false; // Make the back to order button invisible

ResetTimer(); // Reset timer as action taken

}

private void BtnCompleteItemRemoval\_Click(object sender, EventArgs e) // Function to handle Complete Remove Button Click

{

int tempIndex; // Creates a variable to store a temporary index value

if (orderList.Contains("Item Name: " + itemName)) // If statement check if Item Name is within the Orderlist

{

tempIndex = orderList.IndexOf("Item Name: " + itemName) + 2; // Finds the index of item name and then increases that value by 2 to get the index of item price and stores in temp index

tempString = (orderList[tempIndex]); // Gets the value held at temp index value in orderlist and stores in temp string

itemPrice = Convert.ToDouble(tempString.Substring(12)); // Creates a substring of temp string to isolate the price of the item and stores in item price

}

else // Exception handler, shouldn't ever run is here to prevent crashes

{

MessageBox.Show("Item Selected is Not in Order List."); // Message box to show item selected is not in order list

}

if (index == 0) // If statement checks if index is 0 to find which item is being adjusted

{

itemQuantity = (int)nudQuantityToRemoveItem1.Value; // Gets the new item quantity from numeric up down value and stores it in item quantity

if (itemQuantity != 0) // If statement check to see if new quantity is not equal to 0

{

tempIndex = orderList.IndexOf("Item Quantity: " + previousItemQuantity); // Finds the index storing previous item quantity in order list

orderList.RemoveAt(tempIndex); // Removes the value at index of item quantity in order list

orderList.Insert(tempIndex, "Item Quantity: " + itemQuantity); // Inserts the new item quantity retrieved from the numeric up down value

lstQuantityInBasket.Items.RemoveAt(0); // Removes the index holding the previous quantity in quantity listbox

lstQuantityInBasket.Items.Insert(0, itemQuantity); // Inserts the new item quantity at the index

}

else // Else statement is if new item quantity is equal to 0

{

lstQuantityInBasket.Items.RemoveAt(0); // Remove the index holding the item quantity from item quantity listbox

lstItemsInBasket.Items.RemoveAt(0); // Remove the index holding item name from items in basket listbox

orderList.Remove("Item Name: " + itemName); // Removes the item name from orderlist

orderList.Remove("Item Quantity: " + previousItemQuantity); // Removes the item quantity from orderlist

orderList.Remove("Item Price: " + itemPrice); // Removes the item price from orderlist

}

}

else if (index == 1)

{

itemQuantity = (int)nudQuantityToRemoveItem2.Value;

if (itemQuantity != 0)

{

tempIndex = orderList.IndexOf("Item Quantity: " + previousItemQuantity);

orderList.RemoveAt(tempIndex);

orderList.Insert(tempIndex, "Item Quantity: " + itemQuantity);

lstQuantityInBasket.Items.RemoveAt(1);

lstQuantityInBasket.Items.Insert(1, itemQuantity);

}

else

{

lstQuantityInBasket.Items.RemoveAt(1);

lstItemsInBasket.Items.RemoveAt(1);

orderList.Remove("Item Name: " + itemName);

orderList.Remove("Item Quantity: " + previousItemQuantity);

orderList.Remove("Item Price: " + itemPrice);

}

}

else if (index == 2)

{

itemQuantity = (int)nudQuantityToRemoveItem3.Value;

if (itemQuantity != 0)

{

tempIndex = orderList.IndexOf("Item Quantity: " + previousItemQuantity);

orderList.RemoveAt(tempIndex);

orderList.Insert(tempIndex, "Item Quantity: " + itemQuantity);

lstQuantityInBasket.Items.RemoveAt(2);

lstQuantityInBasket.Items.Insert(2, itemQuantity);

}

else

{

lstQuantityInBasket.Items.RemoveAt(2);

lstItemsInBasket.Items.RemoveAt(2);

orderList.Remove("Item Name: " + itemName);

orderList.Remove("Item Quantity: " + previousItemQuantity);

orderList.Remove("Item Price: " + itemPrice);

}

}

else if (index == 3)

{

itemQuantity = (int)nudQuantityToRemoveItem4.Value;

if (itemQuantity != 0)

{

tempIndex = orderList.IndexOf("Item Quantity: " + previousItemQuantity);

orderList.RemoveAt(tempIndex);

orderList.Insert(tempIndex, "Item Quantity: " + itemQuantity);

lstQuantityInBasket.Items.RemoveAt(3);

lstQuantityInBasket.Items.Insert(3, itemQuantity);

}

else

{

lstQuantityInBasket.Items.RemoveAt(3);

lstItemsInBasket.Items.RemoveAt(3);

orderList.Remove("Item Name: " + itemName);

orderList.Remove("Item Quantity: " + previousItemQuantity);

orderList.Remove("Item Price: " + itemPrice);

}

}

else if (index == 4)

{

itemQuantity = (int)nudQuantityToRemoveItem5.Value;

if (itemQuantity != 0)

{

tempIndex = orderList.IndexOf("Item Quantity: " + previousItemQuantity);

orderList.RemoveAt(tempIndex);

orderList.Insert(tempIndex, "Item Quantity: " + itemQuantity);

lstQuantityInBasket.Items.RemoveAt(4);

lstQuantityInBasket.Items.Insert(4, itemQuantity);

}

else

{

lstQuantityInBasket.Items.RemoveAt(4);

lstItemsInBasket.Items.RemoveAt(4);

orderList.Remove("Item Name: " + itemName);

orderList.Remove("Item Quantity: " + previousItemQuantity);

orderList.Remove("Item Price: " + itemPrice);

}

}

else if (index == 5)

{

itemQuantity = (int)nudQuantityToRemoveItem6.Value;

if (itemQuantity != 0)

{

tempIndex = orderList.IndexOf("Item Quantity: " + previousItemQuantity);

orderList.RemoveAt(tempIndex);

orderList.Insert(tempIndex, "Item Quantity: " + itemQuantity);

lstQuantityInBasket.Items.RemoveAt(5);

lstQuantityInBasket.Items.Insert(5, itemQuantity);

}

else

{

lstQuantityInBasket.Items.RemoveAt(5);

lstItemsInBasket.Items.RemoveAt(5);

orderList.Remove("Item Name: " + itemName);

orderList.Remove("Item Quantity: " + previousItemQuantity);

orderList.Remove("Item Price: " + itemPrice);

}

}

totalItemQuantity -= previousItemQuantity; // Decreases total item quantity by previous item quantity

totalItemQuantity += itemQuantity; // Increases total item quantity by new item quantity

orderTotal -= previousItemQuantity \* itemPrice; // Decreases order total by previous quantity multipled by item price (removing the previous total item cost from order)

orderTotal += itemQuantity \* itemPrice; // Increases order total by new item quantity multiplied by item price (adding the new total item cost to order)

tenderDue -= previousItemQuantity \* itemPrice; // Decreases tender due by previous quantity multipled by item price

tenderDue += itemQuantity \* itemPrice; // Increases tender due by new item quantity multipled by item price

DisplayValues(totalItemQuantity, orderTotal, tenderDue); // Call Display Values function passing total item quantity, order total and tender due to Update the listboxes across the form with new values

lblQuantityToRemove.Visible = false; // Hides all remove item panels, numeric up downs and buttons

pnlQuantityToRemove.Visible = false;

btnCompleteItemRemoval.Visible = false;

nudQuantityToRemoveItem1.Visible = false;

nudQuantityToRemoveItem2.Visible = false;

nudQuantityToRemoveItem3.Visible = false;

nudQuantityToRemoveItem4.Visible = false;

nudQuantityToRemoveItem5.Visible = false;

nudQuantityToRemoveItem6.Visible = false;

ResetTimer(); // Resets timer as action has been taken

}

private void ItemListBox\_MouseDown(object sender, MouseEventArgs e) // Function to handle the mouse dragging objects from the listbox

{

if (orderList.Count >= 1) //If statement checks order list has values if it does start item removal process else show message box error

{

index = lstItemsInBasket.SelectedIndex; // Gets the index value of the selected item in basket

itemName = lstItemsInBasket.SelectedItem.ToString(); // Gets the item name of the selected item in basket

lstQuantityInBasket.SelectedIndex = index; // Gets the index of item quantity from quantity listbox

previousItemQuantity = Convert.ToInt32(lstQuantityInBasket.SelectedItem); // Gets the previous item quantity

lstItemsInBasket.DoDragDrop(itemName, DragDropEffects.Move); // Completes do drag drop operation

}

else // Else statement shows error message as an exception if no items have been added

{

MessageBox.Show("No Items Have Been Added, Add Some Items First.");

}

ResetTimer(); // Resets timer as action has been taken

}

private void RemoveItem\_DragEnter(object sender, DragEventArgs e) // Function to handle Drag enter event

{

e.Effect = DragDropEffects.Move; // Sets the drag drop effect to move

ResetTimer(); // Resets timer as action has been taken

}

private void RemoveItem\_DragDrop(object sender, DragEventArgs e) // Function to handle the drag drop event

{

RemoveItem(itemName, previousItemQuantity); // Calls Remove item function passing item name and previous item quantity to the function

ResetTimer(); // Resets timer as action has been taken

}

private void ResetTimer() // Function to reset the time out timer

{

timerForFormTimeOut.Stop(); // Timeout timer stop

timerForFormTimeOut.Start(); // Timeout timer start

}

private void TimerForTimeOut\_Tick(object sender, EventArgs e) // Function to handle the timeout timer running out or ticking

{

timerForTimeOutMessageBox.Enabled = true; // Enables the timer for the timeout message box to auto close the program

timerForTimeOutMessageBox.Start(); // Starts the timer for the timeout message box

string message = "No Actions Performed For A Long Time. Do You Want To Continue?"; // Timeout message box message

string title = "Timeout Warning"; // Timeout message box title

MessageBoxButtons buttons = MessageBoxButtons.YesNo; // Timeout message box will have yes/no button selection

DialogResult result = MessageBox.Show(message, title, buttons); // Displays the message box storing result from key press in result

switch (result) // Switch statement to control yes or no used this for the break if yes was pressed

{

case DialogResult.Yes: // If yes pressed resets timer, stops the timer for the timeout messagebox and disables it then breaks from the switch statement

ResetTimer();

timerForTimeOutMessageBox.Stop();

timerForTimeOutMessageBox.Enabled = false;

break;

case DialogResult.No: // If no pressed closes the form, exits the application and breaks from the switch statement

Close();

Application.Exit();

break;

}

}

private void TimerForTimeOutMessageBox\_Tick(object sender, EventArgs e) // Function to handle the timer for timeout message box running out or ticking

{

Close(); // Close the form

Application.Exit(); // Exits from the application

}

private void Form1\_Click(object sender, EventArgs e) // Function to handle clicks on the form to reset timer

{

ResetTimer(); // Calls the reset timer function

}

private void Form1\_MouseMove\_1(object sender, MouseEventArgs e) // Function to handle mouse movement on the form to reset timer

{

ResetTimer(); // Calls the reset timer function

}

}

}