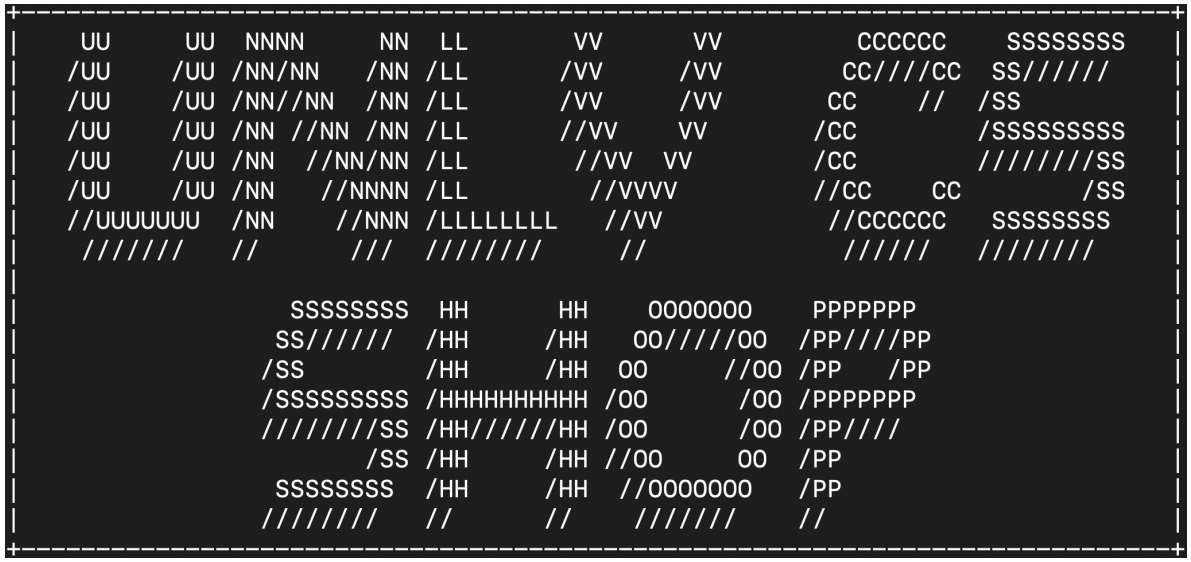
CS 135
Computer Science I

The UNLV CS Shop



For use by:
School of Computer Science
Howard Hughes College of Engineering
University of Nevada, Las Vegas

Created and Authored by:
Alex St. Aubin © 2022

Unauthorized reproductions of this
handout are strictly forbidden.
© 2022 Alex St. Aubin. All Rights Reserved.

Academic Integrity Policy

To students completing this handout as an assignment,

Let's face it, Computer Science is tough. You will run into problems you need help solving, **this is absolutely okay**. When you run into these problems we want you to feel comfortable coming to us for help. The resources provided to you by the university for you to get help are:

1. Information/code provided in the textbook.
2. Information/code provided in the class notes (from class or on Canvas).
3. Assistance provided by the course instructor.
4. Assistance provided by the course teaching assistants.
5. Assistance provided by the College of Engineering Tutoring Center, as long as the tutor is not writing your code.
6. Assistance from any other resource provided from the university, as long as the resource is not doing your assignment.

The internet is also a wealth of knowledge for us. There are many resources across the internet that help making coding easier to understand. These resources are:

1. Pages showing/explaining formulas that are needed to perform calculations.
2. Online calculators to verify the accuracy of calculations.
3. Documentation provided by the writers of languages, libraries, packages, APIs, etc.
4. Websites like cplusplus.com which explain syntax in easier terms than documentation.

There is however a clear line between seeking help and academic misconduct. Academic misconduct is where you use assistance outside the authorized resources listed by your class instructor to come about the answer/solution to an assignment, lab, exam etc. Academic integrity is one of the most difficult problems faced in a university setting. When you act in an academically dishonest manner not only does it affect your ability to learn, but it also affects the faculty which has to sign off on the harsh consequences that come with academic misconduct. Some examples of academic misconduct are (but not limited to):

1. Asking and/or paying someone to complete your assignments or agreeing to complete assignments for someone else.
2. Copying answers/code from a fellow student, friend, relative.
3. Providing/sharing answers/code to fellow student(s).
4. Using the Internet to develop a strategy for solving a problem (finding an algorithm/pseudo code).
5. Using the Internet to find an example of code that solves a portion of an assignment.
6. There is no difference between copy and pasting the code or if the student types the code in using an editor.

If you cheat an Alleged Academic Misconduct Report will be submitted to the Office of Student Conduct and you will be subject to the following:

1st violation - Student(s) will receive a grade of zero on the assignment/examination.
2nd violation - A grade of **F** will be issued for the course; no further assignments/labs/exams can be completed for credit.

It should be noted that every semester we find in many CS classes that there are instances of academic misconduct involving websites where students upload part or all of their assignment to a website and pay for an answer. An example of one of these websites is Chegg. There is 0 tolerance when an assignment is found on one of these websites. When you post to these websites we most likely will find it. If we do find it and you are caught to have used the solution from one of these websites you will receive an automatic **F** for the entire class, no exceptions. If you drop the class after being caught for cheating the university will re-add you to the class and give you the **F**.

The cost of a university education is significant. Students are encouraged to use the resources provided to learn the material. The material in CS 135 is the foundation of your CS degree.

Do not cheat, it hurts more than just yourself.

(See the following page for an agreement to this policy)

Academic Integrity Policy Acknowledgement

By signing below I hearby accept and understand the entirety of the Academic Integrity Policy. I understand that breaking this policy will result in the consequences outlined in the policy.

I understand the entire policy is important but I would like to make note of:

1. If I need help I need to actively seek it from instructors, teaching assistants, and College of Engineering tutors.
2. If I am caught cheating I will be subject to the consequences laid out in the Academic Integrity Policy.
3. If I am caught paying for a solution or posting any part of any assignment/lab to solicit a solution on the internet on any website I will receive an automatic F for the entire class.

Sign and date this page and return it to your lecture instructor by the date they specify. Failure to do so will result in your assignments not being graded.

Student _____ Date _____

Introduction

Each month billions are spent in the retail industry making it one of the largest industries in the world. Because of this, large systems have been built to scale to the growing sales numbers in retail sales. In this assignment you will not be building some massive enterprise system, but you will simply learn how to solve some small retail problems using the C++ programming language.

In the assignments in this handout you will be building a system to manage the sales at a fictional store called The UNLV CS Shop. This fictional store sells all the essentials a student could possibly need for their CS degrees. From laptops to coffee, The UNLV CS Shop has what you need to get that degree.

The items that are sold at The UNLV CS Shop are:

Item	Cost	Item	Cost	Item	Cost
Laptop	$799.99	Speakers	$79.99	iPhone Charger	$9.99
Desktop	$949.99	Headphones	$99.99	Android Charger	$9.99
Monitor	$249.99	Microphone	$39.99	Laptop Charger	$19.99
Mouse	$15.95	Webcam	$24.98	Notebook	$1.49
Keyboard	$15.95	Flash Drive	$4.48	Pen	$0.99
Laptop Bag	$19.99	Hard Drive	$39.99	Pencil	$0.99
Laptop Case	$19.99	Solid State Drive	$69.95	Drinks	$1.29
Laptop Riser Stand	$24.99	CPU Upgrade	$299.99	Chips	$1.49
Laptop Lock	$14.99	RAM Upgrade	$149.99	Coffee	$0.99

Table 1: Items sold at The UNLV CS Shop

The assignments in this handout have been broken into 6 sections:

1. I/O and Arithmetic
2. Selection
3. Loops
4. Arrays
5. Functions
6. Structs

Each section will be assigned to you as an individual assignment. For each section you will be given a provided program contained in a `.cpp` file which is the file you will code in alongside this handout. Make sure to study the provided program given to you at the beginning of each section, it will help you understand what is going on and in many cases it will contain the answer to the previous section.

Contents

1 Input/Output and Arithmetic

In this section you will learn, in C++, how to read input and write output to the command line, use arithmetic, use the <cmath> library, and use the <iomanip> library.

Our store will sell many items. When these sales are made we will need to be able to calculate the subtotal, tax, and total of the items the customer wants. We will then need to show this data to the customer so they know how much they need to pay. We can then take the payment, calculate the change due, and print a receipt for the customer. In this section you will build a section to do exactly this.

Make sure to download the provided program main.cpp to code along with this section.

1.1 Capturing Input

To begin, we will take in some information for a single product that a customer wants to buy. We need 3 pieces of information about the item the customer wants: the name of the product, the cost of the product, and the quantity the customer wants of the product. To get these 3 pieces of information we can first prompt the user for the name using a cout[1] statement, then read in the name using getline[2]. getline is used for the name input since the name is a string and it may have spaces in it (remember cin[3] cannot read in spaces). To get the cost the user can be prompted to enter a cost using a cout statement, and then the cost can be read in using a cin statement). The same process used for cost can then be used to read in qty.

You will need to read in 3 pieces of data into 3 separate variables of varying data types. The variables have already been declared in the provided program. These variables are:

data type	identifier	description
string	name	name of the item
double	cost	cost of the item
integer	qty	quantity wanted of the item

Table 2: Variables to be used in section 1.1

An example of how your prompts/input should look like follows.

```
Item Name: Example
Item Cost: 5.99
Quantity:  1
```

Figure 1: Prompts/inputs for name/cost/qty

See this video for help with data types: www.youtube.com/watch?v=KN78kXrdqxA
See this video for help with variables: www.youtube.com/watch?v=B27K3OB9fVA
See this video for help with input/output: www.youtube.com/watch?v=42n9YyKYJMo

[1] https://www.cplusplus.com/reference/iostream/cout/
[2] https://www.cplusplus.com/reference/string/string/getline/
[3] https://www.cplusplus.com/reference/iostream/cin/

1.2 Using Arithmetic

Now that we know the `cost` and the `qty` wanted of the item we can calculate 3 pieces of data: the subtotal, the tax charge, and the total. We can use arithmetic to first calculate the `subtotal`, then the `tax` using the `subtotal`, then the `total` using the `subtotal` and `tax`. Three variables have already been declared in the provided program which you can use to save these numbers. These variables are:

data type	identifier	description
double	subtotal	subtotal for qty of the item
double	tax	tax to be charged for qty of the item
double	total	total price including tax for qty of the item

Table 3: Variables to be used in section 1.2

To calculate the `subtotal` we will use the formula:

$$subtotal = cost * qty$$

Once we calculate the `subtotal` we can calculate the `tax` that needs to be charged. We will use the sales tax in Nevada of 8.25%. A global constant `TAX_RATE` has been provided at the top of the provided program, use this global constant to calculate the amount of tax you need to charge. The formula for how much `tax` needs to be charges is:

$$tax = subtotal * TAX\_RATE$$

As we know money only goes to the hundredths place (i.e. 0.99), but the number we just calculated for the `tax` could trail on much past the hundredths place (i.e. 0.99999999...). To fix this problem we must round our result using the below formula. It should be noted though that you must use the `<cmath>`[4] library to use the `ceil()`[5] function contained within this formula. To use this library make sure to `#include` it at the top of your program. The formula to round a floating point number to the nearest hundredth is:

$$tax = ceil(tax * 100.0)/100.0$$

Lastly we need to calculate the `total` including `tax` using the formula:

$$total = subtotal + tax$$

We can then output the `subtotal`, `tax`, and `total` amount the customer must pay using `cout` statements so that the customer knows how much they have to pay in the next section. An example of how your output should look combined with the output from section 1.1 follows.

```
Item Name: Example
Item Cost: 5.99
Quantity:  1

Subtotal:  5.99
Tax:       0.50
Total:     6.49
```

Figure 2: Combined input/output for sections 1.1 & 1.2

See this video for help with arithmetic: `www.youtube.com/watch?v=f-l0po70XKQ`
See this video for help with the `<cmath>` library: `www.youtube.com/watch?v=zkQhBECLBd0`

[4] `https://www.cplusplus.com/reference/cmath/`
[5] `https://www.cplusplus.com/reference/cmath/ceil/`

1.3 Calculating Change

Now that the `total` to be charged is being displayed we can get the payment from the customer and enter it into our program. Just as we did in 1.1, we will use `cout` statements to prompt for the amount of money `tendered` from the customer, then use `cin` statements to read the `tendered` amount in. We can then calculate the `change` using the `total` and the `tendered` amount. The variables for these amounts have already been declared in the provided program. These variables are:

data type	identifier	description
double	tendered	payment amount from the customer
double	change	change owed to the customer

Table 4: Variables to be used in section 1.3

Once `tendered` has been read in you can calculate the `change` owed to the customer using the formula:

$$change = tendered - total$$

An example of how your prompts/input should look like follows.

```
Item Name: Example
Item Cost: 5.99
Quantity:  1

Subtotal:  5.99
Tax:       0.50
Total:     6.49

Amount tendered from customer:  10

Change Due: 3.51
```

Figure 3: Combined input/output for sections 1.1, 1.2, & 1.3

1.4 Output Manipulation

Lastly, everything that has been input/calculated must be printed out on a receipt. You will see in the provided program the beginning of the printed receipt after the comment `// Print the receipt`. After this portion you will need to output all the input and calculated data using the `<iomanip>`[6] library to format everything properly like a receipt. Specifically you will need to use `setw()`[7] to set the field widths of outputs, `setfill()`[8] to set the character output in the white space of the buffer `setw()` provides, and `left`[9] and `right`[10] to justify output to the left and right in the `setw()` buffers.

To output the item `name`, `cost` and `qty` you will need to first output the string `"| "`. You can then justify the output `left`, set a field width of `26`, and output the `name`. You then need to justify the output `right`, set a field width of `6`, and then output the `cost`. Then you can output the string `" x "` followed by the `qty` and the string `" |\n"`. You can then output a blank line using the same code I used one line after the phone number in the receipt.

[6] https://www.cplusplus.com/reference/iomanip/
[7] https://www.cplusplus.com/reference/iomanip/setw/
[8] https://www.cplusplus.com/reference/iomanip/setfill/
[9] https://www.cplusplus.com/reference/ios/left/
[10] https://www.cplusplus.com/reference/ios/right/

You then need to output the `subtotal`, `TAX_RATE`, `tax`, and `total`. For the `subtotal` you can output the string `"| Subtotal "`, set a field width of 27, output the `subtotal`, then output the string `" |\n"`. For the `TAX_RATE` you can output the string `"| Tax Rate "` set a field width of 26, output the `Tax_Rate * 100`, then output the string `"% |\n"`. For the `tax` you can output the string `"| Tax "`, set a field width of 32, output the `tax`, then output the string `" |\n"`. And lastly, for the `total` you can output the string `"| Total "`, set a field width of 30, output the `total`, then output the string `" |\n"`. You can then output a blank line using the same code I used one line after the phone number in the receipt.

Lastly, you need to output the `tendered` amount and the `change` due. For the `tendered` amount you can output the string `"| Tendered "`, set a field width of 27, output the `tendered` amount, then output the string `" |\n"`. For the `change` due you can output the string `"| Change "`, set a field width of 29, output the `change` due, then output the string `" |\n"`. The code to then print the bottom of the receipt is already included in the provided program.

Your output receipt should look similar to the following (with differing numbers/item names depending on the values input):

```
/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
|                                              |
|                UNLV CS Shop                  |
|            4505 S Maryland Pkwy              |
|             Las Vegas,NV 89154               |
|               (702) 895-3011                |
|                                              |
| Example                        5.99 x 1      |
|                                              |
| Subtotal                              5.99   |
| Tax Rate                              8.25%  |
| Tax                                   0.50   |
| Total                                 6.49   |
|                                              |
| Tendered                             10.00   |
| Change                                3.51   |
|                                              |
\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
```

Figure 4: Example receipt

See this video for help with the `<iomanip>` library: `www.youtube.com/watch?v=JAEKyNfqm0A`

(See the next page for example executions)

1.5 Example Output

Below are sample executions of the complete program. Text in red is input from the user, this coloring is simply here to distinguish components and not needed in your program.

Example **1**

```
Alexs-iMac desktop % g++ main.cpp
Alexs-iMac desktop % ./a.out
+--------------------------------------------------------------------------------+
|    UU     UU  NNNN     NN  LL        VV       VV          CCCCCC    SSSSSSSS   |
|   /UU    /UU /NN/NN   /NN /LL       /VV     /VV         CC////CC  SS//////    |
|   /UU    /UU /NN//NN  /NN /LL       /VV     /VV        CC    //  /SS          |
|   /UU    /UU /NN //NN /NN /LL       //VV    VV        /CC        /SSSSSSSSS   |
|   /UU    /UU /NN  //NN/NN /LL        //VV  VV         /CC        ////////SS   |
|   /UU    /UU /NN   //NNNN /LL         //VVVV          //CC    CC         /SS   |
|   //UUUUUUU  /NN    //NNN /LLLLLLLL   //VV            //CCCCCC   SSSSSSSS    |
|    ///////   //      ///  ////////     //              //////   ////////     |
|                                                                                |
|                   SSSSSSSS  HH      HH    OOOOOOO    PPPPPPP                 |
|                  SS//////  /HH     /HH   OO/////OO  /PP////PP                |
|                 /SS        /HH     /HH  OO     //OO /PP   /PP                |
|                 /SSSSSSSSS /HHHHHHHHHH /OO       /OO /PPPPPPP                |
|                 ////////SS /HH//////HH /OO       /OO /PP////                 |
|                        /SS /HH     /HH //OO      OO  /PP                      |
|                  SSSSSSSS  /HH     /HH  //OOOOOOO   /PP                      |
|                 ////////   //      //    ///////    //                       |
+--------------------------------------------------------------------------------+
Item Name: Example
Item Cost: 5.99
Quantity:  1

Subtotal:  5.99
Tax:       0.50
Total:     6.49

Amount tendered from customer: 10

Change Due: 3.51

/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
|                                                                              |
|                              UNLV CS Shop                                    |
|                          4505 S Maryland Pkwy                                |
|                           Las Vegas,NV 89154                                 |
|                             (702) 895-3011                                   |
|                                                                              |
| Example                                                  5.99 x 1            |
|                                                                              |
| Subtotal                                                              5.99   |
| Tax Rate                                                             8.25%   |
| Tax                                                                   0.50   |
| Total                                                                 6.49   |
|                                                                              |
| Tendered                                                             10.00   |
| Change                                                                3.51   |
|                                                                              |
\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
```

Example **2**

```
Alexs-iMac desktop % ./a.out
+--------------------------------------------------------------------------------+
|    UU     UU  NNNN     NN  LL        VV       VV          CCCCCC    SSSSSSSS   |
|   /UU    /UU /NN/NN   /NN /LL       /VV     /VV         CC////CC  SS//////    |
|   /UU    /UU /NN//NN  /NN /LL       /VV     /VV        CC    //  /SS          |
|   /UU    /UU /NN //NN /NN /LL       //VV    VV        /CC        /SSSSSSSSS   |
|   /UU    /UU /NN  //NN/NN /LL        //VV  VV         /CC        ////////SS   |
|   /UU    /UU /NN   //NNNN /LL         //VVVV          //CC    CC         /SS   |
|   //UUUUUUU  /NN    //NNN /LLLLLLLL   //VV            //CCCCCC   SSSSSSSS    |
|    ///////   //      ///  ////////     //              //////   ////////     |
|                                                                                |
|                   SSSSSSSS  HH      HH    OOOOOOO    PPPPPPP                 |
|                  SS//////  /HH     /HH   OO/////OO  /PP////PP                |
|                 /SS        /HH     /HH  OO     //OO /PP   /PP                |
|                 /SSSSSSSSS /HHHHHHHHHH /OO      /OO /PPPPPPP                 |
|                 ////////SS /HH//////HH /OO      /OO /PP////                  |
|                        /SS /HH     /HH //OO     OO  /PP                      |
|                  SSSSSSSS  /HH     /HH  //OOOOOOO   /PP                      |
|                 ////////   //      //    ///////    //                       |
+--------------------------------------------------------------------------------+
Item Name: Laptop
Item Cost: 799.99
Quantity:  3

Subtotal:  2399.97
Tax:       198.00
Total:     2597.97

Amount tendered from customer: 2600

Change Due: 2.03

/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
|                                                                              |
|                UNLV CS Shop                                                  |
|            4505 S Maryland Pkwy                                              |
|             Las Vegas,NV 89154                                               |
|               (702) 895-3011                                                 |
|                                                                              |
| Laptop                                                  799.99 x 3           |
|                                                                              |
| Subtotal                                                          2399.97    |
| Tax Rate                                                             8.25%   |
| Tax                                                               198.00     |
| Total                                                             2597.97    |
|                                                                              |
| Tendered                                                          2600.00    |
| Change                                                               2.03    |
|                                                                              |
\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
```

1.6 Submission

The first thing you need to do before submitting this assignment is read and understand the **Academic Integrity Policy** at the beginning of this handout. By submitting this assignment you are accepting this policy even if you did not sign/submit the required **Academic Integrity Policy Acknowledgement**.

Make sure your code is saved as **main.cpp**. Do not ignore this step or save your file with different names.

Your program source code must be submitted via **CodeGrade** as a properly named **.cpp** file. **Late submissions will not be accepted**.

2 Selection

In this section you will learn, in `C++`, how to use selection (if...else and switch), check numerical input for input error, and check input is within certain bounds.

In the previous section we built a system that can handle single item cash transactions. This system has many missing features to it though. What if the user wants to pay with a card instead of using cash? This is not possible in the state the system is currently in. What if someone pays less than the total? Our previous program would print a receipt saying they get negative change. In this section we will fix many bugs and add some features related to selection.

Make sure to download the provided program `main.cpp` to code along with this section.

2.1 Bounds Checking

We first need to make sure that our four inputs are in bounds. The 4 inputs we must keep in bounds are:

1. The length of the `name` of the product can not be 0 otherwise it would not have a name. Also, on our receipt the output for the `name` has a field width of 26 so we will limit the length of a product's name `name` to 25. For these reasons the `name` must be in the bounds:

$$0 < name.length() < 26$$

2. The `cost` of the item can not be negative, we do not want to pay anyone for taking from our store. The `cost` must also be less than \$1000.00 since we only have a field width of 6 for the `cost` on our receipt. For these reasons the `cost` must be in the bounds:

$$0.00 \leq cost < 1000.00$$

3. The `qty` wanted of the item must be greater than 0 otherwise the customer isn't buying anything. The `qty` must also be less than 10 since we only have a field width of 1 for the `qty` on our receipt. For these reasons the `qty` must be in the bounds:

$$0 < qty < 10$$

4. The `tendered` amount must be at least the amount of the total, otherwise the customer still owes us money.For these reasons the `tendered` amount must be in the bounds:

$$total \leq tendered$$

These bounds can by checked by building a **logical expression** and testing it using an `if...else` statement after each of the 4 inputs, using the logic explained above for each input respectively. If an out of bounds value is input an appropriate error message should be displayed and the program should be terminated using `exit(0)`[11].

See this video for help with `relational expressions`: `www.youtube.com/watch?v=GDuZ-pFZ9V8`
See this video for help with `logical operators`: `www.youtube.com/watch?v=oVBp92Osgu8`
See this video for help with `logical expressions`: `www.youtube.com/watch?v=rf5OHqDi3v0`
See this video for help with `relational expressions`: `www.youtube.com/watch?v=vGu7DLYRvAE`

[11] `www.cplusplus.com/reference/cstdlib/exit/`

Examples of how your error messages should look follow.

```
Item Name: This name is too long for the program
Error: Invalid name.
```

Figure 5: Error on the `name` input

```
Item Name: Example
Item Cost: 1000.00
Error: Invalid cost.
```

Figure 6: Error on the `cost` input

```
Item Name: Example
Item Cost: 5.99
Quantity:  0
Error: Invalid quantity.
```

Figure 7: Error on the `qty` input

```
Item Name: Example
Item Cost: 5.99
Quantity:  1

Subtotal:  5.99
Tax:       0.50
Total:     6.49

Amount tendered from customer: 5
Error: Invalid tendered amount
```

Figure 8: Error on the `tendered` input

2.2 Input Failure

We now need to make sure that when we take in numerical inputs we are checking for input failure. Input failure occurs on numerical inputs when the user enters characters that are not valid numerical input (any character besides '0'-'1' and '.'). There are 3 places where we need to add this error checking:

1. When the `cost` of the item is read in.
2. When the `qty` wanted is read in.
3. When the `tendered` amount is read in.

To add these checks we can just add onto the logical expressions from part 2.1. To do this add a check for `cin.fail()`[12] to each of the 3 numerical inputs. This will cause the program to print the same error message as when inputs are out of bounds and then terminate.

See this video for help with `input failure`: `www.youtube.com/watch?v=HaFjFFACGfU`

[12] `https://www.cplusplus.com/reference/ios/ios/fail/`

2.3 Cash or Card Payment

We now want to add the feature of being able to select between cash and card payments. To do this we will get a selection from the user by prompting for and reading in a character. You will need to add a variable to `main()` to hold this character selection. You can then prompt the user to enter a character and read it in. You will add this prompt just before getting the amount tendered from the customer. There is a comment in `main()` which reads `// 2.3 TODO: Get cash/card selection`, get the user's selection there. Your prompt should look as follows:

```
(Getting name/cost/qty truncated for space)

Subtotal:  5.99
Tax:       0.50
Total:     6.49

(C/c) - Cash
(D/d) - Card
Selection:
```

Figure 9: Waiting for cash/card input

We can then add a switch statement which will switch based on the character that the user input. This switch statement will have 5 cases:

1. The user enters `'c'`
2. The user enters `'C'`
3. The user enters `'d'`
4. The user enters `'D'`
5. The default case for when the user enters any other character

See this video for help with `switch statements`: `www.youtube.com/watch?v=5WLPqzjihC0`

2.3.1 Cash

In the case `'c'` or `'C'` is entered the user has selected cash. All of the logic for getting the `tendered` amount, calculating/displaying the `change`, and printing the cash receipt should be ran. Simply cut the code for doing this that is already there and paste it after the cases for `'c'` and `'C'` in the switch statement. Don't forget to follow this code with a `break` statement so the cash case doesn't continue to the card case.

(See the next page for an example)

```
(Getting name/cost/qty truncated for space)

Subtotal:  5.99
Tax:       0.50
Total:     6.49

(C/c) - Cash
(D/d) - Card
Selection: c

Amount tendered from customer: 10

Change Due: 3.51

(cash receipt truncated for space)
```

Figure 10: Cash selected by the user

2.3.2 Card

In the case `'d'` or `'D'` is entered the user has selected card. Logic will needed to be added after these two cases that:

1. Prompts and reads in a card number.
2. Checks if the card number is 16 digits.
3. Prints a credit card receipt.

To do this a string variable is needed to hold the `cardNumber`. The `cardNumber` can then be prompted for and read in the same way the previous inputs have.

```
(Getting name/cost/qty truncated for space)

Subtotal:  5.99
Tax:       0.50
Total:     6.49

(C/c) - Cash
(D/d) - Card
Selection: d

Card number:
```

Figure 11: Card selected/waiting for card number

Once the `cardNumber` has been input it's length then needs to be checked as all card numbers are 16 digits in length. If the length of the card number is not 16 then it is known that it is invalid so an error can be output and the program can be terminated.

(See the next page for an example)

```
(Getting name/cost/qty truncated for space)

Subtotal:  5.99
Tax:       0.50
Total:     6.49

(C/c) - Cash
(D/d) - Card
Selection: d

Card number: 1234
Error: Invalid card number.
```

Figure 12: Invalid card number length

The final thing that needs to be done for cards is a card receipt needs to be printed. The only difference between the cash and card receipt is the cash receipt has 2 values on the final line: `tendered` and `change`. These numbers are not used with cards though, instead the card receipt has a line that says a card was used, and a line that displays the last 4 digits of the card number. To do this simply copy/paste the cash receipt into your card case and modify the last 2 lines to show this information. To display the last 4 digits of the card simply `cout` characters 12-15 of the card number.

```
(Getting name/cost/qty truncated for space)

(C/c) - Cash
(D/d) - Card
Selection: d

Card number: 4123456789876543

/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
|                                                                              |
|                             UNLV CS Shop                                     |
|                         4505 S Maryland Pkwy                                 |
|                          Las Vegas,NV 89154                                  |
|                            (702) 895-3011                                    |
|                                                                              |
| Example                                                    5.99 x 1          |
|                                                                              |
| Subtotal                                                              5.99   |
| Tax Rate                                                              8.25%  |
| Tax                                                                   0.50   |
| Total                                                                 6.49   |
|                                                                              |
| Card Sale                                                                    |
| Card                                                  XXXXXXXXXXXX6543       |
|                                                                              |
\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
```

Figure 13: Card selected by the user

2.3.3 Invalid Selection

Any input that is not `'c'`, `'C'`, `'d'`, or `'D'` is invalid. To deal with this invalid selection the default case of the switch statement must be ran. Simply output an error message and terminate the program if an invalid selection is received.

```
(Getting name/cost/qty truncated for space)

(C/c) - Cash
(D/d) - Card
Selection: e
Error: Invalid selection.
```

Figure 14: Card selected by the user

2.4 Necessary Errors

A table of every error your program should output follows.

description	error message
name too short/long	Error: Invalid name.
cost too little/much/invalid	Error: Invalid cost.
qty too little/much/invalid	Error: Invalid quantity.
invalid cash/card selection	Error: Invalid selection.
amount tendered too little/invalid	Error: Invalid amount tendered.
card number too short/long	Error: Invalid card number.

Table 5: Errors messages

2.5 Example Output

Below are sample executions of the complete program. Text in red is input from the user, this coloring is simply here to distinguish components and not needed in your program.

Example **1** (name too short)

```
Alexs-iMac desktop % g++ AS2.cpp
Alexs-iMac desktop % ./a.out
+--------------------------------------------------------------------------------+
|    UU     UU  NNNN     NN  LL        VV       VV         CCCCCC    SSSSSSSS   |
|   /UU    /UU /NN/NN   /NN /LL       /VV     /VV        CC////CC  SS//////    |
|   /UU    /UU /NN//NN  /NN /LL       /VV     /VV       CC    //  /SS          |
|   /UU    /UU /NN //NN /NN /LL       //VV    VV       /CC        /SSSSSSSSS   |
|   /UU    /UU /NN  //NN/NN /LL        //VV  VV        /CC        ////////SS   |
|   /UU    /UU /NN   //NNNN /LL         //VVVV         //CC    CC         /SS   |
|   //UUUUUUU  /NN    //NNN /LLLLLLLL   //VV           //CCCCCC   SSSSSSSS    |
|    ///////   //      ///  ////////     //             //////   ////////     |
|                                                                                |
|                   SSSSSSSS  HH      HH    OOOOOOO    PPPPPPP                 |
|                  SS//////  /HH     /HH   OO/////OO  /PP////PP                |
|                 /SS        /HH     /HH  OO     //OO /PP   /PP                |
|                 /SSSSSSSSS /HHHHHHHHHH /OO       /OO /PPPPPPP               |
|                 ////////SS /HH//////HH /OO       /OO /PP////                |
|                        /SS /HH     /HH //OO      OO  /PP                     |
|                  SSSSSSSS  /HH     /HH  //OOOOOOO   /PP                     |
|                 ////////   //      //    ///////    //                      |
+--------------------------------------------------------------------------------+
Item Name:
Error: Invalid name.
```

Example **2** (name too long)

```
Alexs-iMac desktop % ./a.out
(header truncated for space)

Item Name: This name is too long for the program
Error: Invalid name.
```

Example **3** (negative cost)

```
Alexs-iMac desktop % ./a.out
(header truncated for space)

Item Name: Example
Item Cost: -1
Error: Invalid cost.
```

Example **4** (cost too large)

```
Alexs-iMac desktop % ./a.out
(header truncated for space)

Item Name: Example
Item Cost: 1000
Error: Invalid cost.
```

Example **5** (cost input failure)

```
Alexs-iMac desktop % ./a.out
(header truncated for space)

Item Name: Example
Item Cost: error
Error: Invalid cost.
```

Example **6** (qty too small)

```
Alexs-iMac desktop % ./a.out
(header truncated for space)

Item Name: Example
Item Cost: 5.99
Quantity:  0
Error: Invalid quantity.
```

Example **7** (qty too large)

```
Alexs-iMac desktop % ./a.out
(header truncated for space)

Item Name: Example
Item Cost: 5.99
Quantity:  10
Error: Invalid quantity.
```

Example **8** (qty input failure)

```
Alexs-iMac desktop % ./a.out
(header truncated for space)

Item Name: Example
Item Cost: 5.99
Quantity:  error
Error: Invalid quantity.
```

Example **9** (invalid selection)

```
Alexs-iMac desktop % ./a.out
(header truncated for space)

Item Name: Example
Item Cost: 5.99
Quantity:  1

Subtotal:  5.99
Tax:       0.50
Total:     6.49

(C/c) - Cash
(D/d) - Card
Selection: e
Error: Invalid selection
```

Example **10** (cash selected, not enough given)

```
Alexs-iMac desktop % ./a.out
(header truncated for space)

Item Name: Example
Item Cost: 5.99
Quantity:  1

Subtotal:  5.99
Tax:       0.50
Total:     6.49

(C/c) - Cash
(D/d) - Card
Selection: c

Amount tendered from customer: 5
Error: Invalid amount tendered.
```

Example **11** (cash selected, input error)

```
Alexs-iMac desktop % ./a.out
(header truncated for space)

Item Name: Example
Item Cost: 5.99
Quantity:  1

Subtotal:  5.99
Tax:       0.50
Total:     6.49

(C/c) - Cash
(D/d) - Card
Selection: c

Amount tendered from customer: error
Error: Invalid amount tendered.
```

Example **12** (cash selected)

```
Alexs-iMac desktop % ./a.out
(header truncated for space)

Item Name: Example
Item Cost: 5.99
Quantity:  1

Subtotal:  5.99
Tax:       0.50
Total:     6.49

(C/c) - Cash
(D/d) - Card
Selection: c

Amount tendered from customer: 10

Change Due: 3.51

/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
|                                                       |
|              UNLV CS Shop                             |
|          4505 S Maryland Pkwy                         |
|           Las Vegas,NV 89154                          |
|             (702) 895-3011                            |
|                                                       |
| Example                        5.99 x 1               |
|                                                       |
| Subtotal                                  5.99        |
| Tax Rate                                 8.25%        |
| Tax                                       0.50        |
| Total                                     6.49        |
|                                                       |
| Tendered                                 10.00        |
| Change                                    3.51        |
|                                                       |
\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
```

Example **13** (card selected, card number too short)

```
Alexs-iMac desktop % ./a.out
(header truncated for space)

Item Name: Example
Item Cost: 5.99
Quantity:  1

Subtotal:  5.99
Tax:       0.50
Total:     6.49

(C/c) - Cash
(D/d) - Card
Selection: d

Card number: 1234
Error: Invalid card number.
```

Example **14** (card selected, card number too long)

```
Alexs-iMac desktop % ./a.out
(header truncated for space)

Item Name: Example
Item Cost: 5.99
Quantity:  1

Subtotal:  5.99
Tax:       0.50
Total:     6.49

(C/c) - Cash
(D/d) - Card
Selection: D

Card number: 412345678987654321
Error: Invalid card number.
```

Example **15** (card selected)

```
Alexs-iMac desktop % ./a.out
(header truncated for space)

Item Name: Example
Item Cost: 5.99
Quantity:  1

Subtotal:  5.99
Tax:       0.50
Total:     6.49

(C/c) - Cash
(D/d) - Card
Selection: D

Card number: 4123456789876543

/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
|                                                      |
|              UNLV CS Shop                            |
|          4505 S Maryland Pkwy                        |
|           Las Vegas,NV 89154                         |
|             (702) 895-3011                           |
|                                                      |
| Example                          5.99 x 1            |
|                                                      |
| Subtotal                                    5.99     |
| Tax Rate                                   8.25%     |
| Tax                                         0.50     |
| Total                                       6.49     |
|                                                      |
| Card Sale                                            |
| Card                      XXXXXXXXXXXX6543           |
|                                                      |
\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
```

2.6 Submission

The first thing you need to do before submitting this assignment is read and understand the **Academic Integrity Policy** at the beginning of this handout. By submitting this assignment you are accepting this policy even if you did not sign/submit the required **Academic Integrity Policy Acknowledgement**.

Make sure your code is saved as **main.cpp**. Do not ignore this step or save your file with different names.

Your program source code must be submitted via **CodeGrade** as a properly named **.cpp** file. **Late submissions will not be accepted**.

3 Loops

In this section you will learn, in C++, how to use loops (while, do...while, for), loop until valid input is received, continuously loop until the user wants to quit the program, and loop through strings.

In the previous section whenever the system encountered an error it was handled by quitting the program. The system also could only handle one receipt at a time. If another receipt needed to be produced then the program had to be ran again. This had to be done because in the previous section only selection was used. With the introduction of loops the program can loop until valid inputs are received and loop until the user decides to quit the program, as well as many more things which will be added in this section and the sections that follow.

Make sure to download the provided program `main.cpp` to code along with this section.

3.1 Looping Until Valid Input

In the previous section the following bounds were applied on the inputs from the user:

Input	condition
`name`	`name` length < 1 **or** `name` length > 25
`cost`	input error **or** `cost` < 0 **or** `cost` ≥ 1000
`qty`	input error **or** `qty` < 1 **or** `qty` > 9
`tendered`	input error **or** `tendered` $<$ `total`
`cardNumber`	`cardNumber` length != 16

Table 6: Input bounds

Currently in `main()` when getting each of these inputs if an input is out of bounds an error is displayed and the program is exited. Now we will change these inputs to loop until a valid input is received. To do this the following steps should be performed for each input listed in table 5:

1. All of the logic for a specific input should be placed into a `do...while` loop. The reason why a `do...while` loop would be chosen is since each input has to be asked for at least 1 time we will be guaranteed to need to run the loop at least one time. We can then use the `while` statement of the `do...while` loop to tell if the loop needs to run again if the input was out of bounds.
2. Remove the `exit(0)` that happens if an input is out of bounds
3. Add a boolean called `flag` (you can create 1 variable and reuse it on each input).
4. Set `flag` equivalent to the logical expression that is used to check the bounds of the input.
5. Replace logical expression in the bounds check `if` statement with the new `flag` variable. Also use this `flag` variable as the loop control variable for the `while` statement of the `do...while` loop.
6. Clear and ignore any input in `cin` using `cin.clear()`[13] and `cin.ignore()`[14] .

Add this looping to the inputs in table 5.

See this video for help with `do...while` loops: `www.youtube.com/watch?v=fD2q_ko7AAo`
See this video for help with `input failure` looping: `www.youtube.com/watch?v=D8vPr9QLggs`

[13] `https://www.cplusplus.com/reference/ios/ios/clear/`
[14] `https://www.cplusplus.com/reference/istream/istream/ignore/`

```
Item Name: Example
Item Cost: -1
Error: Invalid cost.
Item Cost: 1000
Error: Invalid cost.
Item Cost: error
Error: Invalid cost.
Item Cost: 5.99
Quantity:  ...
```

Figure 15: Looping until a valid `cost` is input

3.2 Looping Cash/Card Selection

When the user enters in an invalid selection when selecting between cash and card the program should loop until a valid selection is made. This is done similar to how the looping was done in the previous subsection but with a few minor tweaks:

1. Place the logic for prompting/getting the cash/card selection and the switch statement inside of a loop (I will leave this one up to you to decide on which type you want to use
2. In the default case of the switch statement, update a variable of your choosing to alert your program that it needs to loop/prompt/read in again.
3. Loop until a valid input is received.

Add this looping to the cash/card selection/switch statement. An example of how this should look on one input follows. All inputs should loop in the same manner when invalid input is received, but with different prompts/error messages.

```
...
(C/c) - Cash
(D/d) - Card
Selection: error
Error: Invalid selection.
(C/c) - Cash
(D/d) - Card
Selection: err
Error: Invalid selection.
(C/c) - Cash
(D/d) - Card
Selection: c
...
```

Figure 16: Looping until a valid `selection` is input

3.3 Looping Until User Quits Program

The program currently only supports one receipt per run. If a second receipt is needed the program must be ran again. To fix this this the entire program needs to be wrapped in a loop. Again, I will leave the choice of what loop to use up to you. Once you have chosen what type of loop you want to use enclose everything in main after printing the header in a loop of your choice (There is a comment for `3.3 TODO`, your loop should start after this comment). Everything up to the final return 0 in main() should be enclosed in the loop (but not including the return 0).

In order to know when the looping stops a prompt/input for a selection must occur at the end of the loop after the switch statement but still inside of the loop. To do this the standard prompt/`cin` method should be used to ask the user to `Enter (C/c) to Continue`. If the user enters `'c'` or `'C'` then the program should just continue looping. Otherwise the program should break out of the loop. Doing it this way makes it so we don't have to check this selection for invalid inputs since any input that is not `'c'` or `'C'` will just quit the program. :)

Add this looping to your program. An example of how your program should look with this looping is on the following page.

See this video for help with `while` loops: `www.youtube.com/watch?v=6_NqsD5fSp4`

```
(Header truncated for space)
Item Name: Example
Item Cost: 5.99
Quantity:  1

Subtotal:  5.99
Tax:       0.50
Total:     6.49

(C/c) - Cash
(D/d) - Card
Selection: d

Card number: 4123123412341234

/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
|                                      |
|             UNLV CS Shop             |
|         4505 S Maryland Pkwy         |
|          Las Vegas,NV 89154          |
|            (702) 895-3011            |
|                                      |
| Example                     5.99 x 1 |
|                                      |
| Subtotal                        5.99 |
| Tax Rate                       8.25% |
| Tax                             0.50 |
| Total                           6.49 |
|                                      |
| Card Sale                            |
| Type                            Visa |
| Card                XXXXXXXXXXXX1234 |
|                                      |
\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/

Enter (C/c) to Continue: c
Item Name: Another Example
Item Cost: 59.19
Quantity:  2

...
```

Figure 17: Looping until the user doesn't want to continue.

3.4 Card Number Validation

Lastly the card number input is only being checked currently that it is 16 characters long. But the user could enter a character that is not a digit (`'0' - '9'`) which would not be valid for a card number. Credit cards must also start with certain numbers. These numbers identify what type of card the card number is. These types are:

First Digit	Card Type
3	Amex
4	Visa
5	Master
6	Discover

Table 7: Card types based on first digit

To check the card number entered follows these rules first check the character in the 0th position of the string is equivalent to one of the four starting digits contained within **Table 7**. If it is set a string variable equivalent to the proper Card Type from the table based on that first digit. If it is not simply output an error message and loop back to prompting/reading in the card number again.

The rest of the string (characters 1-15) must then be iterated through to check they are digits. Again, I will let you choose which type of loop to use to iterate through these characters. As the loop iterates through the indices 1-15 of the string it should check the character is a valid digit. To check if a character is a valid digit start with checking the ASCII[15] for the decimal value of the characters `'0' - '9'`, you will notice they run consecutively from 48-57. This means the program can just check for each character that:

$$48 \leq character \leq 57$$

In algorithm form, here is how you can validate the card number:

1. Check the 0th character is equivalent to one of the digits in **Table 7**. If it is set a string variable equivalent to the proper Card Type. If it is not error and start again from prompting and reading in the card number.
2. Start a loop of your choice to iterate from character 1-15 of the card number.
3. Check the current character the loop is on is a valid digit. If it is not output an error, quit looping, and start again from prompting and reading in the card number.
4. If the current character is a digit move onto the next character and start from (3).

Once the card number has been validated the program can then continue on to printing the card receipt with only making 1 change to the receipt, showing the type of card used on the receipt. To do this the type can be output on a new line above the line for the card number, using `<iomanip>` to format the row like is shown in **Figure 18**.

Add card validation to `main()` after the comment for `3.4 TODO`. An example of how the validation/errors look when running the program is shown on the next page.

See this video for help with `for` loops: `www.youtube.com/watch?v=1KS3M09TqtM`

[15] `www.asciitable.com`

```
...
(C/c) - Cash
(D/d) - Card
Selection: d

Card number:
Error: Invalid card number.
Card number: 4123123
Error: Invalid card number.
Card number: 412312341234123
Error: Invalid card number.
Card number: 44443333222211110
Error: Invalid card number.
Card number: 444433332222111100000000000
Error: Invalid card number.
Card number: 41231error341234
Error: Invalid card number.
Card number: 1111222233334444
Error: Invalid card number.
Card number: 4444333322221111

/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
|                                                  |
|              UNLV CS Shop                        |
|          4505 S Maryland Pkwy                    |
|           Las Vegas,NV 89154                     |
|             (702) 895-3011                       |
|                                                  |
| Example                          5.99 x 1        |
|                                                  |
| Subtotal                                 5.99    |
| Tax Rate                                8.25%    |
| Tax                                      0.50    |
| Total                                    6.49    |
|                                                  |
| Card Sale                                        |
| Type                                     Visa    |
| Card                      XXXXXXXXXXXX1111       |
|                                                  |
\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/

...
```

Figure 18: Looping until valid card/card receipt with type

3.5 Example Output

Below are sample executions of the complete program. Text in red is input from the user, this coloring is simply here to distinguish components and not needed in your program.

Example **1**

```
Alexs-iMac desktop % g++ AS2.cpp
Alexs-iMac desktop % ./a.out
+--------------------------------------------------------------------------------+
|    UU     UU  NNNN     NN  LL        VV       VV         CCCCCC    SSSSSSSS   |
|   /UU    /UU /NN/NN   /NN /LL       /VV     /VV        CC////CC  SS//////    |
|   /UU    /UU /NN//NN  /NN /LL       /VV     /VV       CC    //  /SS          |
|   /UU    /UU /NN //NN /NN /LL       //VV    VV       /CC        /SSSSSSSSS   |
|   /UU    /UU /NN  //NN/NN /LL        //VV  VV        /CC        ////////SS   |
|   /UU    /UU /NN   //NNNN /LL         //VVVV         //CC    CC         /SS   |
|   //UUUUUUU  /NN    //NNN /LLLLLLLL   //VV           //CCCCCC   SSSSSSSS    |
|    ///////   //      ///  ////////     //             //////   ////////     |
|                                                                                |
|                   SSSSSSSS  HH      HH    OOOOOOO    PPPPPPP                 |
|                  SS//////  /HH     /HH   OO/////OO  /PP////PP                |
|                 /SS        /HH     /HH  OO     //OO /PP   /PP                |
|                 /SSSSSSSSS /HHHHHHHHHH /OO       /OO /PPPPPPP                |
|                 ////////SS /HH//////HH /OO       /OO /PP////                 |
|                        /SS /HH     /HH //OO      OO  /PP                      |
|                  SSSSSSSS  /HH     /HH  //OOOOOOO   /PP                      |
|                 ////////   //      //    ///////    //                       |
+--------------------------------------------------------------------------------+
Item Name: Example
Item Cost: 5.99
Quantity:  1

Subtotal:  5.99
Tax:       0.50
Total:     6.49

(C/c) - Cash
(D/d) - Card
Selection: d

Card number: 5555123400004321

/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
|                                                                              |
|                              UNLV CS Shop                                    |
|                          4505 S Maryland Pkwy                                |
|                           Las Vegas,NV 89154                                 |
|                             (702) 895-3011                                   |
|                                                                              |
| Example                                                  5.99 x 1            |
|                                                                              |
| Subtotal                                                            5.99     |
| Tax Rate                                                           8.25%     |
| Tax                                                                 0.50     |
| Total                                                               6.49     |
|                                                                              |
| Card Sale                                                                    |
| Type                                                             Master      |
| Card                                                 XXXXXXXXXXXX4321        |
|                                                                              |
\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/

Enter (C/c) to Continue: e
```

Example **2**

```
Alexs-iMac desktop % ./a.out
(header truncated for space)

Item Name:
Error: Invalid name.
Item Name: this name is too long for the program to handle
Error: Invalid name.
Item Name: Example
Item Cost: -100
Error: Invalid cost.
Item Cost: 1000
Error: Invalid cost.
Item Cost: error
Error: Invalid cost.
Item Cost: 5.99
Quantity:  0
Error: Invalid quantity.
Quantity:  10
Error: Invalid quantity.
Quantity:  error
Error: Invalid quantity.
Quantity:  1

Subtotal:  5.99
Tax:       0.50
Total:     6.49

(C/c) - Cash
(D/d) - Card
Selection: c

Amount tendered from customer: 5
Error: Invalid amount tendered.
Amount tendered from customer: error
Error: Invalid amount tendered.
Amount tendered from customer: 6.48
Error: Invalid amount tendered.
Amount tendered from customer: 10

/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
|                                                  |
|              UNLV CS Shop                        |
|          4505 S Maryland Pkwy                    |
|           Las Vegas,NV 89154                     |
|             (702) 895-3011                       |
|                                                  |
| Example                        5.99 x 1          |
|                                                  |
| Subtotal                                  5.99   |
| Tax Rate                                 8.25%   |
| Tax                                       0.50   |
| Total                                     6.49   |
|                                                  |
| Tendered                                 10.00   |
| Change                                    3.51   |
|                                                  |
\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/

Enter (C/c) to Continue: q
```

Example **3**

```
Alexs-iMac desktop % ./a.out
(header truncated for space)

Item Name: Laptop
Item Cost: 799.99
Quantity:  2

Subtotal:  1599.98
Tax:       132.00
Total:     1731.98

(C/c) - Cash
(D/d) - Card
Selection: e
Error: Invalid selection.

(C/c) - Cash
(D/d) - Card
Selection: invalid
Error: Invalid selection.

(C/c) - Cash
(D/d) - Card
Selection: err
Error: Invalid selection.

(C/c) - Cash
(D/d) - Card
Selection: D

Card number: 4312655
Error: Invalid card number.
Card number: 4657498412168574659874362
Error: Invalid card number.
Card number: 1111222233334444
Error: Invalid card number.
Card number: 2222111133334444
Error: Invalid card number.
Card number: 4111error3334444
Error: Invalid card number.
Card number: 4894895438799854

/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
|                                      |
|             UNLV CS Shop             |
|         4505 S Maryland Pkwy         |
|          Las Vegas,NV 89154          |
|            (702) 895-3011            |
|                                      |
| Laptop                    799.99 x 2 |
|                                      |
| Subtotal                     1599.98 |
| Tax Rate                       8.25% |
| Tax                           132.00 |
| Total                        1731.98 |
|                                      |
| Card Sale                            |
| Type                            Visa |
| Card                XXXXXXXXXXXX9854 |
|                                      |
\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/

Enter (C/c) to Continue: n
```

Example **4**

```
Alexs-iMac desktop % ./a.out
(header truncated for space)

Item Name: Desktop
Item Cost: 949.99
Quantity:  1

Subtotal:  949.99
Tax:       78.38
Total:     1028.37

(C/c) - Cash
(D/d) - Card
Selection: C

Amount tendered from customer: 1100

/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
|                                                                             |
|                              UNLV CS Shop                                   |
|                          4505 S Maryland Pkwy                               |
|                           Las Vegas,NV 89154                                |
|                             (702) 895-3011                                  |
|                                                                             |
| Desktop                                    949.99 x 1                       |
|                                                                             |
| Subtotal                                                        949.99      |
| Tax Rate                                                          8.25%     |
| Tax                                                              78.38      |
| Total                                                          1028.37      |
|                                                                             |
| Tendered                                                       1100.00      |
| Change                                                           71.63      |
|                                                                             |
\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\

Enter (C/c) to Continue: C

Item Name: Keyboard
Item Cost: 15.95
Quantity:  2

Subtotal:  31.90
Tax:       2.64
Total:     34.54

(C/c) - Cash
(D/d) - Card
Selection: D

Card number: 6000000012345678
```

(Example continued on the next page)

Example **4** Continued

```
/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
|                                        |
|              UNLV CS Shop              |
|          4505 S Maryland Pkwy          |
|           Las Vegas,NV 89154           |
|             (702) 895-3011             |
|                                        |
| Keyboard                   15.95 x 2 |
|                                        |
| Subtotal                       31.90 |
| Tax Rate                       8.25% |
| Tax                             2.64 |
| Total                          34.54 |
|                                        |
| Card Sale                              |
| Type                        Discover |
| Card                XXXXXXXXXXXX5678 |
|                                        |
\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/

Enter (C/c) to Continue: c

Item Name: Chips
Item Cost: 1.49
Quantity:  9

Subtotal:  13.41
Tax:       1.11
Total:     14.52

(C/c) - Cash
(D/d) - Card
Selection: c

Amount tendered from customer: 15

/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
|                                        |
|              UNLV CS Shop              |
|          4505 S Maryland Pkwy          |
|           Las Vegas,NV 89154           |
|             (702) 895-3011             |
|                                        |
| Chips                       1.49 x 9 |
|                                        |
| Subtotal                       13.41 |
| Tax Rate                       8.25% |
| Tax                             1.11 |
| Total                          14.52 |
|                                        |
| Tendered                       15.00 |
| Change                          0.48 |
|                                        |
\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/

Enter (C/c) to Continue: n
```

3.6 Submission

The first thing you need to do before submitting this assignment is read and understand the **Academic Integrity Policy** at the beginning of this handout. By submitting this assignment you are accepting this policy even if you did not sign/submit the required **Academic Integrity Policy Acknowledgement**.

Make sure your code is saved as **main.cpp**. Do not ignore this step or save your file with different names.

Your program source code must be submitted via **CodeGrade** as a properly named **.cpp** file. **Late submissions will not be accepted**.

4 Arrays

In this section you will learn, in C++, how to use arrays, pass and use command line arguments, use the <fstream> library, read from files into arrays, .

There is one large feature our system currently doesn't have that you may have noticed, each order can only contain 1 item. This is extremely impractical. The reason why we have had to work in this impractical manner is because in the previous sections we did not have arrays. This would make it next to impossible for us to declare the proper amount of variables to work with since each transaction's size varies. With arrays though, as you will see in this section, this problem is easily solved.

Make sure to download the provided program main.cpp to code along with this section.

4.1 Command Line Arguments and Opening Files

With multiple orders that contain multiple items reading in from the command line would be extremely slow and inefficient. To fix this the data for all the transactions for a single day are read in from a file.

To read in from a file the name of the file must first be known. To pass the name of the file we will use command line arguments. When a C++ program is ran 2 arguments (variables) are passed to main which we can use, these are:

Argument	Description
argc	How many arguments were passed
argv	An array of the arguments passed

Table 8: Command line arguments passed when running a C++ program

Up to this point, every time the program has been ran it has been with the command ./a.out. This passes 1 command line argument to the program in argv, that being ./a.out, thus argv = [./a.out] and argc = 1 (Note: There is only one argument passed but it is in the 0th index of argv). If the program were to be ran with ./a.out fileName.txt then argv = [./a.out, fileName.txt] and argc = 2. We could then save that file name into a string, we will call it fname, by accessing the value in the 1st index of argv which we know will contain the file name. This file name can then be used to open a file.

To open a file that will be read from, the <fstream> library must first be included at the top of the program. A ifstream object, we will call it iFile, can then be added to the program which will hold the file once it has been opened. The fname can then be passed to iFile.open()[16] to be opened.

The file then needs to be checked that it was opened successfully by checking iFile.is_open(). If the file is open iFile.is_open()[17] will return true, otherwise it will return false. If the file is not open the program should output an error and terminate since the only way to get new command line arguments is to re-run the program, otherwise the program should continue on. Examples of how running the program with command line arguments should look are on the following page.

[16] https://www.cplusplus.com/reference/fstream/fstream/open/
[17] https://www.cplusplus.com/reference/fstream/ifstream/is_open/

```
Invalid file name:
Alexs-iMac desktop % ./a.out error.txt
Error: Invalid file name.
Alexs-iMac desktop %

- or -

Valid file name:
Alexs-iMac desktop % ./a.out a4in.txt
(Program header truncated for space)
```

Figure 19: Using command line arguments

Lastly, if there is no file name passed to the program when it is run a segmentation fault error will occur, this must be fixed. Since it is known that the program should always be run with 2 command line arguments we can just check if `argc` is equal to 2 and if not a usage message can be output and the program can be terminated.

```
Alexs-iMac desktop % ./a.out
Usage: ./a.out <file name>
Alexs-iMac desktop %

- or -

Alexs-iMac desktop % ./a.out a4in.txt too many args
Usage: ./a.out <file name>
Alexs-iMac desktop %
```

Figure 20: Invalid number of command line arguments

Add passing a file name using command line arguments to `main.cpp`.

4.2 Reading From a File

Now that the file has been opened successfully it can be read from. The file type the data is contained within is a `.csv` which stands for comma separated values. The data in these files can be thought of as a table. Each row is a separate entry in the table. Each row has a defined number of columns. Those columns are separated in the file, for each row, by commas. For example if you wanted to keep track of people's names and ages each row would be a separate person. Each row would have 2 columns of data, the first being the name and the second being the age. The name and age would be separated by a comma on each row. For this assignment each row represents a product sold and will have 6 pieces of data:

Column #	Description
1	Transaction ID
2	Name of product
3	Cost of product
4	Quantity of product sold
5	Payment type used for the transaction
6	Payment (dollar amount if cash, card number if card)

Table 9: Columns of data in file

Each row of data represents 1 item sold on a transaction. Multiple rows make up multiple items sold in one transaction. The rows are connected together by a transaction id in the first column. This id will be used to group together items as transactions in later subsections of this section. You can assume all data in the file is valid (thus no error checking needs to be done on the data being read in). You do however need to make sure that you stop reading properly once the end of file is hit, otherwise you may accidentally have an extra row of failed input in the end of your parallel arrays.

For the next part you will need to know how to get pieces of data from strings containing comma separated values and store them in parallel arrays. Take the string: string str = "the start,1,the end"; First find the comma location of the first comma (located between "start" and "1").

```
int commaLoc = str.find(',');
```
[18]

This will set `commaLoc == 9`. Then take a sub-string of `str` from the beginning to the comma location.

```
str data = str.substr(0, commaLoc);
```
[19]

This will set `data == "the start"`. Then `"the start"` is still contained in `str` it needs to be removed.

```
str = str.substr(commaLoc + 1, str.length());
```

This will set `str == "1,the end"`. Now `data` contains the single piece of data (as a string) and `str` contains the rest of the string. `data` can now be either stored in a string parallel array or, if needed, be typecast. In this case `data` would just be stored as a string in a parallel array. The rest of the row can then be broken down using the same `find()` and `substr()` method from above. If we do this one more time we will be left with `data == "1"` and `str == "the end"`. `data` is currently a string but it contains data that should be an integer. This needs to be typecast.

```
int i\_data = stoi(data)
```
[20]

This will set `i\_data == 1` as an integer which can then be stored in a int parallel array. Also, `str` only contains the last piece of data in the row now, `str == "the end"`. Since it is the only piece of data in the row now with no comma this piece of data can also be stored in a parallel array without using the `find()` and `substr()` method.

Note: Only `stoi` was demonstrated here but any typecasting can be used on the data held in `data`.

The first thing that must be done to read the data from secondary storage into arrays in the program is to add an end of file controlled while loop (which has aleady been provided to you in `main()`). A variable `itemCnt` has been provided in `main()` which can be used to keep track of the current position in the parallel arrays while reading in. Inside of this loop the file will be read from. The loop should perform the following steps:

1. Use `getline()` to get a row of data from the file.
2. Use the `find()` and `substr()` method to get the first piece of data from the row, and save it into the `itemCnt` in the `ids` array. Use `stoi()` as shown in the example to convert the piece of data before storing it in the array.
3. Use the `find()` and `substr()` method to get the next piece of data from the row, and save it into the `itemCnt` in the `names` array. The piece of data does not need to be converted and can be left as a string.
4. Use the `find()` and `substr()` method to get the first piece of data from the row, and save it into the `itemCnt` in the `costs` array. Use `stoi()` as shown in the example to convert the piece of data before storing it in the array. Instead of using `stoi()` to cast the piece of data to an integer use `stod()`[21] to cast the data to a double.

[18] https://www.cplusplus.com/reference/string/string/find/
[19] https://www.cplusplus.com/reference/string/string/substr/
[20] https://www.cplusplus.com/reference/string/stoi/
[21] https://www.cplusplus.com/reference/string/stod/

5. Use the `find()` and `substr()` method to get the first piece of data from the row, and save it into the `itemCnt` in the `qtys` array. Use `stoi()` as shown in the example to convert the piece of data before storing it in the array.

6. Use the `find()` and `substr()` method to get the next piece of data from the row, and save it into the `itemCnt` in the `paymentTypes` array. The piece of data does not need to be converted and can be left as a string.

7. The only piece of data left in the row is the final piece of data, simply save it in the `payments` array keeping it a string.

8. Increment the `itemCnt` by 1 and loop back to (1) until all the file's data has been read.

Note: All of the variables and arrays you need for this part are already defined in `main()`.

Now that the file has been read in from it should closed using `iFile.close()`[22] and the amount of rows read in should be displayed. An example of how this should look is shown on the next page.

```
Alexs-iMac desktop % ./a.out a4in.txt
Number of rows read in: 8

(Program header truncated for space)
```

Figure 21: Number of rows read in reading from a4in.txt

4.3 Printing Multi-Item Receipts

Now that you have all the data read into 6 parallel arrays a receipt can be printed for each transaction. To do this you will first need to find the start/end of where the current transaction to be printed is in the 6 arrays. Once you know where the start and end are you can then calculate the subtotal, tax, and total of the transaction. These three numbers can the be saved in the `subtotals`, `taxes`, and `totals` arrays respectively. A cash or card receipt can then be printed with all of the items on the list. An algorithm to do this follows.

1. Find the `start` of the transaction in the arrays by setting it equivalent to the previous transaction's `end`.

2. Set `currentId` (already defined in `main()` for you) equivalent to the id in `ids` at `start`. This way you know the id of the transaction you are currently on.

3. Run a while loop that checks if `ids` at `end` is equivalent to the `currentId`. If it is then increment `end` and continue looping until you find a new id. Once you find a new id you know you are at the end of the current transaction in the arrays.

4. Using the `start` and `end` points, iterate through the `costs` and `qtys` arrays, multiplying each cost by each quantity and summing the product for each item purchased on the transaction. This will give you a `subtotal` of all the items on the transaction.

5. Using the `subtotal` calculated in the previous step calculate the `tax` using the same formula as we have been using in the previous sections of this assignment. As a reminder the formula is:

$$tax = ceil(subtotal * TAX\_RATE * 100.0)/100.0$$

[22] `https://www.cplusplus.com/reference/fstream/ofstream/close/`

6. Calculate the total cost by adding the `subtotal` and `tax` together.

7. Increment the `transactionCnt`.

8. The receipt can then be printed. The receipt should be printed to it's own file. Each receipt file should be named: `"receipt-CURRENTID.txt"` (where `CURRENTID` is the id stored in the `currentId` variable). The receipt must be modified slightly from previous sections. Instead of printing only 1 item on each receipt you will have to loop from `start` to `end` in the `names`, `costs`, and `qtys` arrays outputting each item purchased on a new line of the receipt. Then if it is a cash transaction you need to use `stod()` on the tendered amount stored in `payments` at `start` and then subtract the total amount of the transaction from the tendered amount. This will give you the change. The tendered amount and change can then be output. If it is a card transaction the `cardType` can then be determined based on the first character of the card number stored in `payments` at `start`. The card types based on the first digit of a card can be found on Table 7 on page 30. The last 4 digits of the card and the card type can then be output.
   Note: You can assume the tendered amount and card number will be valid for this section of the assignment. They do not need to be checked for errors.

9. Continue from (1) until every transaction in the parallel arrays has a receipt printed.

Figures of how the receipts with multiple items should look follow. These receipts will be printed to their own `.txt` files.

```
/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
|                                                                              |
|                                 UNLV CS Shop                                 |
|                            4505 S Maryland Pkwy                              |
|                             Las Vegas,NV 89154                               |
|                               (702) 895-3011                                 |
|                                                                              |
| Laptop                                                        799.99 x 1     |
| Laptop Bag                                                     19.99 x 1     |
| Notebook                                                        1.49 x 3     |
|                                                                              |
| Subtotal                                                                824.45 |
| Tax Rate                                                                 8.25% |
| Tax                                                                     68.02 |
| Total                                                                  892.47 |
|                                                                              |
| Tendered                                                               1100.00 |
| Change                                                                  207.53 |
|                                                                              |
\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
```

Figure 22: Cash receipt with multiple items

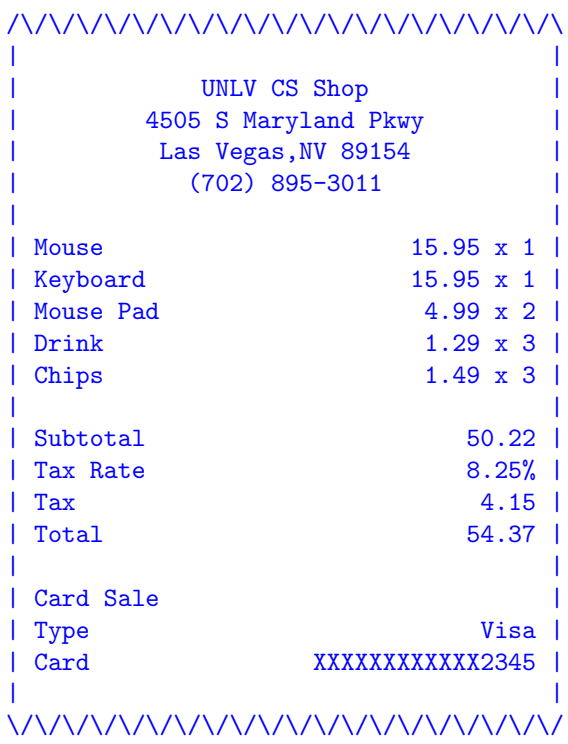


Figure 23: Card receipt with multiple items

4.4 Printing Sales Summary

A summary of every transaction can now be printed if the user want one. The summary will include things like the count of transactions, the total money brought in, max/min transaction amounts, etc. This summary will be printed to a file called `"summary.txt"`. This file will always be named the same so it is okay in this instance to statically open the file name into a `ofstream` object.

First we will check if the user wants to print the summary. The user can be prompted to enter `'y'` or `'Y'` and if they do `"Printing sales report to summary.txt..."` can be output and the summary printed. If anything else is entered `"Exiting program without sales report..."` should be output, the program should exit, and the summary should not be printed.

```
Print sales summary? (Y/y) for yes, any other key for no: y
```

Figure 24: Prompting for summary

Some statistics of the sales need to be calculated if the summary is to be printed: the sum of subtotals, the sum of taxs, the sum of totals, the minimum total transaction index, and the maximum total transaction index. Every subtotal, tax charge, and total was saved into their respective arrays previously. We also know the amount of transactions in `transactionCnt`. We can loop from index 0 to index `transactionCnt` keeping a running total of the subtotals, tax charges, and totals by summing each index as we loop. As we are looping we can also check if each index is the new min or max. To do this check, if the value in `totals` at the current index is less than the value at the `min` index in `totals` or if it is greater than the value at the `max` index in `totals`. If it is the index that is being checked can be stored in the `min` or `max` variable.

Note: You will need to add variables to `main()` to keep track of the sums, min index, and max index.

The summary can then be printed. The summary is printed in 2 columns. The first column (label) should be left justified with a width of 40 and the second column (statistic) should be right justified with a width of 20. The empty spaces of `setw()` buffers should be filled with a `'.'` (period). The following statistics should then be output to the file `"summary.txt"`:

Column #	Description
Total Transactions	value in `transactionCnt`
Total Pre-tax Sales	sum of subtotals
Total Taxes Charged	sum of tax charges
Total Post-tax Sales	sum of totals
Average Pre-tax Sale	sum of subtotals / `transactionCnt`
Average Post-tax Sale	sum of totals / `transactionCnt`
Lowest Sale ID	minimum index
Lowest Sale Subtotal	value in `subtotals` at minimum index
Lowest Sale Tax	value in `taxes` at minimum index
Lowest Sale Total	value in `totals` at minimum index
Highest Sale ID	maximum index
Highest Sale Subtotal	value in `subtotals` at maximum index
Highest Sale Tax	value in `taxes` at maximum index
Highest Sale Total	value in `totals` at maximum index

Table 10: Statistics to include in summary

A figure of how the summary should look follows (with relevant information based on whatever data file is passed to it). This summary will be printed to `summary.txt` any time the user wants it printed.

```
+------------------------------------------------------------+
| Total Transactions.....................................2 |
| Total Number of Items Sold.............................8 |
|                                                            |
| Total Pre-tax Sales...............................874.67 |
| Total Taxes Charged...............................72.17 |
| Total Post-tax Sales..............................946.84 |
|                                                            |
| Average Pre-tax Sale..............................437.34 |
| Average Post-tax Sale.............................473.42 |
|                                                            |
| Lowest Sale ID.........................................1 |
| Lowest Sale Subtotal..............................50.22 |
| Lowest Sale Tax...................................4.15 |
| Lowest Sale Total.................................54.37 |
|                                                            |
| Highest Sale ID........................................0 |
| Highest Sale Subtotal............................824.45 |
| Highest Sale Tax..................................68.02 |
| Highest Sale Total...............................892.47 |
+------------------------------------------------------------+
```

Figure 25: Summary of all transactions

4.5 Example Output

Below are sample executions of the complete program. Text in red is input from the user, this coloring is simply here to distinguish components and not needed in your program.

Example **1**

```
Alexs -iMac desktop % g++ AS4.cpp
Alexs -iMac desktop % ./a.out
Usage: ./a.out <file name>
```

Example **2**

```
Alexs -iMac desktop % ./a.out too many arguments
Usage: ./a.out <file name>
```

Example **3**

```
Alexs -iMac desktop % ./a.out a4-1.txt
Number of rows read in: 8

+--------------------------------------------------------------------------------+
|    UU      UU  NNNN      NN  LL        VV       VV          CCCCCC    SSSSSSSS   |
|   /UU     /UU /NN/NN    /NN /LL       /VV      /VV         CC////CC  SS//////    |
|   /UU     /UU /NN//NN   /NN /LL       /VV      /VV        CC    //  /SS          |
|   /UU     /UU /NN //NN /NN /LL       //VV     VV        /CC         /SSSSSSSSS   |
|   /UU     /UU /NN  //NN/NN /LL        //VV  VV         /CC         ////////SS   |
|   /UU     /UU /NN   //NNNN /LL         //VVVV          //CC    CC         /SS   |
|   //UUUUUUU  /NN    //NNN /LLLLLLLL    //VV            //CCCCCC   SSSSSSSS    |
|    ///////   //      ///  ////////      //              //////   ////////     |
|                                                                                |
|                   SSSSSSSS  HH      HH    OOOOOOO    PPPPPPP                  |
|                  SS//////  /HH     /HH   OO/////OO  /PP////PP                 |
|                 /SS        /HH     /HH  OO     //OO /PP   /PP                 |
|                 /SSSSSSSSS /HHHHHHHHHH /OO       /OO /PPPPPPP                 |
|                 ////////SS /HH//////HH /OO       /OO /PP////                  |
|                        /SS /HH     /HH //OO     OO  /PP                       |
|                  SSSSSSSS  /HH     /HH  //OOOOOOO   /PP                       |
|                 ////////   //      //    ///////    //                        |
+--------------------------------------------------------------------------------+
Printing Customer #1's receipt.
Printing Customer #2's receipt.
Print sales summary? (Y/y) for yes, any other key for no: y

Printing sales report to summary.txt...
Alexs -iMac desktop % cat receipt-1.txt
/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
|                                              |
|              UNLV CS Shop                    |
|          4505 S Maryland Pkwy                |
|           Las Vegas,NV 89154                 |
|             (702) 895-3011                   |
|                                              |
| Laptop                       799.99 x 1      |
| Laptop Bag                    19.99 x 1      |
| Notebook                       1.49 x 3      |
|                                              |
| Subtotal                          824.45     |
| Tax Rate                           8.25%     |
| Tax                               68.02      |
| Total                             892.47     |
|                                              |
| Tendered                         1100.00     |
| Change                            207.53     |
|                                              |
\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
```

Example **3** Continued

```
Alexs-iMac desktop % cat receipt-2.txt
/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
|                                      |
|             UNLV CS Shop             |
|         4505 S Maryland Pkwy         |
|          Las Vegas,NV 89154          |
|            (702) 895-3011            |
|                                      |
| Mouse                      15.95 x 1 |
| Keyboard                   15.95 x 1 |
| Mouse Pad                   4.99 x 2 |
| Drink                       1.29 x 3 |
| Chips                       1.49 x 3 |
|                                      |
| Subtotal                       50.22 |
| Tax Rate                       8.25% |
| Tax                             4.15 |
| Total                          54.37 |
|                                      |
| Card Sale                            |
| Type                            Visa |
| Card                XXXXXXXXXXXX2345 |
|                                      |
\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
Alexs-iMac desktop % cat summary.txt
+--------------------------------------------------------------+
| Total Transactions.........................................2 |
|                                                              |
| Total Pre-tax Sales...................................874.67 |
| Total Taxes Charged....................................72.17 |
| Total Post-tax Sales..................................946.84 |
|                                                              |
| Average Pre-tax Sale.................................437.335 |
| Average Post-tax Sale.................................473.42 |
|                                                              |
| Lowest Sale ID.............................................1 |
| Lowest Sale Subtotal...................................50.22 |
| Lowest Sale Tax.........................................4.15 |
| Lowest Sale Total......................................54.37 |
|                                                              |
| Highest Sale ID............................................0 |
| Highest Sale Subtotal.................................824.45 |
| Highest Sale Tax.......................................68.02 |
| Highest Sale Total....................................892.47 |
+--------------------------------------------------------------+
```

4.6 Submission

The first thing you need to do before submitting this assignment is read and understand the **Academic Integrity Policy** at the beginning of this handout. By submitting this assignment you are accepting this policy even if you did not sign/submit the required **Academic Integrity Policy Acknowledgement**.

Make sure your code is saved as **main.cpp**. Do not ignore this step or save your file with different names.

Your program source code must be submitted via **CodeGrade** as a properly named **.cpp** file. **Late submissions will not be accepted**.

Note: When submitting to CodeGrade the output tests will run your program, then run a command to show all of the receipt/summary files. This means the receipts/summary you see output on CodeGrade are being output from the receipt files and summary file **not** from your program.

5 Functions

In this section you will learn, in C++, how to call/define void and value returning functions, pass parameters via value and reference, and overload functions.

Up until now we have repeated a lot of code and have only coded in main. This is unpractical though. In order to break our large problems down into many smaller sub-problems we must use functions. This allows us to break up our thinking to accomplish small tasks that accomplish a larger task in the end. In this section we will expand on our cash register system by allowing transactions to be added.

Make sure to download the provided program `main.cpp` and `provided_functions.cpp` to code along with this section. In `provided_functions.cpp` you will find many functions you will use in this section. Do not modify this file at all or your submission will not work in CodeGrade.

5.1 Defining Function Behavior

Like in the previous section, when the program is run it will be passed a file name. This file name will be in `argv` and the data contained within it will contain all the items that have previously been been purchased. When the program is run this file name is always expected and it is the only thing that is expected thus `argc` should always be equivalent to 2. In the provided `main()` you will see a call to a function:

```
void openFile(ifstream& iFile, const int argc, char const *argv[])
```

This function takes in a input file to read into (`iFile`), an integer signifying the count of command line arguments (`argc`), and a array of command line arguments (`argv`). In `main()` you will see a call to this function passing 3 variables of the same names. After `main()` you will see a stub for the function `openFile()` but there is no code in the body of the function. You need to write this code. In the body of this function you will want to:

1. Check if `argc` is equal to 2. If it is not output a usage message `"Usage: ./a.out <file name>"` and quit the program using `exit(0)`.
2. Open the file name contained in `argv` at index 1 into `iFile`. If the file does not open output an error message `"Error: Invalid file name"` and quit the program using `exit(0)`.

Once this behavior has been defined in the body of the `openFile()` function, when the program is run it will open the file name that is passed into the `iFile` variable in main. The file will be passed back to main after the call to `openFile()` since `iFile` is passed by reference in the call to `openFile()`.

5.2 Calling functions

In `provided_functions.cpp` there are 4 functions that will need to be called in this subsection:

1. ```
   void readPreviousTransactions(ifstream &iFile, int &itemCnt, int ids[],
                                 string names[], double costs[], int qtys[],
                                 string paymentTypes[], string payments[])
   ```

   This function reads in the rows of data from `iFile` and stores them in the parallel arrays `ids`, `names`, `costs`, `qtys`, `paymentTypes`, and `payments`. As the rows are read in `itemCnt` will be updated to reflect the amount of data in each of the parallel arrays.

2. ```
   void readInventory(string names[], double costs[])
   ```

   This function reads in the names of the items that are available for purchase into `names` and their costs into `costs`. Since the shop always sells the same items this function always reads from the same file `inventory.csv`.

3. `void calculateTotals(int itemCnt, int& transactionCnt, int ids[], double costs[], int qtys[], double subtotals[], double taxes[], double totals[])`

   This function will calculate the subtotals, taxes, and totals for each transaction. As each subtotal, tax, and total is calculated it will be stored in it's respective parallel array (`subtotals`, `taxes`, or `totals`).

4. `void printHeader()`

   This function prints The UNLV CS Shop header to the screen.

First, call `readPreviousTransactions()` passing it `iFile`, `itemCnt`, `ids`, `names`, `costs`, `qtys`, `paymentTypes`, and `payments` in that order. This call to `readPreviousTransactions()` will update the 6 parallel arrays to contain the data in `iFile` and `itemCnt` to being the number of rows of data read in.

Next, call `readInventory()` passing it `invNames` and `invCosts` in that order. This call to `readInventory()` will update the 2 parallel arrays to contain the 27 items in inventory. These 2 arrays should not be changed after this call.

Next, call `calculateTotals()` passing it `itemCnt`, `transactionCnt`, `ids`, `costs`, `qtys`, `subtotals`, `taxes`, and `totals` in that order. This call to `calculateTotals()` will update `subtotals`, `taxes`, and `totals` to contain the subtotal, tax, and total for each transaction and `transactionCnt` to being the number of transactions total.

Lastly, call `calculateTotals()` which is a void function that takes in no parameters. This function will print The UNLV CS Shop header to the screen.

5.3 Writing a Function

Now that all the needed data has been read in the user can start using the cash register. The first thing that should happen is the user should be prompted to select between printing a summary, adding a transaction, or exiting the program. To do this you will need to define then call a function:

```
char getMode()
```

Which will:

1. Prompt the user to select an option. The prompt should look as follows:
   ```
   OPTIONS
   S/s - Print Summary
   T/t - Add Transaction
   E/e - Exit
   Selection:
   ```

2. Read in the `selection` from the user into a character. If input failure occurs or the user enters anything other than `'S'`, `'s'`, `'T'`, `'t'`, `'E'`, or `'e'` an error message `"Error: Invalid selection"` should be displayed and the failed input should be cleared/ignored. Once the failed input is cleared/ignored you should continue from (1). If there is no failure then the function should continue on.

3. Once a valid selection has been made, if the selection is `'E'` or `'e'` then a message `"Exiting..."` should be output and the program terminated with `exit(0)`. If the selection is `'S'` or `'s'` then `'s'` should be returned. If the selection is `'T'` or `'t'` then `'t'` should be returned.

Now that `getMode()` has been defined it can be called from main and the value that is returned can be saved in the variable `registerMode` (which has already been declared in `main()` for you). There is already an `if...else` statement in main that will calculate and print the summary if the user enters `'S'` or `'s'`. In the next subsection you will add what happens if the user enters `'T'` or `'t'`.

5.4 Adding Transactions

In the previous subsection the user selected the mode they wanted (print a summary or add a transaction). If the user selected `'S'` or `'s'` then a summary is calculated and printed (this is already done for you in `main()`). Though, if the user selected `'T'` or `'t'` then they want to add another transaction. To add a transaction a function must be added:

```
void addTransaction(int& itemCnt, int& transactionCnt, int ids[], string names[],
                    int qtys[],double costs[], string paymentTypes[],
                    string payments[],double subtotals[], double taxes[],
                    double totals[], string invNames[], double invCosts[]).
```

This function will get the items the user wants, calculate the subtotal/tax/total, get a cash or card payment, then print the receipt for the transaction to a file.

5.4.1 Getting Items

To get the items the user wants you will want to start a infinite while loop (`while(true){...}`). To exit this loop we will break from it later. You will need to get the item the user wants. A function has been provided to you:

```
int getSelectedItem(string invNames[], double invCosts[])
```

This function will display a menu to the user, then get the selected item from the user. It will do all the error checking needed on the selection and once a valid selection is entered it will be returned as a integer. Call `getSelectedItem` and pass it `invNames` and `invCosts` in that order. Then save the selected item to an integer variable (called `item` in this handout). If `item` is -1 then this means the user is done selecting items. In this case break out of the loop.

The quantity of items is now needed. You need to declare/define a function:

```
int getQuantity()
```

This function will:

1. Prompt the user to enter the `quantity` using the prompt `"Quantity: "`.
2. Read in the `quantity` the user wants.
3. If input error occurs or the `quantity < 1` or the `quantity > 9` then an error message `"Error: Quantity must be in range: 0 < qty < 10"` should be output and the failed input should be cleared/ignored. The program should then continue from (1) again. If no error occurs then the quantity should be returned.

You can now call this function from `addTransaction` inside the infinite while loop and save the returned quantity into an integer variable (called `qty` in this handout).

Now you can save the transaction id, item name, item cost, and quantity into their respective parallel arrays. You will need to store:

1. The transaction id the item being purchased belongs to. This transaction id is the current `transactionCnt + 1`. This will be stored at the current `itemCnt` index in the `ids` array.
2. The name of the item being purchased. The integer returned from your call to `getSelectedItem()` stored in `item` can be used as the index into `invNames` to get the name of the item. This item name can then be stored at the current `itemCnt` index in the `names` array.
3. The cost of the item being purchased. Again, the integer returned from your call to `getSelectedItem()` stored in `item` can be used as the index into `invCosts` to get the cost of the item. This item cost can then be stored at the current `itemCnt` index in the `costs` array.
4. The quantity wanted of the item. The integer returned from your call to `getQuantity()` stored in `qty` can just be stored at the current `itemCnt` index in the `qtys` array.
5. The cost for the `qty` of the item being purchased (`qty * cost`). This cost should be summed together with the costs of the other items being purchased on the transaction at the current `transactionCnt` index in the `subtotals` array.

The `itemCnt` can then be incremented by 1 and you can continue on to another iteration of your infinitely looping while loop. Once the user selects -1 this loop will stop executing/saving items.

5.4.2 Calculating Subtotal/Tax/Total

Once the user has enterered -1 in the previous step you know that the user no longer wants to add items to the transaction. The taxes and total total can now be calculated and saved to their respective arrays. You summed the costs of the items multiplied by their quantities already, this gives you the subtotal for the transaction. This subtotal can then be passed to the tax equation.

$$tax = ceil(subtotal * TAX\_RATE * 100.0)/100.0$$

Once the taxes have been calculated add together the subtotal and tax for the transaction and save this total for the transaction to the `totals` array at the current `transactionCnt` index.

5.4.3 Getting Cash/Card Selection

The payment for the transaction now needs to be received. As usual this payment can be either cash or card. to get this selection you will need to declare/define a function:

```
char getPaymentType()
```

This function will prompt the user for a selection using the prompt:

```
Payment Type
C/c - Cash
D/d - Card
Selection:
```

The selection will then be read into a character variable. This variable should then be checked that it is equivalent to `'C'`, `'c'`, `'D'`, or `'d'`. If it is not an error message `"Error: Invalid selection"` should be output. The user should then be re-prompted to enter a selection, etc. If `'C'` or `'c'` is entered then `'c'` should be returned. If `'D'` or `'d'` is entered then `'d'` should be returned.

This function can then be called from `addTransaction()` and save the payment type that is selected in a character variable (called `paymentType` in this handout).

5.4.4 Capturing Card Payment and Printing Receipt

If the user selected 'C' or 'c' then paymentType will be equivalent to 'c'. This means the user wants to use cash. The amount tendered needs to be read in. To do this you will need to declare/define a function:

```
string getPayment(double total)
```

This function will first output to the screen the prompt "Total: $" followed by the total that was passed as a parameter. It will then prompt the user to enter a tendered amount using the prompt "Tendered: " and read the tendered amount into a double variable. You then need to perform error checking on the selection the user made:

- If input failure happens when reading in the tendered amount or if the tendered amount is less than the total an error should be output: "Error: must receive at least $total" where total is the number passed to the function. The program should then re-prompt for a new tendered amount etc.
- If no error occurs then the change should be output to the screen by first outputting "Change: $" and then outputting the change by subtracting the total from the tendered amount. The tendered amount can then be cast to a string using to_string(), which will cause a lot of trailing 0's to be added to the string version of the number. These can easily be removed by finding the location of the '.' using string.find('.') and taking a substring from the beginning of the string to the location of the '.' using string.substr(0, location + 3).

Now that getPayment() has been declared/defined it can be called in addTransaction() and the result saved in a string variable (called payment in this handout). The payment type and payment amount now need to be saved to the paymentTypes and payments arrays for each of the items added for this transaction. For this you will need to know the start and end indices in these 2 arrays to add the payment type and payment amount to. The end point has already been found, it is itemCnt since this transaction is the last transaction in the arrays. The start is itemCnt prior to adding any items in this function. Simply save the item count into an integer variable (called start in this handout) prior to adding any items to the parallel arrays. You can then loop from start to itemCnt writing "cash" to the paymentTypes array and the value contained in payment to the payments array.

Now the receipt must be printed. Luckily 2 functions have been provided to print the receipt for us. Before calling these 2 functions a ofstream object must be created to print the receipt to (called receiptFile in this handout). First, open a file named "receipt_ID.txt" where ID is equivalent to the transactionCnt + 1 (i.e. receipt_2.txt) into receiptFile. Then, call the function printReceiptTop() (which has been provided for you in provided_functions.cpp) and pass it receiptFile, start, itemCnt, names, costs, qtys, the subtotal (contained in subtotals at transactionCnt), the tax (contained in taxes at transactionCnt), and the total (contained in totals at transactionCnt). Pass these parameters in this exact order. This call will print the top of the receipt and the items purchased. Lastly, call stod on the string payment and save the double value in a variable and use this variable to calculate the change by subtracting the total amount from the tendered amount. You can then call the function printReceiptBottom() (which has been provided for you in provided_functions.cpp) and pass it oFile, d_payment, and change in that exact order. This call will print the receipt bottom with the payment amount and change.

5.4.5 Capturing Cash Payment and Printing Receipt

If the user selected 'D' or 'd' then paymentType will be equivalent to 'd'. This means the user wants to use card. The card number needs to be read in. To do this you will need to declare/define a function (which overloads the function written in 5.4.4):

```
void getPayment(string& cardNum, string& cardType)
```

This function will prompt the user to enter a card number using the prompt `"Card Number: "` and read the card number into the `cardNum` reference parameter passed to this function. You then need to perform error checking on the selection the user made. Luckily a function has been provided in `provided_functions.cpp` which will error check the `cardNum`, this function is called `getCardType()`. `getCardType()` takes in a card number and returns the type (visa, master, amex, or discover) of the card if the card number is valid, if the card number is invalid it returns an empty string. Call `getCardType()` and pass it `cardNum`, save the value returned from this call to the `cardType` reference parameter passed to this function. If the `cardType` is equivalent to the empty string then output an error message `"Error: Invalid card number"` and re-prompt the user to to enter the card number etc. If no error happened then the card number will be saved/returned in the `cardNum` reference parameter and the card type will be saved/returned in the `cardType` reference parameter.

Now that this overloaded version of `getPayment()` has been declared/defined it can be called in `addTransaction()` and the results saved in two string variables which are passed by reference in the call (called `payment` and `cardType` in this handout). The payment type and card number now need to be saved to the `paymentTypes` and `payments` arrays for each of the items added for this transaction. Just like before the start point for the transaction in these 2 arrays is `itemCnt` prior to adding any items and the end point is `itemCnt` after. Again, you can then loop from `start` to `itemCnt` writing `"card"` this time to the `paymentTypes` array and the value contained in `payment` (the card number this time) to the `payments` array.

Lastly, the card receipt must be printed. Again, 2 functions have been provided to print the receipt for us. Before calling these 2 functions a `ofstream` object must be created to print the receipt to (called `receiptFile` in this handout). First, open a file named `"receipt_ID.txt"` where `ID` is equivalent to the `transactionCnt` + 1 (i.e. `receipt_2.txt`) into `receiptFile`. Then, call the function `printReceiptTop()` (which has been provided for you in `provided_functions.cpp`) and pass it `receiptFile`, `start`, `itemCnt`, `names`, `costs`, `qtys`, the subtotal (contained in `subtotals` at `transactionCnt`), the tax (contained in `taxes` at `transactionCnt`), and the total (contained in `totals` at `transactionCnt`). Pass these parameters in this exact order. This call will print the top of the receipt and the items purchased. Lastly, call the overloaded function `printReceiptBottom()` (which has been provided for you in `provided_functions.cpp`) and pass it `oFile`, `cardNum`, and `cardType` in that exact order. This call will print the receipt bottom with the card type and last 4 of the card number.

5.4.6 Add Transaction Conclusions

After either the cash or card receipt is printed the transactionCnt should be incremented by 1. This is the end point of `addTransaction()`. Call `addTransaction()` in `main()` after the `TODO` comment for section 5.4. You will need to pass this function call `itemCnt`, `transactionCnt`, `ids`, `names`, `qtys`, `costs`, `paymentTypes`, `payments`, `subtotals`, `taxes`, `totals`, `invNames`, and `invCosts` in this order.

5.5 Notes

You must write the every function described in this section naming them exactly as described and passing the exact parameters in the exact order they are described. Failure to do so will result in you receiving a 0 for this assignment. You should only edit `main()` by adding calls to 6 functions as described above. Any other edits to main will result in you receiving a 0 for this assignment. A table of the function prototypes for all the functions you must add follows.

Section Used	Prototype
5.3	`char getMode()`
5.4	`void addTransaction(int&, int&, int[], string[], int[], double[], string[], string[],double[], double[], double[], string[], double[])`
5.4.1	`int getQuantity()`
5.4.3	`char getPaymentType()`
5.4.4	`string getPayment(double)`
5.4.5	`void getPayment(string&, string&)`

Table 11: Function prototypes for section 5

5.6 Example Output

Below are sample executions of the complete program. Text in red is input from the user, this coloring is simply here to distinguish components and not needed in your program.

Example **1**

```
Alexs -iMac desktop % g++ main.cpp
Alexs -iMac desktop % ./a.out
Usage: ./a.out <file name>
```

Example **2**

```
Alexs -iMac desktop % ./a.out too many arguments
Usage: ./a.out <file name>
```

Example **3**

```
Alexs -iMac desktop % ./a.out invalid.csv
Error: Invalid file name
```

Example **4**

```
Alexs -iMac desktop AS5 % ./a.out data/s4-1.csv
+------------------------------------------------------------------------------+
|    UU     UU  NNNN     NN  LL       VV       VV         CCCCCC    SSSSSSSS   |
|   /UU    /UU /NN/NN   /NN /LL      /VV      /VV        CC////CC  SS//////   |
|   /UU    /UU /NN//NN  /NN /LL      /VV      /VV       CC    //  /SS         |
|   /UU    /UU /NN //NN /NN /LL      //VV    VV        /CC        /SSSSSSSSS  |
|   /UU    /UU /NN  //NN/NN /LL       //VV  VV        /CC        ////////SS  |
|   /UU    /UU /NN   //NNNN /LL        //VVVV         //CC    CC        /SS   |
|   //UUUUUUU  /NN    //NNN /LLLLLLLL   //VV           //CCCCCC   SSSSSSSS   |
|    ///////   //      ///  ////////     //             //////   ////////     |
|                                                                              |
|                  SSSSSSSS  HH      HH    OOOOOOO    PPPPPPP                 |
|                 SS//////  /HH     /HH   OO/////OO  /PP////PP                |
|                /SS        /HH     /HH  OO     //OO /PP   /PP                |
|                /SSSSSSSSS /HHHHHHHHHH /OO       /OO /PPPPPPP                |
|                ////////SS /HH//////HH /OO       /OO /PP////                 |
|                       /SS /HH     /HH //OO     OO  /PP                      |
|                 SSSSSSSS  /HH     /HH  //OOOOOOO   /PP                      |
|                ////////   //      //    ///////    //                       |
+------------------------------------------------------------------------------+

OPTIONS
S/s - Print Summary
T/t - Add Transaction
E/e - Exit
Selection: e
Exiting...
```

Example **6**

```
Alexs -iMac desktop AS5 % ./a.out data/s4-1.csv
(header truncated for space)

OPTIONS
S/s - Print Summary
T/t - Add Transaction
E/e - Exit
Selection: T

+--------------------------------+--------------------------------+------------------------------+
| -1   End Transaction           |                                |                              |
| 0    Laptop              799.99 | 9    Speakers            79.99 | 18   iPhone Charger    9.99 |
| 1    Desktop             949.99 | 10   Headphones          99.99 | 19   Android Charger   9.99 |
| 2    Monitor             249.99 | 11   Microphone          39.99 | 20   Laptop Charger   19.99 |
| 3    Mouse                15.95 | 12   Webcam              24.98 | 21   Notebook          1.49 |
| 4    Keyboard             15.95 | 13   Flash Drive          4.48 | 22   Pen               0.99 |
| 5    Laptop Bag           19.99 | 14   Hard Drive          39.99 | 23   Pencil            0.99 |
| 6    Laptop Case          19.99 | 15   Solid State Drive   69.95 | 24   Drinks            1.29 |
| 7    Laptop Riser Stand   24.99 | 16   CPU Upgrade        299.99 | 25   Chips             1.49 |
| 8    Laptop Lock          14.99 | 17   RAM Upgrade        149.99 | 26   Coffee            0.99 |
+--------------------------------+--------------------------------+------------------------------+
Selected Item: error
Error: Invalid selection

Selected Item: invalid
Error: Invalid selection

Selected Item: 0
Quantity: 1

+--------------------------------+--------------------------------+------------------------------+
| -1   End Transaction           |                                |                              |
| 0    Laptop              799.99 | 9    Speakers            79.99 | 18   iPhone Charger    9.99 |
| 1    Desktop             949.99 | 10   Headphones          99.99 | 19   Android Charger   9.99 |
| 2    Monitor             249.99 | 11   Microphone          39.99 | 20   Laptop Charger   19.99 |
| 3    Mouse                15.95 | 12   Webcam              24.98 | 21   Notebook          1.49 |
| 4    Keyboard             15.95 | 13   Flash Drive          4.48 | 22   Pen               0.99 |
| 5    Laptop Bag           19.99 | 14   Hard Drive          39.99 | 23   Pencil            0.99 |
| 6    Laptop Case          19.99 | 15   Solid State Drive   69.95 | 24   Drinks            1.29 |
| 7    Laptop Riser Stand   24.99 | 16   CPU Upgrade        299.99 | 25   Chips             1.49 |
| 8    Laptop Lock          14.99 | 17   RAM Upgrade        149.99 | 26   Coffee            0.99 |
+--------------------------------+--------------------------------+------------------------------+
Selected Item: 5
Quantity: 1

+--------------------------------+--------------------------------+------------------------------+
| -1   End Transaction           |                                |                              |
| 0    Laptop              799.99 | 9    Speakers            79.99 | 18   iPhone Charger    9.99 |
| 1    Desktop             949.99 | 10   Headphones          99.99 | 19   Android Charger   9.99 |
| 2    Monitor             249.99 | 11   Microphone          39.99 | 20   Laptop Charger   19.99 |
| 3    Mouse                15.95 | 12   Webcam              24.98 | 21   Notebook          1.49 |
| 4    Keyboard             15.95 | 13   Flash Drive          4.48 | 22   Pen               0.99 |
| 5    Laptop Bag           19.99 | 14   Hard Drive          39.99 | 23   Pencil            0.99 |
| 6    Laptop Case          19.99 | 15   Solid State Drive   69.95 | 24   Drinks            1.29 |
| 7    Laptop Riser Stand   24.99 | 16   CPU Upgrade        299.99 | 25   Chips             1.49 |
| 8    Laptop Lock          14.99 | 17   RAM Upgrade        149.99 | 26   Coffee            0.99 |
+--------------------------------+--------------------------------+------------------------------+
Selected Item: -1

Payment Type
C/c - Cash
D/d - Card
Selection: c

Total: $887.63

```

```
Tendered: 500
Error: Must receive at least $887.63

Tendered: 1000

Change: $112.37

OPTIONS
S/s - Print Summary
T/t - Add Transaction
E/e - Exit
Selection: S

Printing sales report to summary.txt...

OPTIONS
S/s - Print Summary
T/t - Add Transaction
E/e - Exit
Selection: e
Exiting...
Alexs -iMac desktop AS5 % cat summary.txt
+-------------------------------------------------------------+
| Total Transactions.....................................3 |
|                                                             |
| Total Pre-tax Sales................................1694.65 |
| Total Taxes Charged................................139.82 |
| Total Post-tax Sales...............................1834.47 |
|                                                             |
| Average Pre-tax Sale...............................564.883 |
| Average Post-tax Sale..............................611.49 |
|                                                             |
| Lowest Sale ID.........................................1 |
| Lowest Sale Subtotal...................................50.22 |
| Lowest Sale Tax.....................................4.15 |
| Lowest Sale Total.....................................54.37 |
|                                                             |
| Highest Sale ID........................................0 |
| Highest Sale Subtotal..............................824.45 |
| Highest Sale Tax....................................68.02 |
| Highest Sale Total.................................892.47 |
+-------------------------------------------------------------+
Alexs -iMac desktop AS5 % cat receipt-3.txt
/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
|                                             |
|              UNLV CS Shop                   |
|          4505 S Maryland Pkwy               |
|           Las Vegas,NV 89154                |
|             (702) 895-3011                  |
|                                             |
| Laptop                      799.99 x 1      |
| Laptop Bag                   19.99 x 1      |
|                                             |
| Subtotal                        819.98      |
| Tax Rate                         8.25%      |
| Tax                              67.65      |
| Total                           887.63      |
|                                             |
| Tendered                       1000.00      |
| Change                          112.37      |
|                                             |
\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
```

5.7 Submission

The first thing you need to do before submitting this assignment is read and understand the **Academic Integrity Policy** at the beginning of this handout. By submitting this assignment you are accepting this policy even if you did not sign/submit the required **Academic Integrity Policy Acknowledgement**.

Make sure your code is saved as **main.cpp**. Do not ignore this step or save your file with different names. Do not submit `provided_functions.cpp` to CodeGrade, it is already included in CodeGrade.

Your program source code must be submitted via **CodeGrade** as a properly named **.cpp** file. **Late submissions will not be accepted**.