Ben Small

Vending Machine Requirements

1. The system shall allow the machine operator to use any two horizontally adjacent slots for a wide product. For products occupying two slots in the machine, the system shall use leftmost code of the two slots. For example, if a product is occupying slots A3 and A4, A3 will correspond to that product, A4 will be an invalid code, and A5 will be the code for the next product to the right.

2. The system shall allow the machine operator to set a price for any valid slot.

3. The system shall accept currency in the form of $1 bills, $5 bills, quarters, dimes, nickels, and $1 coins, and shall update the balance by 1.00, 5.00, 0.25, 0.10, 0.05, and 1.00 respectively as each bill or coin is accepted.

4. When the customer enters a letter (between A and E) followed by a number (between 0 and 9), the system shall determine if the entered product code (e.g., A9) corresponds to a valid slot in the machine's current configuration.

5. The system shall display the price of an item when a valid product code (a letter A-E followed by a valid number 0-9) are pressed by the customer and insufficient funds to purchase that item have been deposited.

6. If an entered letter followed by an entered number is not a valid code, the machine shall display “INVALID CODE” in lieu of an item’s price.

7. If sufficient funds have been deposited, the system shall turn the crank(s) corresponding to the slot after a valid product code (e.g., A9) has been entered.

8. The system shall return change equal to the amount of money deposited minus the price of the vended product. Change shall be returned in the largest coin denomination(s) available. For example, assuming the machine is not out of any coin types, $1.65 will be returned as one dollar coin, two quarters, one dime, and one nickel.

a. If the machine is out of dollar coins, four quarters will be dispensed instead.

b. If the machine is out of quarters, two dimes and one nickel will be dispensed instead.

c. If the machine is out of dimes, two nickels will be dispensed instead.

9. When a valid product code has been entered but the amount of money deposited is less than the corresponding product price, the system shall display the price of that item for three seconds.

10. If a customer cancels the transaction before making a final item selection, the system shall return the amount deposited by the customer using the convention in Requirement 8.

11. The system shall not accept any excess currency (i.e., money deposited after current balance already exceeds the maximum price of a single item in the machine).

12. The system shall know if there is sufficient change in the machine to complete a transaction. If there is not sufficient change, the machine will not accept any money. This means the system must know whether or not there is:

a. enough change to refund at least $4.95 + the current maximum price of a product, and

b. at least three quarters and four nickels in the machine.

For Requirement 12, we can assume the machine's hardware has a sensor that determines if this is true.

Test Plan

Operator Test Cases

Note for other systems: anything with (work in progress) was for my own entertainment and was not a requirement thus I would expect them to fail on other systems if used. That being said, it is included in expected output for my system, and had to be included.

Test Case 1.1: Operator wants to skip setup

Precondition: program is in status of asking if the user wants to enter operator mode

Input: n

Expected output: “Please input money, row selection or cancel. ($5, $1, d1, d25, d19, d5,(A-E) or cancel)”

Actual output: “Please input money, row selection or cancel. ($5, $1, d1, d25, d19, d5,(A-E) or cancel)”

Pass/Fail: Pass

Test Case 1.2: Operator wants to go to operator setup.

Precondition: program is in status of asking if the user wants to enter operator mode

Input: y

Expected output: “Operator mode selected”

“Please select one of the following options:”

"1: restock item (Work in Progress)"
"2: change item (Work in Progress)"
"3: use wide item"
"4: change price"
“9: exit"

Actual output: “Operator mode selected”

“Please select one of the following options:”

"1: restock item (Work in Progress)"

"2: change item (Work in Progress)"

"3: use wide item"

"4: change price"

“9: exit"

Pass/Fail: Pass

Test Case 1.3: Operator tries to get to both work in progress functions before going to wide setting.

Preconditions: program is in status of asking if the user wants to enter operator mode

Options 1 and 2 are work in progress functions that send user back to main menu.

Inputs: A: y

B: 1

C: 2

D: 3

Expected output: A: “Operator mode selected”

“Please select one of the following options:”

"1: restock item (Work in Progress)"

"2: change item (Work in Progress)"

"3: use wide item"

"4: change price"

“9: exit"

B: “This a work in progress please choose a different number”

“Please select one of the following options:”

"1: restock item (Work in Progress)"

"2: change item (Work in Progress)"

"3: use wide item"

"4: change price"

“9: exit"

C: “This a work in progress please choose a different number”

“Please select one of the following options:”

"1: restock item (Work in Progress)"

"2: change item (Work in Progress)"

"3: use wide item"

"4: change price"

“9: exit"

D: “Please select the row of the slot for the product that is wide. (A-E)”

Actual output: A: “Operator mode selected”

“Please select one of the following options:”

"1: restock item (Work in Progress)"

"2: change item (Work in Progress)"

"3: use wide item"

"4: change price"

“9: exit"

B: “This a work in progress please choose a different number”

“Please select one of the following options:”

"1: restock item (Work in Progress)"

"2: change item (Work in Progress)"

"3: use wide item"

"4: change price"

“9: exit"

C: “This a work in progress please choose a different number”

“Please select one of the following options:”

"1: restock item (Work in Progress)"

"2: change item (Work in Progress)"

"3: use wide item"

"4: change price"

“9: exit"

D: “Please select the row of the slot for the product that is wide. (A-E)”

Pass/Fail: Pass

Test Case 2.1: Operator wants to change an item to wide.

Preconditions: operator mode is selected

The selection of A 2 is not already a wide slot for another item

all inputs are kept separate (press enter for row before putting in column)

Input: A: 3

B: A

C: 2

Expected output: A: "Please select the row of the slot for the product that is wide.(A-E)”

B: “Please select column of item that is wide.(0 - 8. (9 is not a valid selection))”

C: “Selection A2 has been made wide using A3.”

“Returning to operator menu.”

“Please select one of the following options:”

"1: restock item (Work in Progress)"

"2: change item (Work in Progress)"

"3: use wide item"

"4: change price"

“9: exit"

Actual output: A: "Please select the row of the slot for the product that is wide.(A-E)”

B: “Please select column of item that is wide.(0 - 8. (9 is not a valid selection))”

C: “Selection A2 has been made wide using A3.”

“Returning to operator menu.”

“Please select one of the following options:”

"1: restock item (Work in Progress)"

"2: change item (Work in Progress)"

"3: use wide item"

"4: change price"

“9: exit"

Pass/Fail: Pass

Test Case 2.2: Operator inputs an invalid choice for the column selection

Preconditions: operator mode is selected.

Operator is using the use wide item selection

Operator input a valid row selection

Input: T

Expected output: “Invalid column selected.”

“Returning to operator menu.”

“Please select one of the following options:”

"1: restock item (Work in Progress)"

"2: change item (Work in Progress)"

"3: use wide item"

"4: change price"

“9: exit"

Actual output: “Invalid column selected.”

“Returning to operator menu.”

“Please select one of the following options:”

"1: restock item (Work in Progress)"

"2: change item (Work in Progress)"

"3: use wide item"

"4: change price"

“9: exit"

Pass/Fail: Pass*

*I explain the star further in the closing comments as it was noted when I had to make a change to the program due to a bug.

Test Case 2.3: Operator inputs an invalid choice for row selection.

Preconditions: Operator mode is selected.

Operator is using the use wide item selection.

Input: 4

Expected output: “Invalid row selected.”

“Returning to operator menu.”

“Please select one of the following options:”

"1: restock item (Work in Progress)"

"2: change item (Work in Progress)"

"3: use wide item"

"4: change price"

“9: exit"

Actual output: “Invalid row selected.”

“Returning to operator menu.”

“Please select one of the following options:”

"1: restock item (Work in Progress)"

"2: change item (Work in Progress)"

"3: use wide item"

"4: change price"

“9: exit"

Pass/Fail: Pass

Test Case 3.1: Operator wants to select change price

Preconditions: Operator mode is selected.

Input: 4

Expected output: “Please select the product row to change price.(A-E)”

Actual output: “Please select the product row to change price.(A-E)”

Pass/Fail: Pass

Test Case 3.2: Operator wants to change price of an item but inputs an invalid row selection.

Preconditions: Operator mode is selected.

Operator is using change price selection.

Input: 1

Expected output: “Invalid row selected.”

“Returning to operator menu.”

“Please select one of the following options:”

"1: restock item (Work in Progress)"

"2: change item (Work in Progress)"

"3: use wide item"

"4: change price"

“9: exit"

Actual output: “Invalid row selected.”

“Returning to operator menu.”

“Please select one of the following options:”

"1: restock item (Work in Progress)"

"2: change item (Work in Progress)"

"3: use wide item"

"4: change price"

“9: exit"

Pass/Fail: Pass

Test Case 3.2: Operator inputs a valid row selection for change price.

Preconditions: Operator mode is selected.

Change price mode is selected.

Input: A

Expected output: “Please select column of item to change price.(A-E)”

Actual output: “Please select column of item to change price.(A-E)”

Pass/Fail: Pass

Test Case 3.3: Operator inputs an invalid column selection for change price.

Preconditions: Operator mode is selected.

Change price mode is selected.

Operator has used a valid row.

Input: A

Expected output: “Invalid column selected.”

“Returning to operator menu.”

“Please select one of the following options:”

"1: restock item (Work in Progress)"

"2: change item (Work in Progress)"

"3: use wide item"

"4: change price"

“9: exit"

Actual output: “Invalid column selected.”

“Returning to operator menu.”

“Please select one of the following options:”

"1: restock item (Work in Progress)"

"2: change item (Work in Progress)"

"3: use wide item"

"4: change price"

“9: exit"

Pass/Fail: Pass

Test Case 3.4: Operator inputs a valid column selection for change price.

Preconditions: Operator mode is selected.

Change price mode is selected.

Operator has used row A.

Selection A 2 is not a wide item.

Input: 2

Expected output: “Please input the price (Do not include decimal. ex $1.25 input as 125)”

Actual output: “Please input the price (Do not include decimal. ex $1.25 input as 125)”

Pass/Fail: Pass

Test Case 3.5: Operator selects a wide item to try to change the price of.

Preconditions: Operator mode is selected.

Change price mode is selected.

Operator has used row A.

Selection A 2 is a wide item.

Input: 2

Expected output: “Cannot change price for extra slots of a wide item.”

Actual output: “Cannot change price for extra slots of a wide item.”

Pass/Fail: Pass

Test Case 3.6: Operator is changing the price.

Preconditions: Operator mode is selected.

Change price mode is selected.

Operator has used row A.

Operator has used column 2.

Selection A 2 is not a wide item.

Selection A 2 is Cheetos.

Selection A 2 has 10 items in stock.

Input: 125

Expected output: “Cheetos $1.50 10”

“Returning to operator menu.”

“Please select one of the following options:”

"1: restock item (Work in Progress)"

"2: change item (Work in Progress)"

"3: use wide item"

"4: change price"

“9: exit"

Actual output: “Cheetos $1.50 10”

“Returning to operator menu.”

“Please select one of the following options:”

"1: restock item (Work in Progress)"

"2: change item (Work in Progress)"

"3: use wide item"

"4: change price"

“9: exit"

Pass/Fail: Pass

Customer Test Cases

Test Case 4.1.1: Customer inputs a $5 bill

Preconditions: there is enough coins in the machine.

Customer has not input more money than $4.95 + the highest priced item

Machine is in vending mode

Input: $5

Expected output: “Please input money, row selection or cancel. ($5, $1, d1, d25, d19, d5,(A-E) or cancel)”

Actual output: “Please input money, row selection or cancel. ($5, $1, d1, d25, d19, d5,(A-E) or cancel)”

Pass/Fail: Pass

Test Case 4.1.2: Customer tries to input a $5 bill after exceeding the cost limit.

Preconditions: There is enough coins in the machine.

Customer has input more money than $4.95 + the highest priced item

Machine is in vending mode.

Input: $5

Expected output: “Too much money inserted, returning your $5.”

Actual output: “Too much money inserted, returning your $5.”

Pass/Fail: Pass

Test Case 4.2: Customer inputs a $1 bill

Preconditions: there is enough coins in the machine.

Customer has not input more money than $4.95 + the highest priced item

Machine is in vending mode

Input: $1

Expected output: “Please input money, row selection or cancel. ($5, $1, d1, d25, d19, d5,(A-E) or cancel)”

Actual output: “Please input money, row selection or cancel. ($5, $1, d1, d25, d19, d5,(A-E) or cancel)”

Pass/Fail: Pass

Test Case 4.3: Customer inputs a quarter.

Preconditions: there is enough coins in the machine.

Customer has not input more money than $4.95 + the highest priced item

Machine is in vending mode

Input: d25

Expected output: “Please input money, row selection or cancel. ($5, $1, d1, d25, d19, d5,(A-E) or cancel)”

Actual output: “Please input money, row selection or cancel. ($5, $1, d1, d25, d19, d5,(A-E) or cancel)”

Pass/Fail: Pass

Test Case 4.4: Customer inputs a dime.

Preconditions: there is enough coins in the machine.

Customer has not input more money than $4.95 + the highest priced item

Machine is in vending mode

Input: d10

Expected output: “Please input money, row selection or cancel. ($5, $1, d1, d25, d19, d5,(A-E) or cancel)”

Actual output: “Please input money, row selection or cancel. ($5, $1, d1, d25, d19, d5,(A-E) or cancel)”

Pass/Fail: Pass

Test Case 4.5: Customer inputs a nickel.

Preconditions: there is enough coins in the machine.

Customer has not input more money than $4.95 + the highest priced item

Machine is in vending mode

Input: d5

Expected output: “Please input money, row selection or cancel. ($5, $1, d1, d25, d19, d5,(A-E) or cancel)”

Actual output: “Please input money, row selection or cancel. ($5, $1, d1, d25, d19, d5,(A-E) or cancel)”

Pass/Fail: Pass

Test Case 5.1: Customer inputs an invalid row selection

Preconditions: there is enough coins in the machine.

Machine is in vending mode.

Customer has input $5

Input: 7

Expected output: “Invalid input”

“Returning your money of $5”

“Exiting”

Actual output: “Invalid input”

“Returning your money of $5”

“Exiting”

Pass/Fail: Pass

Test Case 5.2: Customer inputs a valid row selection.

Preconditions: there is enough coins in the machine.

Machine is in vending mode.

Input: A

Expected output: “Please select column selection(0 - 9)”

Actual output: “Please select column selection(0 - 9)”

Pass/Fail: Pass

Test Case 5.3: Customer inputs an invalid column selection.

Preconditions: there is enough coins in the machine.

Machine is in vending mode.

Customer has used a valid row selection.

Input: A

Expected output: “Invalid column”

“Please input money, row selection or cancel. ($5, $1, d1, d25, d19, d5,(A-E) or cancel)”

Actual output: “Invalid column”

“Please input money, row selection or cancel. ($5, $1, d1, d25, d19, d5,(A-E) or cancel)”

Pass/Fail: Pass

Test Case 5.4: Customer inputs a column that has been set to wide.

Preconditions: there is enough coins in the machine.

Machine is in vending mode.

A 1 is a slot used for a wide item.

Customer has input A as the row selection.

Input: 1

Expected output: “Cannot select the right side of large items.”

“Please input money, row selection or cancel. ($5, $1, d1, d25, d19, d5,(A-E) or cancel)”

Actual output: “Cannot select the right side of large items.”

“Please input money, row selection or cancel. ($5, $1, d1, d25, d19, d5,(A-E) or cancel)”

Pass/Fail: Pass

Test Case 5.5: Customer tries to purchase an item with insufficient change.

Preconditions: there is enough coins in the machine.

Machine is in vending mode.

A 2 costs $1.25

Customer has input $1.

Customer has input A as row selection.

Input: 2

Expected output: “Price of item is $1.25”

“Current funds are $1”

“Please input money, row selection or cancel. ($5, $1, d1, d25, d19, d5,(A-E) or cancel)”

Actual output: “Price of item is $1.25”

“Current funds are $1”

“Please input money, row selection or cancel. ($5, $1, d1, d25, d19, d5,(A-E) or cancel)”

Pass/Fail: Pass

Test Case 5.6: Customer tries to purchase an item from an empty slot.

Preconditions: there is enough coins in the machine.

Machine is in vending mode.

B 0 costs $1.50

B 0 is out of stock

Customer input B as row selection.

Customer has input $5.

Input: 0

Expected output: “Selected empty slot, have a nice day”

“exiting”

Actual output: “Selected empty slot, have a nice day”

“exiting”

Pass/Fail: Pass

Test Case 5.7: Customer selects a valid, in stock item.

Preconditions: there is enough coins in the machine.

Machine is in vending mode.

A 0 costs $1.50.

A 0 is Cheetos

Customer input A as row selection.

Customer has input $6.45.

There is 16 quarters, 12 dimes, and 40 nickels in the machine.

Input: 0

Expected output: “Dispensing Cheetos.”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d10”

“Dispensing d10”

“Dispensing d10”

“Dispensing d10”

“Dispensing d10”

“Dispensing d10”

“Dispensing d10”

“Dispensing d10”

“Dispensing d10”

“Dispensing d5”

“exiting”

Actual output: “Dispensing Cheetos.”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d10”

“Dispensing d10”

“Dispensing d10”

“Dispensing d10”

“Dispensing d10”

“Dispensing d10”

“Dispensing d10”

“Dispensing d10”

“Dispensing d10”

“Dispensing d5”

“exiting”

Pass/Fail: Pass

Test Case 6.1: Customer cancels the purchase

Preconditions: there is enough coins in the machine.

Machine is in vending mode.

Customer has input $5

There is 16 quarters, 12 dimes, and 40 nickels in the machine.

Input: cancel

Expected output: “Returning your money of $5.00”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d10”

“Dispensing d10”

“Dispensing d10”

“Dispensing d10”

“Dispensing d10”

“Dispensing d10”

“Dispensing d10”

“Dispensing d10”

“Dispensing d10”

“Dispensing d10”

“exiting”

Actual output: “Returning your money of $5.00”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d25”

“Dispensing d10”

“Dispensing d10”

“Dispensing d10”

“Dispensing d10”

“Dispensing d10”

“Dispensing d10”

“Dispensing d10”

“Dispensing d10”

“Dispensing d10”

“Dispensing d10”

“Dispensing d10”

“exiting”

Pass/Fail: Pass

Test Case 7.1: Customer tries to input more money than $4.95 + the highest price item.

Preconditions: there is enough coins in the machine.

Machine is in vending mode.

Customer has input $6.75.

$4.95 + the highest priced item ($1.75) is $6.70

Input: $5

Expected output: “Too much money inserted, returning $5”

Actual output: “Too much money inserted, returning $5”

Pass/Fail: Pass

Test Case 8.1: There is not enough money in the machine. (<$4.95)

Preconditions: There is not enough change in the machine.

In coin shortage mode.

There are 3 quarters and 4 nickels in the machine.

Input: (intentionally left blank)

Expected output: “Cannot engage transaction. Not enough change in storage”
“exiting”

Actual output: “Cannot engage transaction. Not enough change in storage”
“exiting”

Pass/Fail: Pass

Test Case 8.2: there are not enough quarters or nickels in storage.

Preconditions: there is 2 quarters and 5 nickels in the machine.
Machine is in vending mode.
In coin shortage mode.

Input: (intentionally left blank)

Expected output: “Cannot engage transaction, not enough quarters or nickels.”
“exiting”

Actual output: “Cannot engage transaction, not enough quarters or nickels.”
“exiting”

Pass/Fail: Pass

Forward Traceability Matrix

Requirement number	Test Case Number
1	2.1, 2.2, 2.3
2	3.1, 3.2, 3.3, 3.4, 3.5, 3.6
3	4.1.1, 4.1.2, 4.2, 4.3, 4.4, 4.5
4	5.1 – 5.7
5	5.5
6	5.1, 5.2, 5.3
7	5.5, 5.6, 5.7
8	5.7, 6.1
9	5.5
10	6.1
11	7.1
12	8.1, 8.2

Annotated Code Snippet

A
```
cout << "Startup complete, begin operator mode?(y/n)\n";
bool mode = false;
string modeSelect;
int choice;
cin >> modeSelect;
if (modeSelect == "y" || modeSelect == "Y") {
    cout << "Operator mode selected\n";
    mode = true;
}
```

B
```
while (mode == true) {
```

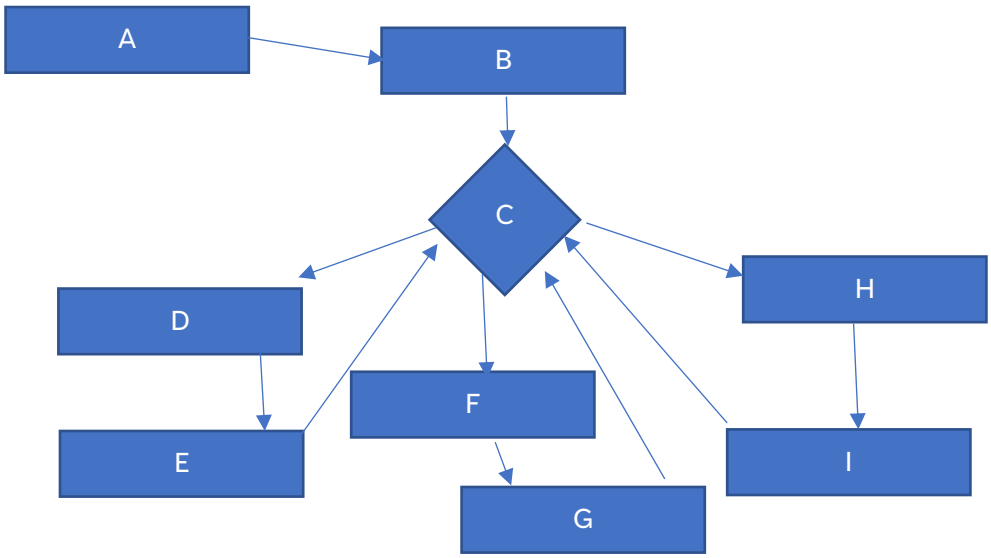
C
```

    cout << "Please select one of the following options:\n";
    cout << "1: restock item (Work in Progress)\n";
    cout << "2: change item (Work in Progress)\n";
    cout << "3: use wide item\n";
    cout << "4: change price\n";
    cout << "9: exit\n";
    cin >> choice;
    string opRow;
    bool valOpRow = true;
    int opCol;
    int priceChange;
    int opSelect = 0;
```

D
```
    if (choice == 1) {
```

E
```
        cout << "This is a work in progress please choose a different number\n";
    }
```

F
```
    else if (choice == 2) {
```

```
G        cout << "This is a work in progress please choose a different number\n";
       }
H    else if (choice == 3) {
I        cout << "Please select the row of the slot for the product that is wide.(A-E)\n";
         cin >> opRow;
```



Code Graph

Statement and branch coverage is the same and can be done in 1 go as it is heavily reliant on the while loop

Path	Test Case
A B C D E C F G C H I C	1.3

Comments:

I found a small bug with column selection that if the input was a letter the program would assume the column number was 0. I would not have caught this if I had not tested the program with a letter instead of column number.

Overall, a very fun and insightful project.

Sample Output

Sample taken from startup to purchase of AA batteries:

Beginning startup

Section	Item Name	Price	Amount in stock
A0	Cheetos	$1.5	5
A2	Nacho Cheese Doritos	$1.25	10
A4	Cool Ranch Doritos	$1.25	10
A6	Munchies Cheese Mix	$1.25	10
A8	Smartfood Popcorn	$1	10
B0	Original Fritos	$1.25	0
B2	Lays Original	$1.25	10
B4	Lays Barbeque	$1.25	10
B6	Rold Gold Pretzels	$1.25	10
B8	Cheez It	$1.25	10
C0	Oreo	$1	10
C2	Pop Tart Strawberry	$1.25	10
C4	Chex Mix	$1.25	10
C6	Beef Jerky	$1.5	10
C8	Gummy Bears	$1.25	10
D0	Mike and Ike	$1.25	15
D1	KitKat	$1.25	15
D2	Skittles	$1.25	15
D3	Twix	$1.25	15
D4	Hersheys	$1	10
D5	Sweettarts	$1	10
D6	Cheese and Crackers	$1	10

D7 PB and Crackers $1 10

D8 Breakfast Shake $1.25 10

D9 Breakfast Bar $1.25 10

E0 Face mask $0.5 20

E2 Covid test kit $1 15

E4 Index cards pack $1 10

E5 6GB flash drive $1.75 10

E6 Pen and Pencil $1.25 10

E7 AA batteries $1.5 10

E8 TicTac $1 10

E9 Chewing Gum $1 10

Is there a coin shortage?(y/n)

y

Startup complete, begin operator mode?(y/n)

y

Operator mode selected

Please select one of the following options:

1: restock item (Work in Progress)

2: change item (Work in Progress)

3: use wide item

4: change price

9: exit

3

Please select the row of the slot for the product that is wide.(A-E)

A

Please select column of item that is wide.(0 - 8. (9 is not a valid selection))

8

Selection A8 has been made wide using A9.

Returning to operator menu.

Please select one of the following options:

1: restock item (Work in Progress)

2: change item (Work in Progress)

3: use wide item

4: change price

9: exit

9

Exiting operator mode. Starting customer mode.

Please input money, row selection or cancel. ($5, $1, d1, d25, d19, d5,(A-E) or cancel)

d25

Please input money, row selection or cancel. ($5, $1, d1, d25, d19, d5,(A-E) or cancel)

d25

Please input money, row selection or cancel. ($5, $1, d1, d25, d19, d5,(A-E) or cancel)

d25

Please input money, row selection or cancel. ($5, $1, d1, d25, d19, d5,(A-E) or cancel)

$1

Please input money, row selection or cancel. ($5, $1, d1, d25, d19, d5,(A-E) or cancel)

E

Please select column selection(0 - 9)

7

Dispensing AA batteries

Dispensing d25

exiting.

Code

```
#include <iostream>
#include <string>
#include <array>
#include <cstdlib>
using namespace std;

//begin inventory setup
class inventory {
   string name;
   int price;          //for ease of calculations this is kept as an int but equations are run to
display as a decimal
   int count;          //balance on hand
public:
   inventory(){}

   inventory(string name, int price, int count) {
      this->name = name;
      this->price = price;
      this->count = count;
   }
   void setter(string n, int p, int c) {
      name = n;
      price = p;
      count = c;
   }
   void setName(string n) {
      name = n;
```

```
}
void setPrice(int p) {
    price = p;
}
void setCount(int c) {
    count = c;
}

string getName() {
    return name;
}
int getPrice() {
    return price;
}
double getDolPrice() {
    double Pr = price;
    Pr = Pr / 100;
    return Pr;
}
int getCount() {
    return count;
}
void print(){
    double Price = price;
    Price = Price / 100;

    cout << name << "\t";
    if (name.size() < 8) {
        cout << "\t";
    }
```

```
        if (name.size() < 16) {
            cout << " \t" << "$" << Price << " \t" << count << "\n";
        }
        else {
            cout << "$" << Price << " \t" << count << "\n";
        }
    }

};



//coin storage
class storage {
    string name;    //name of coin
    int value;
    int count;
public:
    storage() {}

    storage(string name, int value, int count) {
        this->name = name;
        this->value = value;
        this->count = count;
    }
    void setter(string n, int v, int c) {
        name = n;
        value = v;
        count = c;
    }
```

```
    void setCount(int n) {
        count = n;
    }
    string getName() {
        return name;
    }
    int getValue() {
        return value;
    }
    int getCount() {
        return count;
    }
};

int main()
{
    cout << "Beginning startup\n";
    //begin startup phase
    inventory* arr = new inventory[50];
    arr[0].setter("Cheetos", 150, 5);
    arr[1].setter("wide", 0, 0);
    arr[2].setter("Nacho Cheese Doritos", 125, 10);
    arr[3].setter("wide", 0, 0);
    arr[4].setter("Cool Ranch Doritos", 125, 10);
    arr[5].setter("wide", 0, 0);
    arr[6].setter("Munchies Cheese Mix", 125, 10);
    arr[7].setter("wide", 0, 0);
    arr[8].setter("Smartfood Popcorn", 100, 10);
    arr[9].setter("wide", 0, 0);
```

```
arr[10].setter("Original Fritos", 125, 0);
arr[11].setter("wide", 0, 0);
arr[12].setter("Lays Original", 125, 10);
arr[13].setter("wide", 0, 0);
arr[14].setter("Lays Barbeque", 125, 10);
arr[15].setter("wide", 0, 0);
arr[16].setter("Rold Gold Pretzels", 125, 10);
arr[17].setter("wide", 0, 0);
arr[18].setter("Cheez It", 125, 10);
arr[19].setter("wide", 0, 0);

arr[20].setter("Oreo", 100, 10);
arr[21].setter("wide", 0, 0);
arr[22].setter("Pop Tart Strawberry", 125, 10);
arr[23].setter("wide", 0, 0);
arr[24].setter("Chex Mix", 125, 10);
arr[25].setter("wide", 0, 0);
arr[26].setter("Beef Jerky", 150, 10);
arr[27].setter("wide", 0, 0);
arr[28].setter("Gummy Bears", 125, 10);
arr[29].setter("wide", 0, 0);

arr[30].setter("Mike and Ike", 125, 15);
arr[31].setter("KitKat", 125, 15);
arr[32].setter("Skittles", 125, 15);
arr[33].setter("Twix", 125, 15);
arr[34].setter("Hersheys", 100, 10);
arr[35].setter("Sweettarts", 100, 10);
arr[36].setter("Cheese and Crackers", 100, 10);
arr[37].setter("PB and Crackers", 100, 10);
```

```
    arr[38].setter("Breakfast Shake", 125, 10);
    arr[39].setter("Breakfast Bar", 125, 10);

    arr[40].setter("Face mask", 50, 20);
    arr[41].setter("wide", 0, 0);
    arr[42].setter("Covid test kit", 100, 15);
    arr[43].setter("wide", 0, 0);
    arr[44].setter("Index cards pack", 100, 10);
    arr[45].setter("6GB flash drive", 175, 10);
    arr[46].setter("Pen and Pencil", 125, 10);
    arr[47].setter("AA batteries", 150, 10);        //yes this was added because of the clicker
from class
    arr[48].setter("TicTac", 100, 10);
    arr[49].setter("Chewing Gum", 100, 10);

    //display
    cout << "Section\t" << "Item Name\t\t" << "Price\t" << "Amount in stock\n";
    for (int i = 0; i <= 4; i++) {
        for (int x = 0; x <= 9; x++) {
            if (arr[(10 * i) + x].getName() != "wide") {
                if (i == 0) {
                    cout << "A";
                }
                else if (i == 1) {
                    cout << "B";
                }
                else if (i == 2) {
                    cout << "C";
                }
                else if (i == 3) {
```

```
            cout << "D";
        }
        else {
            cout << "E";
        }
        cout << x << "\t";
        arr[(10 * i) + x].print();
      }
   }
   cout << "\n";
}



//begin coin storage setup
cout << "Is there a coin shortage?(y/n)\n";
string shortage = "y";
cin >> shortage;
//cout << "y\n";
storage* coins = new storage[6];
if (shortage == "y" || shortage == "Y") {          //used if there is a coin shortage
   coins[0].setter("$5 bill", 500, 0);
   coins[1].setter("$1 bill", 100, 0);
   coins[2].setter("$1 coin", 100, 0);
   coins[3].setter("quarter", 25, 16);
   coins[4].setter("dime", 10, 12);
   coins[5].setter("nickel", 5, 40);
}
else {
   coins[0].setter("$5 bill", 500, 5);
```

```
    coins[1].setter("$1 bill", 100, 10);
    coins[2].setter("$1 coin", 100, 200);
    coins[3].setter("quarter", 25, 400);
    coins[4].setter("dime", 10, 500);
    coins[5].setter("nickel", 5, 1000);
}


//operating functions
cout << "Startup complete, begin operator mode?(y/n)\n";
bool mode = false;
string modeSelect;
int choice;
cin >> modeSelect;
//modeSelect = "n";
if (modeSelect == "y" || modeSelect == "Y") {
    cout << "Operator mode selected\n";
    mode = true;
}
while (mode == true) {

    cout << "Please select one of the following options:\n";
    cout << "1: restock item (Work in Progress)\n";
    cout << "2: change item (Work in Progress)\n";
    cout << "3: use wide item\n";
    cout << "4: change price\n";
    cout << "9: exit\n";
    cin >> choice;
    string opRow;
    bool valOpRow = true;
```

```
bool valOpCol = true;
string opCol;
int priceChange;
int opSelect = 0;
if (choice == 1) {
    cout << "This is a work in progress please choose a different number\n";
}
else if (choice == 2) {
    cout << "This is a work in progress please choose a different number\n";
}
else if (choice == 3) {
    cout << "Please select the row of the slot for the product that is wide.(A-E)\n";
    cin >> opRow;
    if (opRow == "a" || opRow == "A") {
        opSelect = 0;
    }
    else if (opRow == "b" || opRow == "B") {
        opSelect = 10;
    }
    else if (opRow == "c" || opRow == "C") {
        opSelect = 20;
    }
    else if (opRow == "d" || opRow == "D") {
        opSelect = 30;
    }
    else if (opRow == "e" || opRow == "E") {
        opSelect = 40;
    }
    else {
        cout << "Invalid row selected.\n";
```

```
            valOpRow = false;
        }
        if (valOpRow == true) {
            cout << "Please select column of item that is wide.(0 - 8. (9 is not a valid
selection))\n";
            cin >> opCol;
            if (opCol == "0") {
                opSelect = opSelect;
            }
            else if (opCol == "1") {
                opSelect = opSelect + 1;
            }
            else if (opCol == "2") {
                opSelect = opSelect + 2;
            }
            else if (opCol == "3") {
                opSelect = opSelect + 3;
            }
            else if (opCol == "4") {
                opSelect = opSelect + 4;
            }
            else if (opCol == "5") {
                opSelect = opSelect + 5;
            }
            else if (opCol == "6") {
                opSelect = opSelect + 6;
            }
            else if (opCol == "7") {
                opSelect = opSelect + 7;
            }
```

```
            else if (opCol == "8") {
                opSelect = opSelect + 8;
            }
            else {
                valOpCol = false;
            }
            if (valOpCol == true) {
                //in the future this would bring up a confirmation message
                //as well as display item that is being expanded
                arr[opSelect + 1].setter("wide", 0, 0);
                cout << "Selection " << opRow << opCol << " has been made wide using " <<
opRow << (opSelect % 10)+1 << ".\n";
            }
            else {
                //ITF this will check if item is already either occuppying the entire row or
operator has input an empty row. If yes this message will display.
                cout << "Invalid column selected.\n";
                valOpRow = false;
            }
        }
        cout << "Returning to operator menu.\n";
        valOpCol = true;
        valOpRow = true;
    }
    else if (choice == 4) {
        cout << "Please select the product row to change price.(A-E)\n";
        cin >> opRow;
        if (opRow == "a" || opRow == "A") {
            opSelect = 0;
        }
```

```
else if (opRow == "b" || opRow == "B") {
    opSelect = 10;
}
else if (opRow == "c" || opRow == "C") {
    opSelect = 20;
}
else if (opRow == "d" || opRow == "D") {
    opSelect = 30;
}
else if (opRow == "e" || opRow == "E") {
    opSelect = 40;
}
else {
    cout << "Invalid row selected.\n";
    valOpRow = false;
}
if (valOpRow == true) {
    cout << "Please select column of item to change price.(0 - 9)\n";
    cin >> opCol;
    if (opCol == "0") {
        opSelect = opSelect;
    }
    else if (opCol == "1") {
        opSelect = opSelect + 1;
    }
    else if (opCol == "2") {
        opSelect = opSelect + 2;
    }
    else if (opCol == "3") {
        opSelect = opSelect + 3;
```

```
}
else if (opCol == "4") {
    opSelect = opSelect + 4;
}
else if (opCol == "5") {
    opSelect = opSelect + 5;
}
else if (opCol == "6") {
    opSelect = opSelect + 6;
}
else if (opCol == "7") {
    opSelect = opSelect + 7;
}
else if (opCol == "8") {
    opSelect = opSelect + 8;
}
else if (opCol == "9") {
    opSelect = opSelect + 9;
}
else {
    valOpCol = false;
}
if (valOpCol == true) {
    if (arr[opSelect].getName() == "wide") {
        cout << "Cannot change price for extra slots of a wide item.\n";
        valOpRow = false;
    }
}
else {
```

```
                //ITF this will check if item is already either occuppying the entire row or
operator has input an empty row. If yes this message will display.
                cout << "Invalid column selected.\n";
                valOpRow = false;
            }
            if (valOpRow == true) {
                cout << "Please input the price (Do not include decimal. ex $1.25 input as
125)\n";
                cin >> priceChange;
                arr[opSelect].setPrice(priceChange);
                arr[opSelect].print();
            }
        }
        cout << "Returning to operator menu.\n";
        valOpCol = true;
        valOpRow = true;
    }
    else if (choice == 9) {
        cout << "Exiting operator mode. Starting customer mode.\n";
        mode = false;
    }
    else {
        cout << "Invalid input. Please input a valid option.\n";
    }
}
//end of operator mode


//begin customer mode
//gets highest value
```

```
    int highest = 0;
    for (int i = 0; i <= 49; i++) {
        if (arr[i].getPrice() > highest) {
            highest = arr[i].getPrice();
        }
    }
    //checks for change
    bool takeMoney = true;
    if (coins[3].getCount() >= 3 && coins[5].getCount() >= 4) { //has 3 quarters and 4
nickles
        if ((coins[2].getCount() * coins[2].getValue()) + (coins[3].getCount() *
coins[3].getValue()) + (coins[4].getCount() * coins[4].getValue()) + (coins[5].getCount() *
coins[5].getValue()) >= (495 + highest)) {
            takeMoney = true;                           //can take money
        }
        else {
            cout << "Cannot engage transaction. Not enough change in storage\n";
            takeMoney = false;
        }
    }
    else {
        cout << "Cannot engage transaction. Not enough quarters or nickels.\n";
            takeMoney = false;
    }
    bool dispensed = false;
    int money = 0;
    bool ohno = false;              //This is used for my own humor
    bool moreMoney = true;
    if (takeMoney == true) {
        //insert money function
```

```
        while (dispensed == false) {
            string cuRow;
            int cuSelect = 0;
            bool valCuRow = true;
            bool valCuCol = true;
            string cuCol;
            while (moreMoney == true) {
                cout << "Please input money, row selection or cancel. ( $5, $1, d1, d25, d19,
d5,(A-E) or cancel)\n";
                cin >> cuRow;
                if (cuRow == "a" || cuRow == "A") {
                    cuSelect = 0;
                    moreMoney = false;
                }
                else if (cuRow == "b" || cuRow == "B") {
                    cuSelect = 10;
                    moreMoney = false;
                }
                else if (cuRow == "c" || cuRow == "C") {
                    cuSelect = 20;
                    moreMoney = false;
                }
                else if (cuRow == "d" || cuRow == "D") {
                    cuSelect = 30;
                    moreMoney = false;
                }
                else if (cuRow == "e" || cuRow == "E") {
                    cuSelect = 40;
                    moreMoney = false;
                }
```

```
else if (cuRow == "Cancel" || cuRow == "cancel") {
    double decMoney = money;
    decMoney = decMoney / 100;
    cout << "Returning your money of $" << decMoney << "\n";
    moreMoney = false;
    int change = money;
    dispensed = true;
    while (change != 0) {
        //$5 and $1 are not used as they are not returned based on requirements
        if (change >= 100 && coins[2].getCount() > 0) {
            int coins2 = coins[2].getCount();
            coins2--;
            coins[2].setCount(coins2);
            change = change - 100;
            cout << "Dispensing $1 coin\n";
        }
        else if (change >= 25 && coins[3].getCount() > 0) {
            int coins3 = coins[3].getCount();
            coins3--;
            coins[3].setCount(coins3);
            change = change - 25;
            cout << "Dispensing d25\n";
        }
        else if (change >= 10 && coins[4].getCount() > 0) {
            int coins4 = coins[4].getCount();
            coins4--;
            coins[4].setCount(coins4);
            change = change - 10;
            cout << "Dispensing d10\n";
        }
```

```
        else if (change >= 5 && coins[5].getCount() > 0) {
            int coins5 = coins[5].getCount();
            coins5--;
            coins[5].setCount(coins5);
            change = change - 5;
            cout << "Dispensing d5\n";
        }
        else {
            cout << "Run out of change.\n";
            change = 0;
        }
    }
}
else if (money <= highest && cuRow == "$5") {
    money = money + 500;
}
else if (money <= highest && cuRow == "$1") {
    money = money + 100;
}
else if (money <= highest && cuRow == "d1") {
    money = money + 100;
}
else if (money <= highest && cuRow == "d25") {
    money = money + 25;
}
else if (money <= highest && cuRow == "d10") {
    money = money + 10;
}
else if (money <= highest && cuRow == "d5") {
    money = money + 5;
```

```
            }
            else {
                if (cuRow == "$5" || cuRow == "$1" || cuRow == "d1" || cuRow == "d25" ||
cuRow == "d10" || cuRow == "d5") {
                    cout << "Too much money inserted, returning your " << cuRow << ".\n";
                }
                else {
                    cout << "Invalid input.\n";
                    double decMoney = money;
                    decMoney = decMoney / 100;
                    cout << "Returning your money of $" << decMoney << "\n";
                    moreMoney = false;
                    valCuRow = false;
                    dispensed = true;
                }
            }
        }

        if (valCuRow == true && dispensed == false) {
            cout << "Please select column selection(0 - 9)\n";
            cin >> cuCol;
            if (cuCol == "0") {
                cuSelect = cuSelect;
            }
            else if (cuCol == "1") {
                cuSelect = cuSelect + 1;
            }
            else if (cuCol == "2") {
                cuSelect = cuSelect + 2;
            }
```

```
else if (cuCol == "3") {
    cuSelect = cuSelect + 3;
}
else if (cuCol == "4") {
    cuSelect = cuSelect + 4;
}
else if (cuCol == "5") {
    cuSelect = cuSelect + 5;
}
else if (cuCol == "6") {
    cuSelect = cuSelect + 6;
}
else if (cuCol == "7") {
    cuSelect = cuSelect + 7;
}
else if (cuCol == "8") {
    cuSelect = cuSelect + 8;
}
else if (cuCol == "9") {
    cuSelect = cuSelect + 9;
}
else {
    valCuCol = false;
}
if (valCuCol == true) {
    //in the future this would bring up a confirmation message
    if (money < arr[cuSelect].getPrice()) {
        cout << "Price of item is $" << arr[cuSelect].getDolPrice() << "\n";
        double funds = money;
        funds = funds / 100;
```

```
    cout << "Current funds are $" << funds << "\n";
    moreMoney = true;
}
else {
    if (arr[cuSelect].getName() == "wide") {
        cout << "Cannot select right side of large items.\n";
        moreMoney = true;
    }
    else if (arr[cuSelect].getCount() <= 0) {
        cout << "Selected empty slot, have a nice day\n";
        ohno = true;                    //don't worry about it
        dispensed = true;
    }
    else {
        cout << "Dispensing " << arr[cuSelect].getName() << "\n";
        int change = money - arr[cuSelect].getPrice();
        dispensed = true;
        while (change != 0) {
            if (change >= 100 && coins[2].getCount() > 0) {
                int coins2 = coins[2].getCount();
                coins2--;
                coins[2].setCount(coins2);
                change = change - 100;
                cout << "Dispensing $1 coin\n";
            }
            else if (change >= 25 && coins[3].getCount() > 0) {
                int coins3 = coins[3].getCount();
                coins3--;
                coins[3].setCount(coins3);
                change = change - 25;
```

```
                    cout << "Dispensing d25\n";
                }
                else if (change >= 10 && coins[4].getCount() > 0) {
                    int coins4 = coins[4].getCount();
                    coins4--;
                    coins[4].setCount(coins4);
                    change = change - 10;
                    cout << "Dispensing d10\n";
                }
                else if (change >= 5 && coins[5].getCount() > 0) {
                    int coins5 = coins[5].getCount();
                    coins5--;
                    coins[5].setCount(coins5);
                    change = change - 5;
                    cout << "Dispensing d5\n";
                }
                else {
                    cout << "Run out of change.\n";
                    change = 0;
                }
            }
        }
    }
}
else {
    //ITF this will check if item is already either occuppying the entire row or operator has input an empty row. If yes this message will display.
    cout << "Invalid column.\n";
    valCuCol = true;
    moreMoney = true;
```

```
            }
          }
        }
        cout << "exiting.\n";
    }
}
```