**Acknowledgements**

The researcher would like to express gratitude to his parents, friends and family for their support in completing this paper. Gratitude is extended to Mr. Jamal Kendal for his guidance to the researcher. The researcher also extends thanks to all the persons who have given data for analysis. Finally, all gratitude is given to God the Almighty who guided the researcher throughout the project.

**Problem Definition**

Matthew's Grocery store has been running for 3 years in business. Over the years, the store's profits have decreased in value. To deal with this complication, Matthew decides to change his existing design of the inventory for his stocks. He hopes this will keep track of his annual surplus.

The inventory was previously represented in two (2) main ways. The advantages and disadvantages of the used systems are shown below.

1. **Manually recording stock calculating surpluses.**

**Advantages**

* Even though computer solutions were quicker in entering information, it was easier to recall information from a manual journal ledger since you could flip to the pages you need, and even spread out books on a table.

**Disadvantages**

* It was a very slow and a tedious process.
* Compiling this information required careful location and rewriting of information, which was inefficient.
* It occupied more storage space.
* It was difficult to backup data.

1. **Implementing a database management system to store information.**

**Advantages**

* Manipulation of information was very quick and accurate.
* It occupied less storage space.
* Allowed for backup of data.

**Disadvantages**

* There was the risk of corrupted data.
* Data loss was problematic.
* Duplicates were generally formed which are not required.
* Required familiarity with the database management software.

The researcher has now come up with a digital solution for the problem which is the following:

* **Constructing a program that automatically records and calculates data.**

**Advantages**

* Allows for the automation of inventory-related tasks.
* Data can be updated real-time.
* Data is more secure.

**Disadvantages**

* It is difficult to use as the user needs beforehand information to read and write files from the program.
* *This type of limitation exists for any computer system, and this is solely caused by the lack of familiarity with the program. The problem solves itself as the user adapts to the program as time progresses. Therefore, this problem is solved by the repetitive usage of the program by the user.*
* Technology is required and power failures generally render the system useless temporarily.
* *By using systems such as power surges, uninterruptible power sources (UPSs) and redundant power supplies; this problem is curbed. Also, this enables hot-swapping, where the user can replace damaged components without turning off computer operations.*

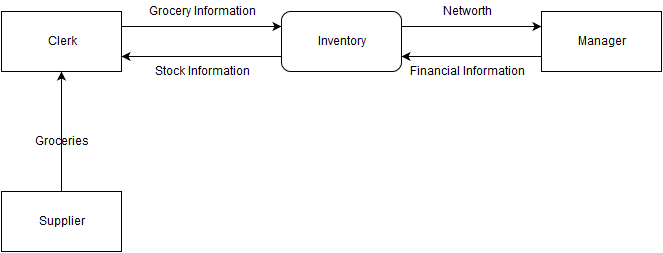
**Techniques of Analysis**

The researcher observed the store’s customer influx for 7 days or 168 hours and interviewed Matthew, the head of the store to gain data about the data management system his store used.

Questionnaires were also handed out to respondents. The questionnaires focused on finding which of the three (3) inventory systems are best suited for a company: manual, database or programmed. A questionnaire is a list of research questions designed to extract quantitative data. A questionnaire was used because large amounts of information can be collected from a large amount of people. It can be done in a short period of time and in a cost-effective way. The questionnaire consisted of ten (10) questions. There were eight (8) closed-ended questions and two (2) open-ended questions.

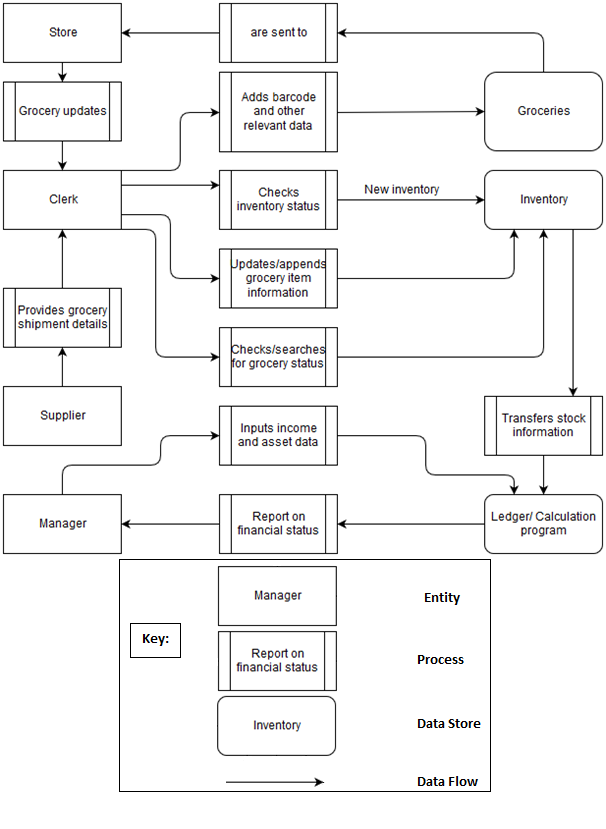
The total persons (population) who operate an inventory system in Georgetown are ~500 people. The sampling method was stratified random sampling: the researcher categorized people in the population by their years of experience in using inventory systems. Questionnaires were distributed to three (3) respondents from three (3) organizations that used inventory systems namely: Guyana Telephone & Telegraph Co Ltd – who used programmed, database and manual inventories, Starr Computers Inc – who used manual inventories, and Bounty Farm Ltd – who used programmed inventories. The questionnaires were handed out on the 9th of April, 2016. The questionnaires were collected on the 16th of April, 2016. Respondents of these companies were given a total of 7 days or 168 hours to complete the questionnaires. However, even after 7 days, some respondents provided very little information on open-based questions.

**Data Flow Diagrams (DFDs) – Context Level**



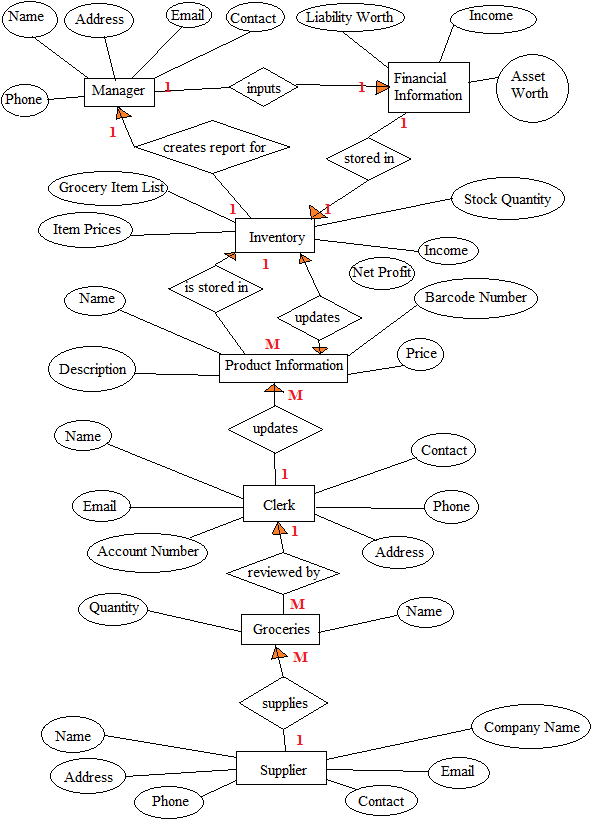
**Figure 1: Level-0 diagram showing data flow in Matthew’s Grocery Store’s workspace**

**Data Flow Diagrams (DFDs) – Level 1 Diagram**

****

**Figure 2: Level-1 diagram showing data flow in Matthew’s Grocery Store’s workspace**

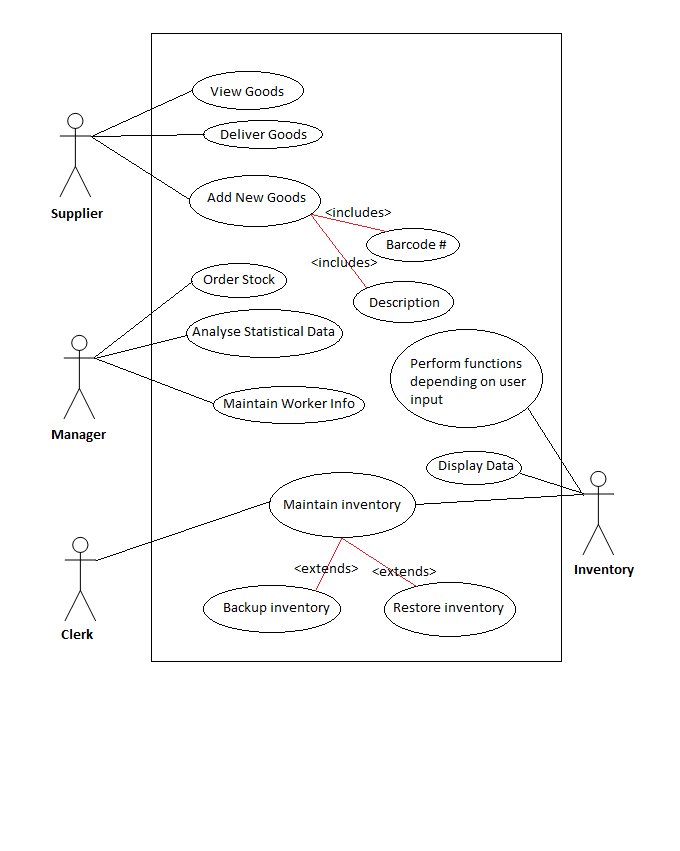
**Entity-Relationship Diagram (ERD)**



**Figure 3: Entity-relationship diagram of the various interactions in Mathew’s Grocery Store’s workspace**

**Functional Requirements**

* The system must accept barcode and other item information.
* The system must feed the item in the catalogue.
* The system must display stock information of all grocery items in the catalogue.
* The system must allow for changes to be made to data in the catalogue.
* The system must calculate company’s net worth.
* The system must allow for the searching for specific data in the catalogue.
* The system must display the number of items in the catalogue.

**Figure 4: Use case diagram showing the requirements of the various users**

**Non Functional Requirements**

* Accessibility

For people who experience disabilities, the system shall be accessible through the use of assistive technology that includes the assistive, adaptive and rehabilitative devices to input and read information from the inventory.

* Audit and control

The system shall be audited annually on performance, quality, energy usage, operations, and financing for stakeholders to effectively evaluate and improve the system whilst checking that personnel are not negligent or fraudulent.

* Availability

The system shall be available 120 hours per week (168 hours) with an availability of or 71.43% of the time.

* Backup

All inventory data shall be backed up to a private server exclusive to the business. The backup will be a combination of full and differential backup – where each differential backup saves the data that has changed since the last full backup.

* Capacity (power and storage)

The power supply shall provide adequate power to a RAID array of 3 1TB SSD hard disks. Each component of the system should not require more than 12 amperes of current to work efficiently.

* Certification

Each component used in building the system shall be certified according to the Energy Star certification scheme to ensure system is energy efficient (20 – 30 % less energy required for usage).

* Compliance

The system shall conform to the International Organization for Standardisation (ISO) 19600:2014

* Configuration management

The system shall allow the software source can be controlled by any worker of the company.

* Dependency on other parties

There is no dependency of the system on other third parties other than those who interact directly with the components of the system.

* Deployment

The inventory software shall be available as a running program on a dedicated server.

* Documentation

A user guide shall be made available for users to access from the manager of the company.

* Disaster recovery

Physical disasters shall be addressed within 168 hours (1 week) after the disaster has been encountered. [for software recovery details, see Recovery/ recoverability]

* Efficiency

The system shall use only the resources which are required by the function selected.

* Effectiveness (resulting performance in relation to effort)

The system shall use a preloaded cache to reduce processing time.

* Emotional factors

The system shall not be designed visually or computationally in any way which may be deemed as offensive to any person or party.

* Environmental protection

The system shall be given 48 hours to cool down every week to cool down to reduce the energy output to the surroundings. The components of the system shall conform to the MAS (Materials Analytical Services) Certified Green IEQ program, and the Greenguard Environmental Institute Indoor Air Quality program.

* Escrow

The inventory program shall be deposited to a third party escrow agent to ensure continued maintenance of code even if the company itself falls to bankruptcy or fails to maintain and update the software.

* Extensibility

The system shall allow for changes to be made to source code to add features, and carry-forward customizations for next major version upgrades.

* Failure management

The manager shall be held responsible for restoring the inventory within 168 hours should a fail be encountered.

* Fault tolerance

The system shall be monitored every 3 hours to ensure it is operative.

* Licensing

The inventory shall be licensed with the All Rights Reserved copyright law.

* Interoperability

The system shall work in conjunction with the barcode and POS systems.

* Maintainability

The system shall use redundant power supplies so that the loss of power is prevented from affecting components. Damaged units shall be replaceable without turning off the system (hot swapping).

* Modifiability

Repairs shall be easily made as the system runs on C code.

* Performance / response time (performance engineering)

Each function given by the user shall only require a maximum of 30 seconds to process.

* Platform compatibility

The inventory software shall be compatible with all versions of Windows operating systems.

* Portability

The data shall be stored on a dedicated system in a confined work environment. A copy of this data shall be available for editing to the POS terminals.

* Recovery / recoverability (e.g. mean time to recovery - MTTR)

Restoring the entire system shall require starting from the most recent full backup and then applying just the last differential backup since the last full backup. This will be done within 48 hours. [for backup details, see Backup]

* Safety or Factor of safety

Common activities and precautions shall be taken to reduce human health risk such as root cause analysis, physical examinations, stress testing and visual examinations.

* Scalability

The system shall allow the data scope to be increased as required by the operator.

* Security

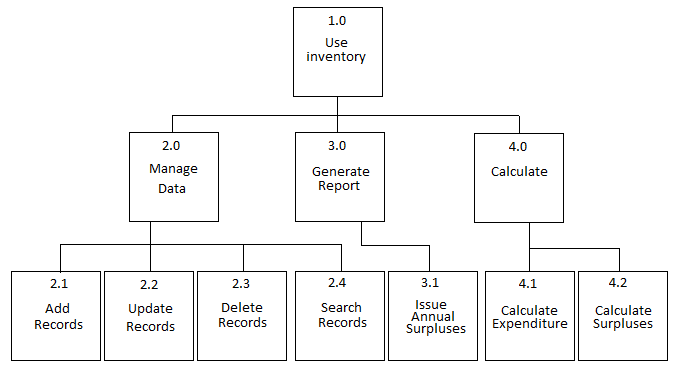
All data and programs shall be encrypted, only available to workers of the company.

* Usability

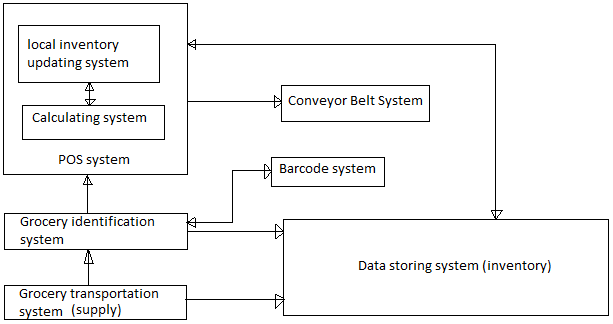
The system shall be simple enough for any person without previous experience of computers to operate. The user guide will aid in understanding the system.

**Design Specification – System Structuring**

System structuring is concerned with decomposing the system into interacting sub-systems.

****

**Figure 6: Hierarchical Input-Process-Output Chart of the Data Flow Functions**



**Figure 7: Architectural Design of System Structure (Overview)**

The system can split into sub-systems. These are represented in Figure 4. They are:

1. The grocery transportation system
2. Grocery identification system
3. Barcode system
4. POS system
5. Conveyor Belt system
6. Data storing system

At the grocery transportation system, the supplier gives at a set time. The data storing system or inventory is now initiated to load previous grocery data or create new grocery data. The grocery is then sent to the grocery identification system.

At the grocery identification system, groceries are sorted and then labelled with specific grocery data. This data is then assigned to a barcode.

The barcode system handles the creation and assigning of barcodes to groceries. The barcode data is sent back to the grocery identification system.

At the grocery identification system, the barcode data is labelled additionally. The barcodes are now sent to the calculating and inventory. The inventory updates its information here.

The calculating system of the POS system then calculates grocery stock information and then stores the data in the local inventory using the local inventory updating system.

The local inventory system can be used as extension of the inventory when there is a network, and a standalone system that updates the inventory at set times when there is no network available.

The calculating system then gives a command to the conveyor belt system to transport the grocery item(s) to the store.

The inventory can also be directly updated with information with the approval of the manager.

Figure 1 shows the representation of the system using a repository model where all data is stored in a main repository or data store upon which most (if not all) system interact.

The advantages of using such a system are that:

* It is an efficient way to share large amounts of data.
* The sub-systems do not need to be concerned with storing the actual data but only manipulate it.

However, it also has disadvantages which include:

* The majority of the sub-systems must agree on the repository as the data store: inevitably a compromise.
* It is difficult to distribute efficiently, i.e. the repository may or may not work for other scenarios other than the one programmed for – it is less versatile.

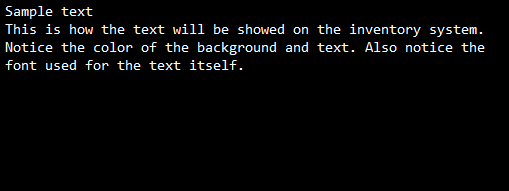
**Design Specification – User Interface Design**

Because the system has to be running 24 hours a day, it must be fast and easy to wake up from rest. For this reason, the researcher chose to use default system fonts as the typeface and a command line display that is always turned on.

Because no data must be deleted unless commanded by the user, the system must have a dedicated system which gives the program all authority over data.

Because the system must start up in a maximum of 6 seconds, the researcher used no background color, and left it as the default black and white text to start up more quickly which loading font data from libraries. The font used was the default system font which does not require to be retrieved from a library.

Based on all these constraints, a command-line user interface was used.



**Figure 8: Interface Design of the System**

The design includes a barcode sensor connected to a data management system. The laser lights act as indicators. This arrangement served to send messages regarding the grocery item (the barcode) to the data management system. The active device throughout this project is a barcode scanner. It is used to input data into the inventory, even though data passes through several systems before reaching the inventory. This design makes use of a repository system and its sub-systems. The design has many potential applications including storing, editing, and destroying grocery data.

Procedures for design

* Within the circuit was a barcode controller integrated circuit whose output was connected to POS system. To control this system, the circuit was programmed following the pseudocode shown in the Algorithm Design. When implementing this circuit, the function Locate was first called to locate previous inventory records. This is to load previous data so that the barcode scanner has data to operate upon, i.e. if there are no groceries, the stocks become negative.
* Afterwards a variable choice is inputted by the user that stores the user’s choice on operations needs to be undertaken by the problem.
* Next, a prompt was shown to verify if the user wants to continue the program.

Assessment of design

* To test the operation of the function Locate (), previous barcode data was manually added to a file and placed in a root folder. During this, it was verified that the developed hardware and software of the lab for this part was functioning smoothly.
* Whilst testing the variable choice, a lot of complication insisted, including wrong data type or no data at all. To debug this error, a character array of size 100 had to be used instead of a character. There was then a check to verify the input size. If the array size was greater than one, then and error message was displayed on the screen. Also, conditions were used to account for the various options, with an else command used to terminate any other false input.
* Initially, a recursion was used at the end of main (). However, this created a major complication in exiting the program as directly closing the program would be unsafe. Therefore, the continue prompt was put in place.

All in all, the barcode scanner worked satisfactorily in conjunction with the program (inventory) to bring about a working data management system.**Design Specification – Report Design**

**Figure 9: Bar Graph Showing the Popularity of the Various Inventory Types**

**Figure 10: Pie chart showing Limitations Proportions of the Various Inventory Types as indicated by Respondents**

This report presents a data management system that makes use of an inventory. It is used by Matthew’s Grocery Store. This system is being implemented because there is a problem with the grocery store where there is too much paperwork being done that is time-consuming and frustrating for the staff of the store. This is evident when the researcher observed the store for a period of seven (7) days or 168 hours and noticed a large influx of customers in the store. This observation was proved using the questionnaires. Various graphical presentations are provided above that support this proof. The manager of the store has realized that their numbers of customers are greater now than a decade ago. Therefore, after drawing conclusions from the data collected, a programmed data management system was hypothesized.

**Design Specification – Algorithm Design (Narrative)**

**START**

Set “MGSI” as the default text for barcode testing.

**main()**

**begin**

Prompt the user to choose between: appending/ creating inventory, displaying inventory, deleting inventory, searching inventory, calculating stocks or using the income calculator.

If user has inputted a string instead of one character, display “error” onto the screen.

Call the functions **Append()**, **Display()**, **Delete()**, **Search()**, **Stocks()** or **IncomeCalculator()** based on the user’s input.

If the user has inputted some other character, display “error” onto the screen.

Prompt the user to continue the program.

If user has inputted a string instead of one character, exit the program immediately.

If the user enters “y” or “Y”, clear the screen and loop back to the start of **main()**.

**end**

**Locate()**

**begin**

Check if file “database.dat” exists.

If file does not exist, display “There is no file.” onto the screen.

Otherwise, display “Successfully located file.” onto the screen.

**end**

**Append()**

**begin**

Prompt user to enter an 8-digit barcode starting with the 4 characters “MGSI”.

If the inputted string is not 8 characters long or the 4 characters are not present, display “Wrong input.” and terminate the program.

Prompt the user to enter the product name and its price.

If the price is inputted as a character or has a negative value, display “Wrong input.” and terminate the program.

Store the 3 inputs in a file called “database.dat”.

Call the function **DataStore()** with price as function parameter.

**end**

**Display()**

**begin**

If “database.dat” does not exist, display “There is no file.” on the screen.

Otherwise, store the information in “database.dat” in a buffer display.output.

Recall this information and free the buffer.

**end**

**Delete()**

**begin**

Delete the files “database.dat” and “price.dat”.

Display “Successfully deleted file(s).” onto the screen.

**end**

**Search()**

**begin**

If “database.dat” does not exist, display “There is no file.” on the screen.

Otherwise, prompt user to enter an 8-digit barcode starting with the 4 characters “MGSI”.

If the inputted string is not 8 characters long or the 4 characters are not present, display “Wrong input.” and terminate the program.

Search each line for any instance of the barcode by comparing the strings.

If any record is found, display “Found record.” and that record.

Otherwise, display “Not found.”

**end**

**DataStore(price)**

**begin**

If “price.dat” does not exist, set the price counter to 1.

Store “price” and the inventory counter in “price.dat”.

Otherwise, recall previous price and add it to the current price.

Recall the previous counter information and increment it.

Overwrite these values in “price.dat”.

**end**

**Stocks()**

**begin**

Prompt the user to enter current budget.

If the input is character(s), display “Wrong input.” and terminate the program.

Prompt the user to enter total income.

If the input is character(s), display “Wrong input.” and terminate the program.

Recall the price and the inventory counter data from “price.dat”.

Display number of items in inventory and the total inventory price.

Calculate and display the Total Budget by adding the budget and the total inventory price.

Calculate and display the Old Budget by subtracting the income from the budget.

**end**

**IncomeCalculator()**

**begin**

Prompt the user to enter the items sold.

If input is character(s), display “Wrong input.” and terminate the program.

Otherwise, prompt the user to sequentially enter the prices for all items.

Add all the prices and display the value on the screen.

**end**

**STOP**

**Design Specification - Algorithm Design (Pseudocode)**

**START**

Declare structure inventory.

Declare pointer1, pointer2 as file pointer(s) in inventory.

Declare tempint, tempint2 as integers in inventory.

Declare tempfloat, tempfloat2 as floats in inventory.

Declare output as character pointer in inventory.

Declare barcode as character array of length 15 in inventory.

Declare label as character array of length 50 in inventory.

Declare price as float in inventory.

Declare tempchar as character array of length 100 in inventory.

Declare size as a size holder in inventory.

Declare Locate(), Append(), Display(), Delete(), Search(), Stocks(), DataStore(float price), IncomeCalculator as void functions.

char barcodetest[ ] 🡨 “MGSI”.

Declare choice as character array of length 100.

**main()**

**begin**

Writeln(“MATTHEW’S GROCERY STORE INFORMATION”)

**Locate()**

**Writeln(“INVENTORY OPTIONS”)**

**Writeln(“a = Append/Create inventory”)**

**Writeln(“b = Display inventory”)**

**Writeln(“c = Delete inventory”)**

**Writeln(“d = Search inventory”)**

**Writeln(“e = Calculate stocks”)**

**Writeln(“f = Income Calculator”)**

**Write(“Choice: ”)**

**Readln(choice)**

**int error 🡸 stringlengthof(choice)**

**if(error>1) then**

**Writeln(“Error parsing data. Check input.”)**

**StopProgram**

**else if(choice[0] = ‘a’) then**

**Append()**

**else if(choice[0] = ‘b’) then**

**Display()**

**else if(choice[0] = ‘c’) then**

**Delete()**

**else if(choice[0] = ‘d’) then**

**Search()**

**else if(choice[0] = ‘e’) then**

**Stocks()**

**else if(choice[0] = ‘f’) then**

**IncomeCalculator()**

**else**

**Writeln(“Error parsing data. Check input.”)**

**StopProgram**

**endif**

**Write(“Continue(y):”)**

**Read(choice)**

**error 🡸 stringlengthof(choice)**

**if(error>1) then**

**Writeln(“Wrong input.”)**

**StopProgram**

**endif**

**if(choice[0] = ‘y’ or choice[0] = ‘Y’) then**

**ClearScreen**

**main()**

**endif**

**end**

**Locate()**

**begin**

Declare instance “locate” of structure inventory.

locate.pointer 🡸 database.dat in read binary mode

if(locate.pointer==null) then

Writeln(“There is no file.”)

else

Writeln(“Successfully located file.”)

endif

closefile(locate.pointer)

**end**

**Append()**

**begin**

Declare instance “append” of structure inventory.

Write(“8-Digit Barcode#(MGSI----): ”)

Readln(append.barcode)

append.tempint 🡸 stringlengthof(append.barcode)

if((append.barcode & barcodetest have the same 4 chars)or(append.tempint!=8)) then

Writeln(“Wrong input.”)

StopProgram

endif

**Write(“Name: ”)**

**Readln(append.label)**

**Write(“Price: $”)**

**Readln(append.price)**

**if(append.price=0 or append.price<0) then**

**Writeln(“Wrong input.”)**

**StopProgram**

**endif**

**append.pointer 🡸 database.dat in append binary mode**

**WriteFile(append.barcode)**

**Write(“Copied [”, append.barcode, “]”)**

**WriteFile(append.price)**

**Write(“, [”, append.price, “]”)**

**WriteFile(append.label)**

**Write(“ and [”, append.label, “]”)**

**closefile(append.pointer)**

**DataStore(append.price)**

**end**

**Display()**

**begin**

Declare instance “display” of structure inventory.

display.pointer 🡸 database.dat in read binary mode

if(display.pointer=NULL) then

Writeln(“There is no file.”)

else

display.size 🡸 sizeof(display.pointer)

display.output[display.size] 🡸 0

Writeln(“Format: ( Barcode Number / Price($) / Label”)

Writeln(display.output)

endif

closefile(display.pointer)

**end**

**Delete()**

**begin**

delete(database.dat)

delete(price.dat)

Writeln(“Successfully deleted file(s).”)

**end**

**Search()**

**begin**

Declare instance “search” of structure inventory.

search.pointer 🡸 database.dat in read binary mode

if(search.pointer=NULL) then

Writeln(“There is no file.”)

else

Write(“8-Digit Barcode#(MGSI----):”)

Readln(search.barcode)

search.tempint 🡸 stringlengthof(search.barcode)

if((search.barcode, barcodetest has same 1st 4 chars)or(search.temp!=8) then

Writeln(“Wrong input.”)

StopProgram

endif

while(endof[search.pointer]is not reached) do

search.tempchar 🡸 ReadLine(search.pointer)

search.output 🡸 stringsearch(search.tempchar, search.barcode)

if(search.output=NULL) then

Writeln(“Not found.”)

else

Writeln(“Found record.”)

Writeln(search.tempchar);

return

endif

endwhile

endif

closefile(search.pointer)

**end**

**DataStore(float price)**

**begin**

Declare instance “data” of structure inventory.

data.pointer 🡸 price.dat in read binary mode

if(data.pointer=NULL) then

closefile(data.pointer)

data.pointer 🡸 price.dat in write binary mode

WriteFile(price, 1)

closefile(data.pointer)

else

ReadFile(data.tempfloat, data.tempint)

price 🡸 price + data.tempfloat

data.tempint 🡸 data.tempint + 1

closefile(data.pointer)

data.pointer 🡸 price.dat in write binary mode

WriteFile(price, data.tempint)

closefile(data.pointer)

endif

**end**

**Stocks()**

**begin**

Declare instance “stocks” of structure inventory

Declare budget, income as floats

Write(“Current Budget: $”)

Readln(budget)

**if(budget=0) then**

**Writeln(“Wrong input.”)**

**StopProgram**

**endif**

**Write(“Total Income: $”)**

**Readln(income)**

**if(income=0) then**

**Writeln(“Wrong input.”)**

**StopProgram**

**stocks.pointer 🡸 price.dat in read binary mode**

**ReadFile(stocks.price, stocks.tempint)**

**Writeln(“Total items: ”, stocks.tempint)**

**Writeln(“Total Inventory Price: ”, stocks.price)**

**closefile(stocks.pointer)**

**printf(“Total Budget: ”, (budget+stocks.price))**

**printf(“Old Budget: ”, (budget-income))**

**end**

**IncomeCalculator()**

**begin**

Declare i, itemssold as integers

Declare current\_price, total\_price as floats

total\_price 🡸 0

Write(“Items Sold: ”)

Readln(itemssold)

if(itemssold=0) then

Writeln(“Wrong input.”)

StopProgram

endif

itemssold 🡸 itemssold + 1

for(i = 1 to itemssolds) do

Write(“Price of item ”, i, “: $”)

Readln(current\_price)

if(current\_price=0) then

Writeln(“Wrong input.”)

StopProgram

endif

total\_price 🡸 total\_price + current\_price

Writeln(“”)

**endfor**

**Writeln(“Total Income: $”, total\_price)**

**end**

**STOP**

**Coding and Testing – Code**

#include <conio.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

**struct** inventory { *// Inventory data and file locations are grouped together in the structure.*

FILE \*pointer, \*pointer2;

**int** tempint, tempint2;

**float** tempfloat, tempfloat2;

**char** \*output;

**char** barcode[15];

**char** label[50];

**float** price;­­

**char** tempchar[100];

size\_t size;

};

**void** Locate(); *// The function prototypes*

**void** Append();

**void** Display();

**void** Delete();

**void** Search();

**void** Stocks();

**void** DataStore(**float** price);

**void** IncomeCalculator();

**char** barcodetest[] = "MGSI";

**char** choice[100];

**int** main() {

puts("MATTHEW'S GROCERY STORE INVENTORY\n");

Locate(); // Loads file

puts("\nINVENTORY OPTIONS\n"); *// Prompts the user to input their choice.*

puts("a = Append/Create inventory\n");

puts("b = Display inventory\n");

puts("c = Delete inventory\n");

puts("d = Search inventory\n");

puts("e = Calculate stocks\n");

puts("f = Income Calculator\n");

printf("Choice: ");

scanf("%s", choice);

printf("\n");

**int** error = strlen(choice);

**if**(error>1) { *// Checks if userinput is greater than 1 character and if input IS a character.*

puts("Error parsing data. Check input.\n");

exit(0);

} **else if**(choice[0] == 'a') { *// Takes the first character of string as input.*

Append();

} **else if**(choice[0] == 'b') {

Display();

} **else if**(choice[0] == 'c') {

Delete();

} **else if**(choice[0] == 'd') {

Search();

} **else if**(choice[0] == 'e') {

Stocks();

} **else if**(choice[0] == 'f') {

IncomeCalculator();

} **else** {

puts("Error parsing data. Check input.\n");

exit(0);

}

printf("Continue(y)?:");

scanf("%s", choice);

error = strlen(choice);

**if**(error>1) {

puts("Wrong input.");

exit(0);

}

**if**(choice[0] == 'y'|| choice[0] == 'Y') { *// Loops the program.*

system("cls");

main();

}

**return** 0;

}

**void** Locate() {

**struct** inventory locate;

**if**((locate.pointer=fopen("database.dat", "rb"))==NULL) *// Checks if file exists already.*

puts("There is no file.");

**else**

puts("Successfully located file.");

fclose(locate.pointer);

**return**;

}

**void** Append() {

**struct** inventory append;

printf("8-Digit Barcode#(MGSI----): "); *// Prompts the user to input data.*

scanf("%s", append.barcode);

printf("\n");

append.tempint = strlen(append.barcode);

**if**((strncmp(append.barcode, barcodetest, 4))||(append.tempint != 8)) {

puts("Wrong input.");

exit(0);

}

getchar(); *// Consumes the newline character that is stored in the input buffer.*

printf("Name: ");

gets(append.label);

printf("\n");

printf("Price: $");

scanf("%f", &append.price);

printf("\n");

**if**(append.price==0||append.price<0) {

puts("Wrong input.");

exit(0);

}

append.pointer = fopen("database.dat", "ab"); // Writes user input to file.

fprintf(append.pointer, "%s", append.barcode);

printf("Copied [%s]", append.barcode);

fprintf(append.pointer, " / %.2f", append.price);

printf(", [%.2f]", append.price);

fprintf(append.pointer, " / %s\n", append.label);

printf(" and [%s].", append.label);

fclose(append.pointer);

printf("\n\n");

DataStore(append.price); *// Calls the function DataStore() to store prices for stock calculation.*

**return**;

}

**void** Display() {

**struct** inventory display;

**if**((display.pointer = fopen("database.dat", "rb"))==NULL) {

puts("There is no file.\n");

} **else** {

fseek(display.pointer, 0L, SEEK\_END); *// Takes the seeker to the end of the file.*

display.tempint = ftell(display.pointer); *// Stores the position of the seeker in a pointer.*

fseek(display.pointer, 0L, SEEK\_SET);  *// Brings the seeker back to start of the program.*

display.output = malloc(display.tempint+1);

display.size = fread(display.output, 1, display.tempint, display.pointer);

display.output[display.size] = 0;

printf("Format: ( Barcode Number / Price($) / Label )\n%s\n", display.output); *// Prints the entire file onto the screen.*

free(display.output); *// Frees the memory allocated for the buffer.*

}

fclose(display.pointer);

**return**;

}

**void** Delete() {

remove("database.dat");

remove("price.dat"); *// Deletes the two files used in the program.*

puts("Successfully deleted file(s).\n");

**return**;

}

**void** Search() {

**struct** inventory search;

**if**((search.pointer = fopen("database.dat", "rb"))==NULL) {

puts("There is no file.\n");

} **else** {

getchar();

printf("8-Digit Barcode#(MGSI----):");

gets(search.barcode);

search.tempint = strlen(search.barcode);

**if**((strncmp(search.barcode, barcodetest, 4))||(search.tempint != 8)) { // *Checks for MGSI or string length that is not 8 characters.*

puts("Wrong input.");

exit(0);

}

**while**(!feof(search.pointer)) {

fgets(search.tempchar, 100, search.pointer);

search.output = strstr(search.tempchar, search.barcode);

**if**(search.output == NULL) { *// Loops until the record is found.*

puts("Not found.");

} **else** {

puts("Found record.");

puts(search.tempchar);

**return**;

}

}

}

fclose(search.pointer);

**return**;

}

**void** DataStore(**float** price) {

**struct** inventory data;

**if**((data.pointer = fopen("price.dat", "rb"))==NULL) {

fclose(data.pointer);

data.pointer = fopen("price.dat", "wb");

fprintf(data.pointer, "%.2f %d", price, 1); *// Writes initial input to create the file.*

fclose(data.pointer);

} **else** {

fscanf(data.pointer, "%f %d", &data.tempfloat, &data.tempint); *// Recalls stock information.*

price += data.tempfloat;

data.tempint++;

fclose(data.pointer);

data.pointer = fopen("price.dat", "wb");

fprintf(data.pointer, "%.2f %d", price, data.tempint);

fclose(data.pointer);

}

**return**;

}

**void** Stocks() { *// Calculates the budget information needed.*

**struct** inventory stocks;

**float** budget, income;

printf("Current Budget: $");

scanf("%f", &budget);

**if**(budget==0) {

puts("Wrong input.");

exit(0);

}

printf("\n");

printf("Total Income: $");

scanf("%f", &income);

**if**(income==0) {

puts("Wrong input.");

exit(0);

}

printf("\n");

stocks.pointer = fopen("price.dat", "rb");

fscanf(stocks.pointer, "%f %d", &stocks.price, &stocks.tempint);

printf("Total items: %d\n", stocks.tempint);

printf("Total Inventory Price: $%.2f\n", stocks.price);

fclose(stocks.pointer);

printf("Total Budget: $%.2f\n", budget+=stocks.price);

printf("Old Budget: $%.2f\n\n", budget-=income);

**return**;

}

**void** IncomeCalculator() { *// Calculates income from user input.*

**int** i, itemssold;

**float** current\_price, total\_price = 0;

printf("Items Sold: ");

scanf("%i", &itemssold);

**if**(itemssold==0) {

puts("Wrong input.");

exit(0);

}

printf("\n");

itemssold++;

**for**(i=1; i<itemssold; i++) { *// Loops for each item sold.*

printf("Price of item %i: $", i);

scanf("%f", &current\_price);

**if**(current\_price==0) {

puts("Wrong input.");

exit(0);

}

total\_price+=current\_price;

printf("\n");

}

printf("Total Income: $%.2f", total\_price);

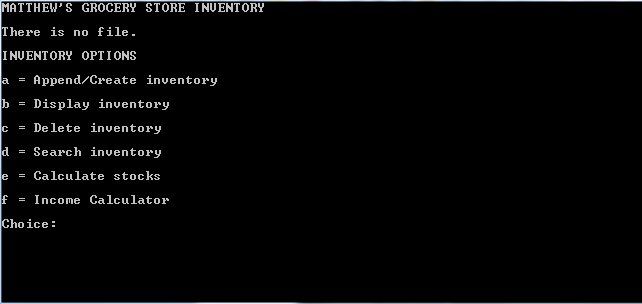
puts("\n");

**return**;

}

**Coding and Testing – Test Plan (Standard Input)**

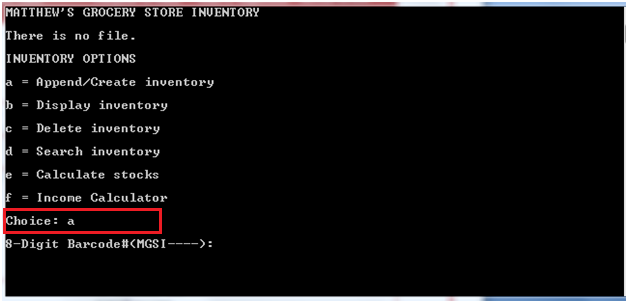
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Variable Name | Test Type | Test Data | Purpose of Data | User’s Expected Result | Actual Result |
| choice[0] | string | y, Y | To continue program | Clears the screen and continues the program. If wrong input is inputted exits the program. Otherwise returns 0 and exits the program. | Clears the screen and continues the program. If wrong input is inputted exits the program. Otherwise returns 0 and exits the program. |



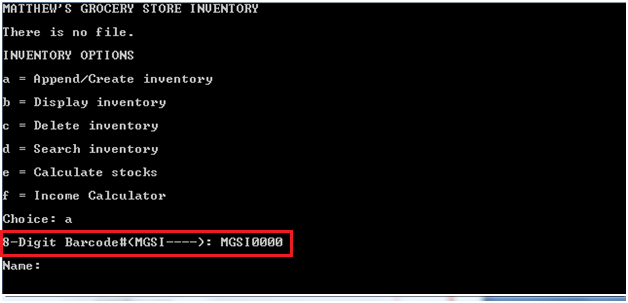
Prompting user to input choice (either “a” “b” “c” “d” “e” or “f” ).

This is also the screen after inputting ‘y’ or ‘Y’ to continue in choice[0].

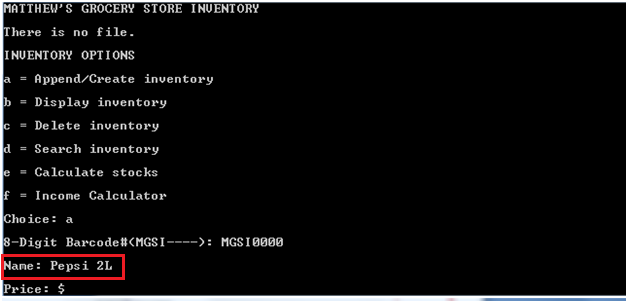
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Variable Name | Test Type | Test Data | Purpose of Data | User’s Expected Result | Actual Result |
| choice[0] | string | a | To append data to inventory | The program prompts the user to input data to append into the file. | The program prompts the user to input data to append into the file. |
| append.barcode | string | MGSI0000 | To store product barcode | The product stores the barcode number and asks to input name and price or exits the program if wrong input is provided. | The product stores the barcode number and asks to input name and price or exits the program if wrong input is provided. |
| append.label | string | Pepsi 2L | To store product name | The program stores the product name. | The program stores the product name. |
| append.price | float | 356.65 | To store product price | The program stores the price information or exits the program if wrong input is provided. | The program stores the price information or exits the program if wrong input is provided. |



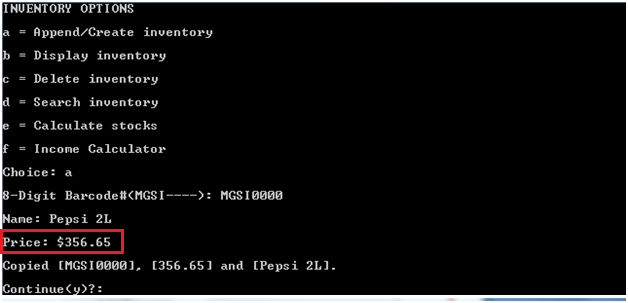
Prompting user to enter information to be appended.



Stores the product barcode in the input buffer.



Stores the product name in the input buffer.

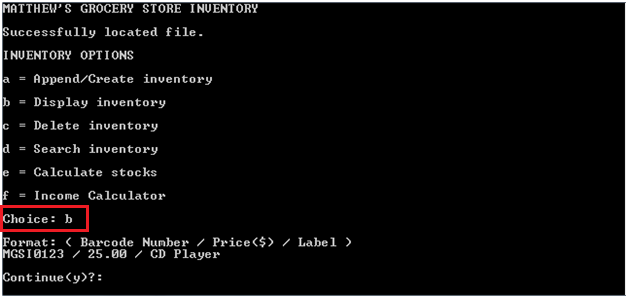


Stores the product price in the input buffer.

Also saves the product barcode, name and price in file and provides feedback.

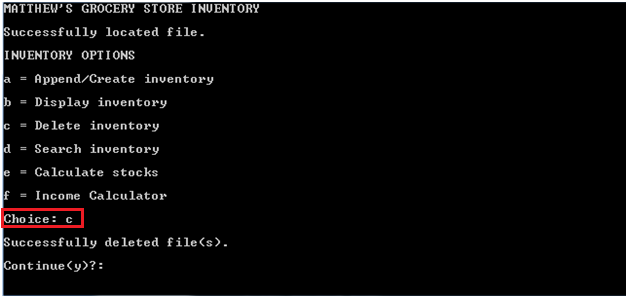
Asks the user if he/she wants to continue the program.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Variable Name | Test Type | Test Data | Purpose of Data | User’s Expected Result | Actual Result |
| choice[0] | string | b | To display inventory data | The program either displays that there is no file or displays the inventory. | The program either displays that there is no file or displays the inventory. |



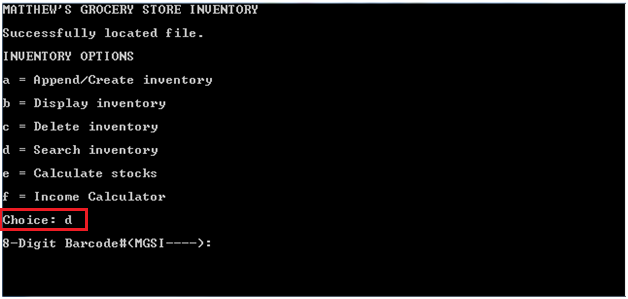
Display of inventory and prompts user to continue the program.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Variable Name | Test Type | Test Data | Purpose of Data | User’s Expected Result | Actual Result |
| choice[0] | string | c | To delete the inventory | The program deletes the inventory files database.dat and price.dat and provides feedback. | The program deletes the inventory files database.dat and price.dat and provides feedback. |

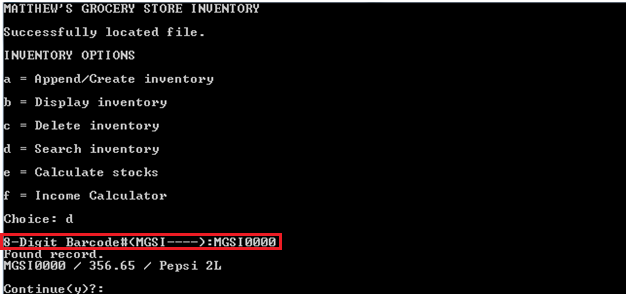


Shows feedback that files are deleted.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Variable Name | Test Type | Test Data | Purpose of Data | User’s Expected Result | Actual Result |
| choice[0] | string | d | To search for a record in the inventory | The program prompts the user to input the barcode number of the record to be searched for. | The program prompts the user to input the barcode number of the record to be searched for |
| search.barcode | string | MGSI0000 | To search for product record | The program searches inventory for record and displays it. If not found or wrong input, provides feedback. | The program searches inventory for record and displays it. If not found or wrong input, provides feedback. |

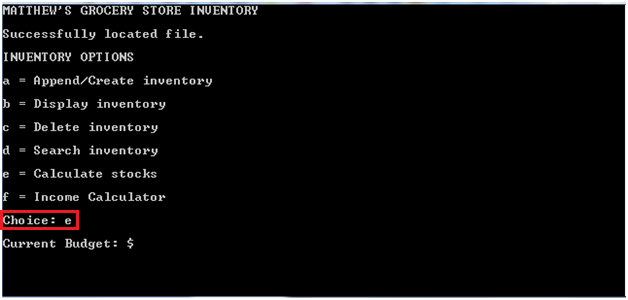


Prompts the user to input the barcode of product to search for.

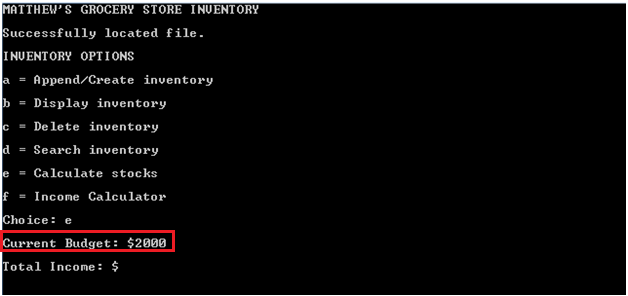


Searches the inventory for record using barcode # and outputs the found record data.

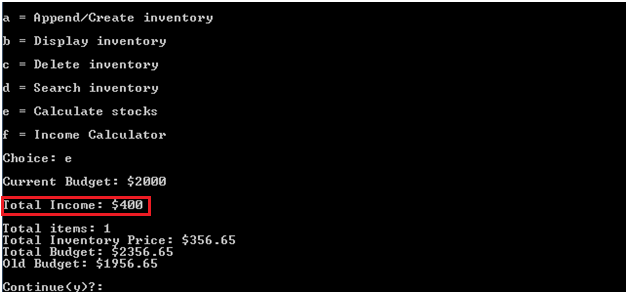
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Variable Name | Test Type | Test Data | Purpose of Data | User’s Expected Result | Actual Result |
| choice[0] | string | e | To calculate the overall stock and budget information | The program prompts the user to input stock information (i.e. Current Budget). | The program prompts the user to input stock information (i.e. Current Budget). |
| budget | float | 2000 | To store budget value to calculate stocks | The program stores the budget price. If characters are inputted, exits the program. | The program stores the budget price. If characters are inputted, exits the program. |
| income | float | 400 | To store total income value to calculate stocks | The program stores the total income value and calculates stock information. | The program stores the total income value and calculates stock information. |



Prompts user to input stock information.



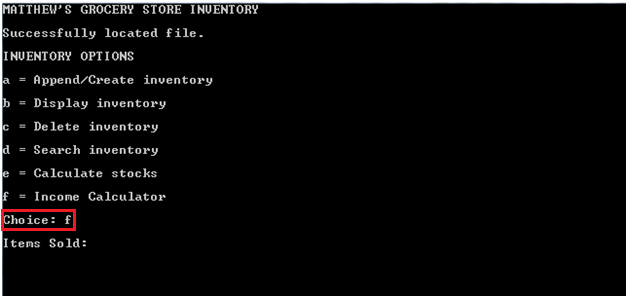
Stores the budget value in the input buffer.



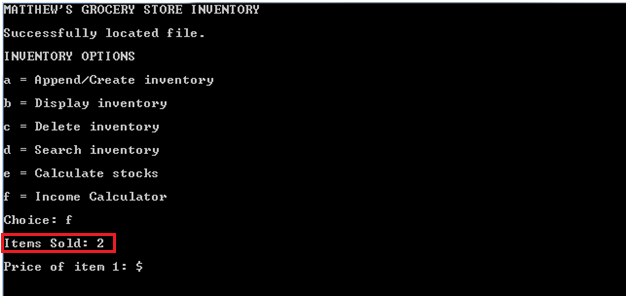
Stores the total income in the input buffer.

Calculates and displays stock information.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Variable Name | Test Type | Test Data | Purpose of Data | User’s Expected Result | Actual Result |
| choice[0] | string | f | To calculate the overall income | The program prompts the user to input income values sequentially. | The program prompts the user to input income values sequentially. |
| itemssold | integer | 2 | To store item number to calculate total income | Stores the number of items. If number of items are 0 or characters, exits the program. | Stores the number of items. If number of items are 0 or characters, exits the program. |
| current\_price | float | 250, 150 | To store prices of items to calculate total income. | Stores the income values and displays the total income. If a character or 0 income is inputted, exits the program. | Stores the income values and displays the total income. If a character or 0 income is inputted, exits the program. |

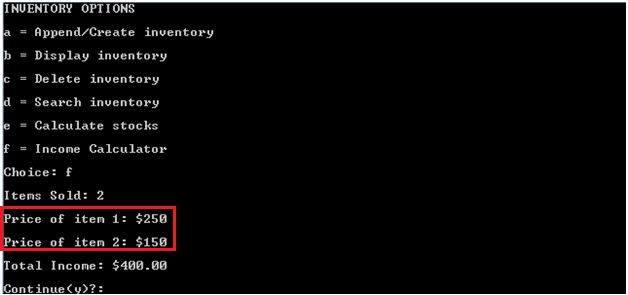


Prompts user to input income information.



Stores the number of items in the input buffer.

Prompts the user to enter the prices of items sequentially.



Stores the income values in the input buffer.

Calculates and displays the total income.

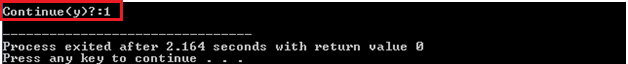
Prompts the user to continue.

**Coding and Testing – Test Plan (Extreme Input)**

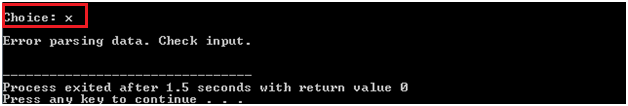
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Variable Name | Test Type | Test Data | Purpose of Data | User’s Expected Result | Actual Result |
| choice[0] | string | wjw, y.1, 1 | To continue program | Clears the screen and continues the program. If wrong input is inputted exits the program. Otherwise returns 0 and exits the program. | Clears the screen and continues the program. If wrong input is inputted exits the program. Otherwise returns 0 and exits the program. |

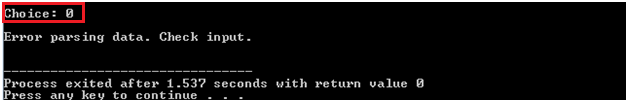




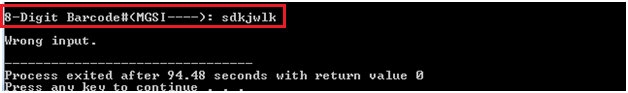


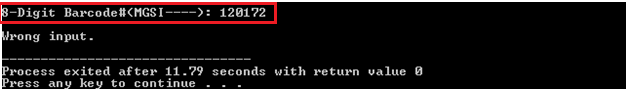
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Variable Name | Test Type | Test Data | Purpose of Data | User’s Expected Result | Actual Result |
| choice[0] | string | x, 0, a.2 | User choice | The user can enter a, b, c, d, e or f to utilize the inventory. Otherwise, exits the program. | The user can enter a, b, c, d, e or f to utilize the inventory. Otherwise, exits the program. |

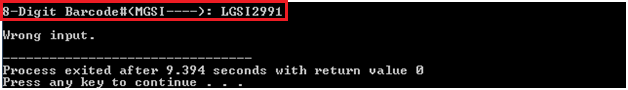


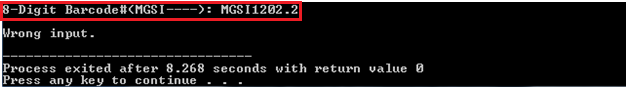


|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Variable Name | Test Type | Test Data | Purpose of Data | User’s Expected Result | Actual Result |
| append.barcode  search.barcode | string | sdkjwlk, 120172, LGSI2991, MGSI1202.2 | To store product barcode | The product stores the barcode number or exits the program if wrong input is provided. | The product stores the barcode number or exits the program if wrong input is provided. |



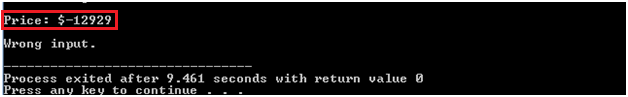






|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Variable Name | Test Type | Test Data | Purpose of Data | User’s Expected Result | Actual Result |
| append.price | float | 0, -12929 | To store product price | The program stores the price information or exits the program if wrong input is provided. | The program stores the price information or exits the program if wrong input is provided. |





|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Variable Name | Test Type | Test Data | Purpose of Data | User’s Expected Result | Actual Result |
| budget | float | sksk | To store budget value to calculate stocks | The program stores the budget price. If characters are inputted, exits the program. | The program stores the budget price. If characters are inputted, exits the program. |



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Variable Name | Test Type | Test Data | Purpose of Data | User’s Expected Result | Actual Result |
| itemssold | integer | soeoe | To store item number to calculate total income | Stores the number of items. If number of items are 0 or characters, exits the program. | Stores the number of items. If number of items are 0 or characters, exits the program. |



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Variable Name | Test Type | Test Data | Purpose of Data | User’s Expected Result | Actual Result |
| current\_price | float | skksks, 0 | To store prices of items to calculate total income. | Stores the income values and displays the total income. If a character or 0 income is inputted, exits the program. | Stores the income values and displays the total income. If a character or 0 income is inputted, exits the program. |





**Conclusion**

It can be concluded that a programmed inventory is the most efficient way to store data for Matthew’s Grocery Store as compared to other methods which are impractical for this setting.

**Appendix**

Questionnaire Instructions: Please put a tick (🗹) in the box next to the answer of your choice or write in the space provided as the case may be.

1. What is your gender?

* Male
* Female

1. In what range does your age fall?

* 16-25 years
* 26-35 years
* 36-45 years
* 46+ years

1. To which ethnic group do you belong?

* African
* Amerindian
* Chinese
* East Indian
* European
* Portuguese
* Mixed
* Other

1. For how many years have you been operating an inventory system?

* Less than a year
* 1 to 3 years
* 4 to 10 years
* More than 10 years

1. Which of the following type of inventory system have you worked on?

* Programmed (you use a computer software to manage data)
* Database (you type the data)
* Manual (you write out the data)

1. Is provision made for obsolete or inactive items/stock?

* Yes
* No

1. What do you think is the best method of storing data and why?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Are recurrent inventory records updated promptly?

* Yes
* No

1. What are some limitations of the inventory system you currently use?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Did you enjoy this questionnaire?

* Yes
* No