
UNIX
management system

the vision:

to create a streamlined grocery management system that is efficient, quick, and takes up minimal space.

///code breakdown:

1. the structure:

```
struct groceries {
  char type[50];
  char prod_name[50];
  char price[50];
} inventory[100];
```

the structure construction was done carefully to avoid the hassle of overly-complicating a grocery item list. after careful consideration, unnecessary elements like QR code, stock, and item condition were removed, keeping only the absolute necessary elements in the code.
further to note, price was stored as a string, instead of a number, in the case that a very expensive item was stored, and the value limit of int would pose a major risk, and using a float or a double type would take up more space, which would destroy our vision of creating an efficient, **compact**, and easy-to-use program.

2. the main function:

the first part of the main function consists of the print statements that show the user the 'main menu', along with the header. the user is then prompted to enter a number, which 'takes' them to the corresponding function, i.e, performing a function call.

```c
int main() {
  FILE *fp;
  int main_no, valid_input, ret_ex;

  // MAIN MENU

  printf("\t# UNIX GROCERY MANAGEMENT SYSTEM #\n\n");

  printf("1.Add Products\n");
  printf("2.View Products\n");
  printf("3.Search Products\n");
  printf("4.Delete Products\n");
  printf("5.Update Products\n");

  printf("Choose any of the options above (enter the corresponding number):");
  scanf("%d", &main_no);
  fflush(stdin);

```

after the user input is stored, the code consists of a switch with multiple cases, each of which is assigned a specific number which is taken from the user's input, hence 'taking' them to that part of the program.

```c
  switch (main_no) {
  case 1:

    printf("Enter number valid inputs(must be less than 100):");
    scanf("%d", &valid_input);
    fflush(stdin);

    prod_counter = valid_input;
    addProd(prod_counter);

```

each of the cases themselves, call a specific non-void or void function which performs tasks as required.

```
18
19    switch (main_no) {
20    case 1:
21
22      printf("Enter number valid inputs(must be less than 100):");
23      scanf("%d", &valid_input);
24      fflush(stdin);
25
26      prod_counter = valid_input;
27      addProd(prod_counter);
28
29      printf("\n0: Return to main menu\nPress anything to exit.\n");
30      scanf("%d", &ret_ex);
31      fflush(stdin);
32
33      if (ret_ex == 0) {
34        main_menu();
35      } else {
36        exit(0);
37      }
38
39      break;
40
41    case 2:
42
43      viewProd();
44
45      printf("\n0: Return to main menu\nPress anything to exit.\n");
46      scanf("%d", &ret_ex);
47      fflush(stdin);
48
49      if (ret_ex == 0) {
50        main_menu();
51      } else {
52        exit(0);
53      }
54
55      break;
56
57    case 3:
58
59      searchProd();
60
61      printf("\n0: Return to main menu\nPress anything to exit.\n");
62      scanf("%d", &ret_ex);
63      fflush(stdin);
64
65      if (ret_ex == 0) {
66        main_menu();
67      } else {
68        exit(0);
```

an additional feature added was the fact that upon completing a task from the ones chosen in the menu **on the first run**, the user is then presented with an additional option to either exit the system program completely, or return to the main menu again, and continue tasks as usual, **all in a single run of the program.**

///actual run:

```
        # UNIX GROCERY MANAGEMENT SYSTEM #

  1.Add Products
  2.View Products
  3.Search Products
  4.Delete Products
  5.Update Products
  Choose any of the options above (enter the corresponding number):
```

the main menu is shown in the terminal.

```
  Choose any of the options above (enter the corresponding number):2
  carrot
  vegetable
  34
  apple
  fruit
  16
  chanachur
  food
  45

  0: Return to main menu
  Press anything to exit.
```

after choosing an option and the task being executed, the user is prompted to either return to the main menu, or to terminate the program.

```
0: Return to main menu
Press anything to exit.
9
Rajins-MacBook-Pro:projectrun matiar$
```

or

```
0: Return to main menu
Press anything to exit.
0
       # UNIX GROCERY MANAGEMENT SYSTEM #

1.Add Products
2.View Products
3.Search Products
4.Delete Products
5.Update Products
Choose any of the options above (enter the corresponding number):
```

as programmed and expected, respective inputs terminate the code, or let the user return to the main menu.

now the question lies, **how was the menu recalled**?

3. recalling the main menu:

the logic behind calling the menu 'again' is deceptively simple.

as you may have noticed before, every time the menu is called again, for each case in the switch, an input of 0 calls a void function named main_menu.

```
28
29      printf("\n0: Return to main menu\nPress anything to exit.\n");
30      scanf("%d", &ret_ex);
31      fflush(stdin);
32
33      if (ret_ex == 0) {
34        main_menu();
35      } else {
36        exit(0);
37      }
```

this void function stores a **copy** of the actual menu code.

```
241
242   void main_menu() {
243     int main_no, valid_input, ret_ex;
244
245     // MAIN MENU
246
247     printf("\t# UNIX GROCERY MANAGEMENT SYSTEM #\n\n");
248
249     printf("1.Add Products\n");
250     printf("2.View Products\n");
251     printf("3.Search Products\n");
252     printf("4.Delete Products\n");
253     printf("5.Update Products\n");
254
255     printf("Choose any of the options above (enter the corresponding number):");
256     scanf("%d", &main_no);
257     fflush(stdin);
258
259   >  switch (main_no) {…
344   }
345
```

so, every time the menu is called again in a single run, the code simply calls this void main_menu function, but to the user, it still seems as if the they are being returned to the main menu itself, further adding to the **streamlined user experience.**

4. adding products:

to start off, this non-void function passes the number of valid inputs, ‘valid_entry’, that the user is going to add to the existing list as its parameter.

```
void addProd(int valid_entry) {
  FILE *fp;

  int counter = 0;

  fp = fopen("INVENTORY.txt", "r");

  for (int i = 0; i < 100; i++) {
    fscanf(fp, "%s%s%s", inventory[i].prod_name, inventory[i].type,
           inventory[i].price);

    if (strcmp(inventory[i].prod_name, "") == 0) {
      break;
    }

    counter++;

    if (feof(fp)) {
      break;
    }
  }

  fclose(fp);
```

then, it opens the file where the items are listed, titled INVENTORY, in read mode. it scans its contents and stores it in an each corresponding element of the **inventory** array, as declared in the structure earlier.
it then checks if the first nested element in the first index of the array is empty with strcmp, and by doing so finds out whether or not the file is empty.
if not, a counter (initialized earlier) keeps track of each item stored.
the loop terminates either after a hundred items are stored, or if the end of file is reached.

it is important to count how many items, if any, are stored. because the addProd function will start adding items **after** the last-stored item.

```
if (counter < 100) {
  fp = fopen("INVENTORY.txt", "a");

  for (int i = 0; i < valid_entry; i++) {
    printf("\nEnter the name of the item: ");
    scanf(" %[^\n]", inventory[i].prod_name);
    fprintf(fp, "\n%s", inventory[i].prod_name);

    printf("Enter the type of the item: ");
    scanf(" %[^\n]", inventory[i].type);
    fprintf(fp, "\n%s", inventory[i].type);

    while(1)
    {
        int flag = 1;

        printf("Enter the price of the item: ");
        scanf(" %[^\n]", inventory[i].price);

        for (int j=0; inventory[i].price[j]!='\0'; j++)
        {
            if (!(inventory[i].price[j]<='9' && inventory[i].price[j]>='0'))
            {
                flag = 0;
                printf("\nInvalid input. Please re-enter your price.\n");
                break;
            }
        }

        if (flag==1)
        {
          break;
        }

    }


    fprintf(fp, "\n%s", inventory[i].price);

    counter++;

    if (counter == 100) {
      printf("\nNow your inventory is full. You cannot store more than 100 "
             "items.\n");
      break;
    }
  }
```

first, the item count determines whether or not the list is already full. if not, the file INVENTORY is opened in append mode, **and** the loop for taking input is started, to continue until the number of inputs specified by the user.

since the price was stored as a string instead of an integer type to save space, the input had to be verified.
in this case, each character's ASCII code was compared to that of the digits between 0 and 9, index by index in the **price** array, and if anything but digits were found, it would prematurely break the verification loop and exit to the infinite loop that would ask the user to enter the price, until they got it right.
if, and only if the price is entered correctly, the program moves on as the infinite loop terminates, and the item list counter increments by 1, after taking 3 inputs for product specifications.

this counter is then used to check whether or not the 'inventory' is full. if it reaches 100 while the input loop is running, the function stops taking any further inputs.

```
66        if (counter == 100) {
67          printf("\nNow your inventory is full. You cannot store more than 100 "
68                 "items.\n");
69          break;
70        }
71      }
72
73      fclose(fp);
74    }
75
76    else {
77      printf("\nNow your inventory is full. You cannot store more than 100 "
78             "items.\n");
79    }
80  }
```

lastly, if the inventory was already full, it simply does not take any further input.

///actual run:

```
          # UNIX GROCERY MANAGEMENT SYSTEM #

  1.Add Products
  2.View Products
  3.Search Products
  4.Delete Products
  5.Update Products

  Choose any of the options above (enter the corresponding number): 1

  Enter the number of items you would like to enter (must be less than 100): 2

  Enter the name of the item: hammer
  Enter the type of the item: tool
  Enter the price of the item: 450
```

as programmed, this menu prompts the user to enter the number of items they would like to input, and then runs the loop that many times and stores them in the list, which is updated once the function is exited.

```
  Enter the name of the item: speakers
  Enter the type of the item: electronics
  Enter the price of the item: fifty

  Invalid input. Please re-enter your price.
  Enter the price of the item: 50

  0: Return to main menu
  Press anything to exit.
  1
  Rajins-MacBook-Pro:projectrun matiar$
```

if anything but digits are put in by the user, they are prompted to re-enter the value until a number is input for the price.

5. viewing products:

```c
void viewProd() {

  struct groceries view_inventory[100];

  FILE *fp;
  int count = 0;

  fp = fopen("INVENTORY.txt", "r");

  while (1) {
    fscanf(fp, "%s%s%s", view_inventory[count].prod_name, view_inventory[count].type, view_inventory[count].price);
    count++;
    if (feof(fp))
      break;
  }
  for (int i = 0; i < count; i++) {
    printf("%s\n%s\n%s\n", view_inventory[i].prod_name, view_inventory[i].type, view_inventory[i].price);
  }
  fclose(fp);
}
```

for viewing the products, a new array of the maximum size is declared, and a counter is initialized again.
then, an infinite loop runs to scan the products and their information from the 'INVENTORY' text file and load it on to the respective nested structure arrays.

afterwards, a simple loop that runs as many times as there are items (using the counter), and simply prints the information from the stored arrays.

the reason that a new array is used is to make the program less error-prone, as it may be susceptible to bugs if we depend on the previously stored array which may or may not have been cleared, or have old records present in it.

///actual run:

```
        # UNIX GROCERY MANAGEMENT SYSTEM #

1.Add Products
2.View Products
3.Search Products
4.Delete Products
5.Update Products

Choose any of the options above (enter the corresponding number): 2

hammer
tool
450

speakers
electronics
50

0: Return to main menu
Press anything to exit.
1
Rajins-MacBook-Pro:projectrun matiar$
```

6. searching for products:

```c
void searchProd() {
  FILE *fp;
  char search_key[50];
  int count = 0;

  fp = fopen("INVENTORY.txt", "r");

  printf("Enter search key: ");
  scanf(" %[^\n]", search_key);

  while (1) {
    fscanf(fp, "%s%s%s", inventory[count].prod_name, inventory[count].type,
           inventory[count].price);

    if (strcmp(search_key, inventory[count].prod_name) == 0) {
      printf("%s\n%s\n%s\n", inventory[count].prod_name, inventory[count].type,
             inventory[count].price);
    }
    count++;
    if (feof(fp))
      break;
  }

  fclose(fp);
}
```

this void function scans and reads the items off the list onto the inventory array.
then, by using the strcmp function, we take the search key (user input) and then compare it with the name of each consecutive item on the list (now stored in an array).
if a match isn't found, the counter increments by 1, and we search in the next index. this goes on until a match is found, and when it is, all the information of that product is printed on the screen.

///actual run:

```
        # UNIX GROCERY MANAGEMENT SYSTEM #

1.Add Products
2.View Products
3.Search Products
4.Delete Products
5.Update Products

Choose any of the options above (enter the corresponding number): 3

Enter search key: bvlgari

bvlgari
perfume
25000

0: Return to main menu
Press anything to exit.
69
Rajins-MacBook-Pro:projectrun matiar$
```

7. deleting products:

before anything is done, all the stored items are loaded up onto the inventory array.

```
void deletProd() {
  FILE *fp;
  int count = 0;
  char search_key[50];

  fp = fopen("INVENTORY.txt", "r");

  while (1) {
    fscanf(fp, "%s%s%s", inventory[count].prod_name, inventory[count].type, inventory[count].price);

    count++;

    if (feof(fp))
      break;
  }

  fclose(fp);
```

then, the user is prompted to enter a search key with which the array index of the item to be deleted is found (just the product name). again, the strcmp function is used here.

```

  printf("Enter search key: ");
  scanf(" %[^\n]", search_key);
  fflush(stdin);

  int sk_idx;

  for (int i = 0; i < count; i++) {
    if (strcmp(search_key, inventory[i].prod_name) == 0) {
      sk_idx = i;
      break;
    }
  }
```

```
27          sk_idx = i;
28          break;
29        }
30      }
31      for (int i = sk_idx; i < count - 1; i++) {
32        strcpy(inventory[i].prod_name, inventory[i + 1].prod_name);
33        strcpy(inventory[i].type, inventory[i + 1].type);
34        strcpy(inventory[i].price, inventory[i + 1].price);
35      }
36      count--;
37
38      fp = fopen("INVENTORY.txt", "w");
39
40      for (int i = 0; i < count; i++) {
41        fprintf(fp, "%s\n%s\n%s\n", inventory[i].prod_name, inventory[i].type, inventory[i].price);
42      }
43
44      fclose(fp);
45    }
```

once the array index of the item to be deleted is found, the same concept used when 'deleting' an item from an array is used.
the only difference is, instead of assigning the value of the next index to the previous one, we use the strcpy function to copy the string stored in the next index to the previous index, hence consecutively 'shifting' all the elements and overwriting the index that needed to be 'deleted'.

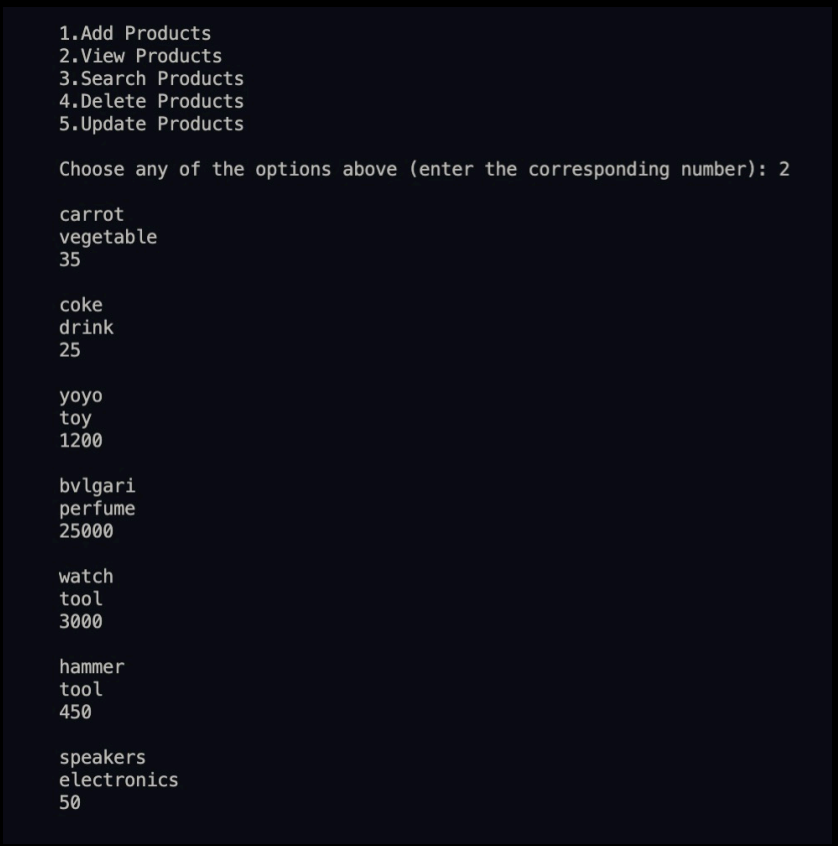
then, the file is opened in write mode, which removes all previous data in the list and prepares everything to be written into the file from scratch.
then, the new array, with the 'deleted' item, is copied on the 'INVENTORY' text file, thereby completely deleting the item we searched for and all its related information.

///actual run:

```
        # UNIX GROCERY MANAGEMENT SYSTEM #

1.Add Products
2.View Products
3.Search Products
4.Delete Products
5.Update Products

Choose any of the options above (enter the corresponding number): 4

Enter search key: hotwheels

0: Return to main menu
Press anything to exit.
0
```

the item to be deleted is read from the user, and after that input, the program simply prompts the user to either return to the main menu, or exit the program. most of the process happens under the hood.

initial list of items.



list after deletion (as viewed from the program itself).

8. updating products:

```c
void updateProd() {
  FILE *fp;
  int count = 0;
  char search_key[50];

  fp = fopen("INVENTORY.txt", "r");

  while (1) {
    fscanf(fp, "%s%s%s", inventory[count].prod_name, inventory[count].type,
           inventory[count].price);

    count++;

    if (feof(fp))
      break;
  }

  fclose(fp);
```

to start off, the arrays are initialized and filled up with the consecutive items on the inventory list. a counter keeps track of the total number of items read.

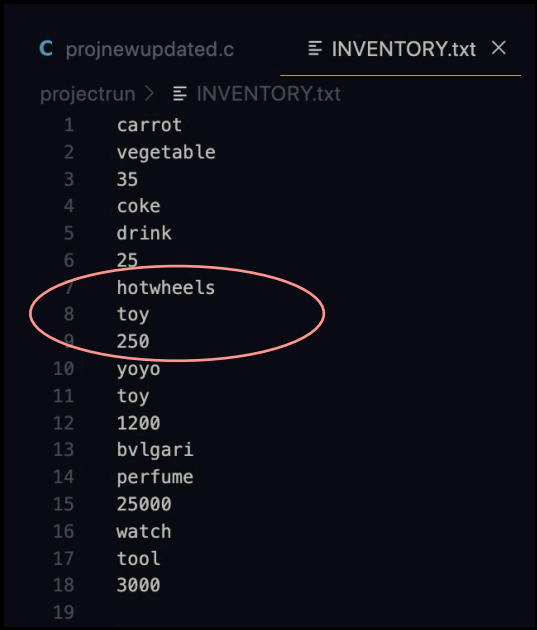
```c
  printf("Enter search key: ");
  scanf(" %[^\n]", search_key);
  fflush(stdin);

  char up_name[50], up_type[50], up_price[50];

  for (int i = 0; i < count; i++) {
    if (strcmp(search_key, inventory[i].prod_name) == 0) {
      printf("\nEnter the name of the item: ");
      scanf(" %[^\n]", up_name);
      fflush(stdin);
      strcpy(inventory[i].prod_name, up_name);

      printf("Enter the type of the item: ");
      scanf(" %[^\n]", up_type);
      fflush(stdin);
      strcpy(inventory[i].type, up_type);

      printf("Enter the price of the item: ");
      scanf(" %[^\n]", up_price);
      fflush(stdin);
      strcpy(inventory[i].price, up_price);
    }
  }
```

then, a search key is read from the user, and character arrays are declared to take the string input for the updated items. the strcmp function is then used to check the product name which matches the search key, and then strcpy is used to update the old record with the one input.

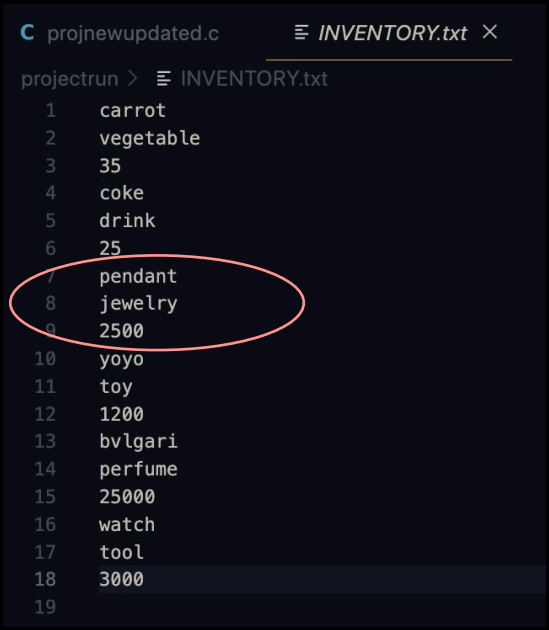
```c
    fp = fopen("INVENTORY.txt", "w");

    for (int i = 0; i < count; i++) {
      fprintf(fp, "%s\n%s\n%s\n", inventory[i].prod_name, inventory[i].type,
              inventory[i].price);
    }

    fclose(fp);
}
```

finally, the file is opened in write mode, and the file is rewritten along with the new data.

///actual run:



initial list of items.



list after updating.

```
        # UNIX GROCERY MANAGEMENT SYSTEM #

1.Add Products
2.View Products
3.Search Products
4.Delete Products
5.Update Products

Choose any of the options above (enter the corresponding number): 5
Enter search key: hotwheels

Enter the name of the item: pendant
Enter the type of the item: jewelry
Enter the price of the item: 2500

0: Return to main menu
Press anything to exit.
9
Rajins-MacBook-Pro:projectrun matiar$
```

the run.