Data Structure and Algorithms Project
All-Mart Management

Member's:

1.Sirigiri.Susant Naidu 17BCE0942, 8.4
+91 95021 16394

2.Nirmal P Mathew 17BCE0328, 8.8
+91 96404 47777

3.Rathnam Sasidhar Achari 17BCE0895, 8.9
+91 91821 56686

4.D.Sai Venkat 17BCI0008, 7.3
91 90033 60335

5.Adithya.S.Anil 17BCI0163 8.2
+919080818339

DECLARATION

We hereby declare that the project work entitled
"All-Mart Management"
submitted to the VIT Vellore, is a record of an original work done by me under the guidance of
Mr. Debi Prasanna Acharjya, and this project work is submitted in the partial fulfillment of the requirements for the award of the degree of B.Tech in Computer Science& Engineering. The results embodied in this thesis have not been submitted to any other University or Institute for the award of any degree or diploma

Sirigiri.Susant Naidu 17BCE0942
Nirmal P Mathew 17BCE0328
Rathnam Sasidhar Achari 17BCE0895
D.Sai Venkat 17BCI0008
Adithya.S.Anil 17BCI0163

Acknowledgment

We have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. We would like to extend my sincere thanks to all of them.

We are highly indebted to VIT for their guidance and constant supervision as well as for providing necessary information regarding the project and also for their support in completing the project.

We would like to express my gratitude towards our parents and member of VIT for their kind co-operation and encouragement which help me in completion of this project.

Our thanks and appreciations also go to all faculty in developing the project and people who have willingly helped us out with their abilities.

Introduction

This program intends to digitize all the operation required to run a modern store such as the all mart .This includes:

- Inventory.
- Creation of a database.
- Billing.

In a store there are various processes that is required for the smooth running of the store, thus automating these process will both provide a economic and temporal benefits for those who run the store

There arises a need for creation of a database as it is required to know the details of the store, such as inventory, price and discounts, etc

The program is done with the use of the C++ language and use of both derived data type and user defined data type using dynamic memory allocation, namely using linked list to create the database. The programming paradigm to be used is object oriented programming, as it makes the various elements to the program, easier to manage, program and update should the need arise.

Aspects this program covers:

INVENTORY MANAGEMENT

Inventory management is the practice overseeing and controlling of the ordering, storage and use of components that a company uses in the production of the items it sells.

Inventory management is the management of inventory and stock. Inventory management is an ongoing process where businesses manage their products. Some are using an automated process. The manual process is very labour intensive.

Benefits of Inventory Management:

- Keep the business profitable
- Reduce costs
- Achieve economies of scale
- Analyze sales patterns and predict future sales
- Analyze performance against competitors
- Prepare the business for the unexpected.

Inventory is vital because it allows a business to have what it needs, at the right time in order to continue to do business. Companies will usually aim to minimize required investment in inventory while maintaining operations and controlling both waste and surplus.

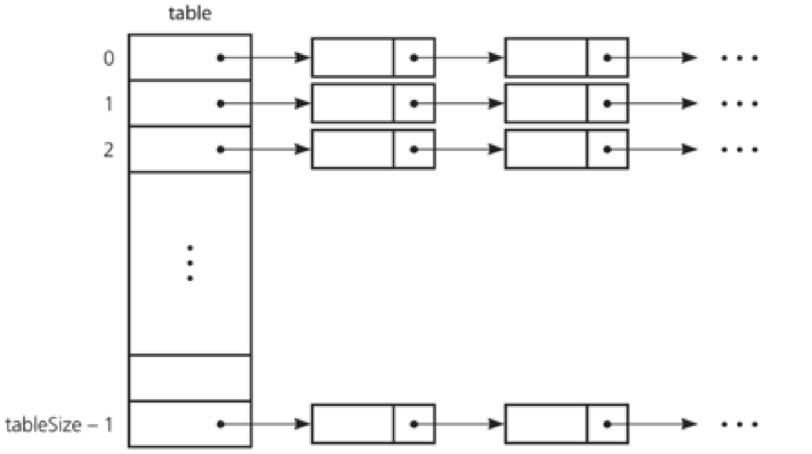
DATABASE MANAGEMENT:

One needs will to maintain the database continuously to be useful. If one does not keep up with things like inventory, stock, billing, etc,the database ends up just being another fixture with no purpose.
The bigger your shop gets, the more time one will need to dedicate to maintaining and managing the data.
Hence, Data base management is important.

This programs uses Text file as a source for its database operation. This in combination with cpp file handling, allows one to load the database from the secondary memory to the primary memory. The upside is that it is easy to use and implement, portable, and small in size. However it is vulnerable to manual manipulation. However, in this case that is not an issue, thus Txt file can satisfy the needs to fulfill our simple database.

The Algorithm used

The program uses Hashing table with Singly Linked Lists, to store the data in the primary memory.



Each location of the hash table contains a pointer to a linked list

A visual representation of the data in the memory

The use of Linked list in combination with array is done in order to avoid collisions, in the indexing. When a collusion occurs the next node is simply added into the linked list.
Else, the node is added to the array of pointers.

This data structure is then interfaced to the secondary memory using file handling operations. The vice versa also holds true(I.e,from secondary memory to primary memory).

This is done so that all the operation on the data is efficient.

Code

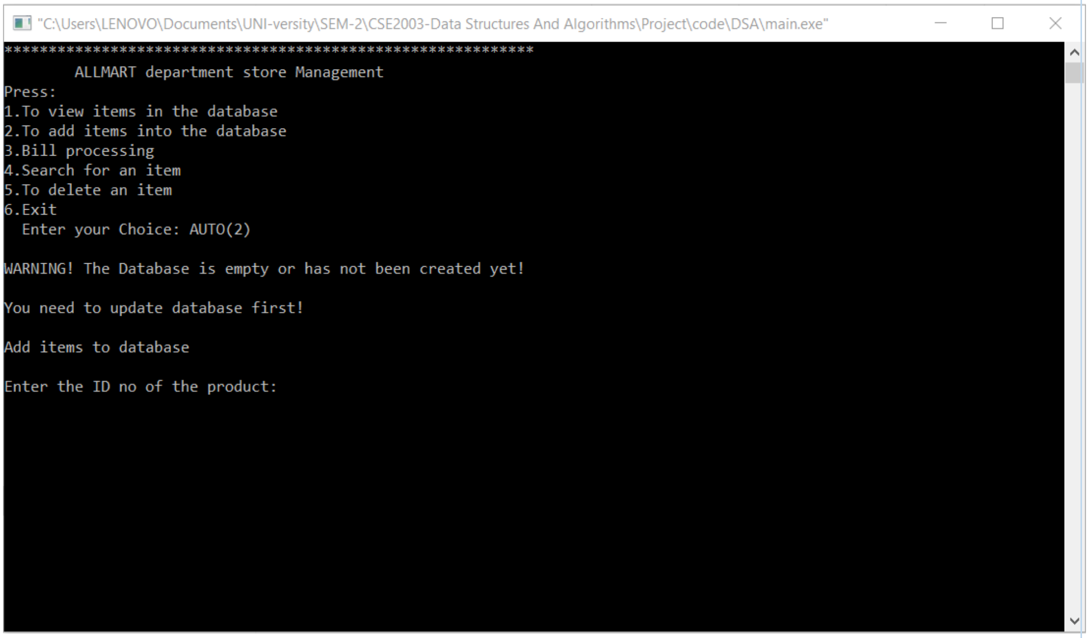
```
#include <iostream>
#include <fstream>
#include <windows.h>
#define nmax 10
using namespace std;
struct hashnode//Defintion of node in the linked list
{
    int id;
    string name;
    int quantity;
    float price;
    hashnode *next;
};
bool check_empty()
{
    ifstream file;
    file.open("record.txt");
    char x;
    file>>x;
    if (isalpha(x)) //if file is empty=false
    {
        file.close();
        return true;
    }
    else
    {
        file.close();
        return false;
    }
}
class hashmap
{
private:
    hashnode **htable;
    int bill[100];
    int billcount=0;

public:

    hashmap()
    {
        htable=new hashnode*[nmax];
        for(int i=0;i<nmax;i++)
            htable[i]=NULL;
    }

    int hashfunc(int id)     {
        return id%nmax;
    }
    void push(int idno,string name,float price,int quantity)
    {
        int index=hashfunc(idno);
        hashnode *first;
        hashnode *prev;
        first=new hashnode;
        if(htable[index]==NULL)
        {
            prev=first;
            htable[index]=first;
            first->id=idno;
            first->name=name;
            first->price=price;
            first->quantity=quantity;
            first->next=NULL;
        }
        else
        {
            prev->next=first;
            first->next=NULL;
            prev=first;
            first->id=idno;
            first->name=name;
            first->price=price;
            first->quantity=quantity;

        }


    }

    //function to pop an element out of the Linked list
    void pop(int idS)
    {
        int index=hashfunc(idS);
       // cout<<"\nindex: "<<index<<endl;
        hashnode *prev=new hashnode;
        hashnode *bottom=htable[index];
        while(bottom!=NULL)
        {

            if(bottom->id==idS && bottom==htable[index])
            {
                htable[index]=NULL;
                delete bottom;
                break;
            }
           if(bottom->id!=idS)
           {
               prev=bottom;
               bottom=bottom->next;

           }
           else
           {
               prev->next=bottom->next;
```

```
                   delete bottom;
                   break;
               }
            }
        }

        //fucntion to display the contents of the Linked list
        void display()
        {
           hashnode *bottom=NULL;
           for(int i=0;i<nmax;i++)
           {
              if(htable[i]!=NULL)
              {
                bottom=htable[i];
                while(bottom!=NULL)
                {
                    cout<<"Name: "<<bottom->name<<'\n'<<"id: 
"<<bottom->id<<'\n';
                    cout<<"Price: 
"<<bottom->price<<endl<<"Quantity: "<<bottom->quantity<<"\n\n";
                    bottom=bottom->next;
                }
                }
           }
         }

         void updatefile()
         {
           hashnode *bottom;
           ofstream file;
           file.open("record.txt");
           for(int i=0;i<nmax;i++)
           {
              if(htable[i]!=NULL)
              {
                bottom=htable[i];
                while(bottom!=NULL)
                {

                    file<<bottom->name<<'\n'<<bottom->id<<'\n';

file<<bottom->price<<'\n'<<bottom->quantity<<'\n';
                    bottom=bottom->next;
                }

             }
           }
           file.close();
         }
        float bill_find(int ids)
        {
            float total_price=0;
            int index=hashfunc(ids);
            hashnode *bottom=htable[index];
            while(bottom!=NULL)
            {
                if(bottom->id==ids)
                {
                    cout<<"Name: "<<bottom->name<<endl<<"id: 
"<<bottom->id<<endl;
                    cout<<"Price: 
"<<bottom->price<<endl<<"Quantity: "<<bottom->quantity<<endl;
                    if(bottom->quantity!=0)
                    {
                        bottom->quantity--;
                        updatefile();
                        cout<<"\nNEW QUANTITY after Bill: 
"<<bottom->quantity<<endl;
                        return bottom->price;
                    }
                    else
                    {
                        cout<<"Stock is empty for this 
item"<<endl;
                    }
                }
                bottom=bottom->next;
            }
        }
        int verifyID(int ids)
        {
            int flag=0;
            int index=hashfunc(ids);
            hashnode *bottom=htable[index];
            while(bottom!=NULL)
            {
                if(bottom->id==ids)
                {
                    return 0;
                }
                bottom=bottom->next;
            }
            if(flag==0)
                return -1;
        }
        int Search(int ids)
        {
            int flag=0;
            int index=hashfunc(ids);
            hashnode *bottom=htable[index];
            while(bottom!=NULL)
            {
                if(bottom->id==ids)
                {
                    cout<<"\nItem found!\n";
                    return 0;
                }
                bottom=bottom->next;
            }
            if(flag==0)
                return -1;

        }

        void bill_func(int ids)
        {
            bill[billcount]=ids;
            billcount++;
        }
        void billing()
        {
            float total_price=0;
            for(int i=0;i<billcount;i++)
            {
               total_price=total_price+bill_find(bill[i]);
            }
            cout<<"\nTotal price is: "<<total_price<<endl;
        }
    };

    bool check_file()
    {

        ifstream file;
        file.open("record.txt");
        if (file)
            {
            file.close();
            return true;
            }
        else
        {
            file.close();
            return false;
        }
    }

    int get_no_of_node()
    {
        int counter=0;
        string line;
        ifstream file("record.txt");
        while (getline(file, line))
            counter++;
        file.close();
        return counter/4;
    }
```

```
    int loadfile(hashmap &s)
    {
        int no_nodes=get_no_of_node();
        int id_temp[no_nodes];
        string nametemp[no_nodes];
        float price_temp[no_nodes];
        int quantity_temp[no_nodes];
        ifstream filein;
        filein.open("record.txt");
        if(check_file()&&check_empty())
        {  for(int i=0;i<no_nodes;i++)
           {

filein>>nametemp[i]>>id_temp[i]>>price_temp[i]>>quantity_temp[i];
           }
           for(int i=0;i<no_nodes;i++)

s.push(id_temp[i],nametemp[i],price_temp[i],quantity_temp[i]);
        }
        else
            return -1;
    }
    void dest_file()
    {
        hashmap t;
        t.updatefile();
    }
    void menu(hashmap &s,int vef)
    {
        int choice;
        int idno;
        int flag=-1;
        int ids;
        string namet;
        float pricet;
        int quantityt;
        char ans;
        retry:
           system("cls");
        for(int i=0;i<60;i++)
            cout<<"*";
        cout<<"\n\tALLMART department store Management";
        cout<<endl;
        cout<<"Press: \n1.To view items in the database\n2.To add
items into the database\n3.Bill processing\n4.Search for an
item\n5.To delete an item\n6.Exit";
        cout<<"\n  Enter your Choice: ";
        if(vef!=-1)
            cin>>choice;
        else
        {
            cout<<"AUTO(2)";
            choice=2;
        }
        switch(choice)
        {
        case 1:
            cout<<endl;
            system("cls");
            cout<<"\tThe Items in the database are:"<<endl;
            s.display();
            cout<<"\n\n";
            system("pause");
            goto retry;
           break;
        case 2:
            cout<<endl;
            if(vef!=-1)
                system("cls");
            else
            {
                cout<<"\nWARNING! The Database is empty or has not
been created yet!"<<endl;
                cout<<"\nYou need to update database first!\n";
            }
            cout<<"\nAdd items to database\n";
            do
            {

                cout<<"\nEnter the ID no of the product: ";
                cin>>idno;
                int dup=s.verifyID(idno);
                if(dup!=0)
                {
                    cout<<"\nEnter the name of the product: ";
                    cin>>namet;
                    cout<<"\nEnter the price of the product: ";
                    cin>>pricet;
                    cout<<"\nEnter the quantity of the product: ";
                    cin>>quantityt;
                    s.push(idno,namet,pricet,quantityt);
                    vef++;
                }
                else
                {
                    cout<<"\nThis ID no has been assigned to
another product             already!\n";
                }

                cout<<"\nDo you want to continue?(y/n): ";
                cin>>ans;
            }while(ans=='y'||ans=='Y');
            s.updatefile();
            if(ans!='y'||ans!='Y')
               goto retry;
            break;
       case 3:
            if(vef!=-1)
                system("cls");
            flag=-1;
            cout<<"\nBilling!\n";
            do
            {
                cout<<"\nEnter the ID of the product:\n";
                cin>>ids;
                s.bill_func(ids);
                cout<<"\nDo you want to continue?(y/n): ";
                cin>>ans;

            }while(ans=='y'||ans=='Y');
            cout<<"\nTHE ITEMS BILLED ARE: "<<endl;
            s.billing();
            system("pause");
            flag++;
            if(ans!='y'||ans!='Y'&&flag!=-1)
                goto retry;
            break;
        case 4:
            if(vef!=-1)
                system("cls");
            cout<<"\nSearching!\n";
            flag=-1;
            do
            {
                cout<<"\nEnter the ID of the product to be
 searched:\n";
                cin>>ids;
                flag=s.Search(ids);
                if(flag==-1)
                    cout<<"\nItem not found!\n";
                cout<<"\nDo you want to continue?(y/n): ";
                cin>>ans;
            }while(ans=='y'||ans=='Y');
            if(ans!='y'||ans!='Y')
               goto retry;
            break;
        case 5:
            if(vef!=-1)
                system("cls");
            cout<<"\nDeletion!\n";
            do
            {
                cout<<"\nEnter the ID of the product to be
 deleted:\n";
                cin>>ids;
                flag=s.Search(ids);
                if(flag==-1)
                    cout<<"\nItem not found!\n";
```

```
            else
                s.pop(ids);
            cout<<"\nDo you want to continue?(y/n): ";
            cin>>ans;
        }while(ans=='y'||ans=='Y');
        s.updatefile();
        if(ans!='y'||ans!='Y')
           goto retry;
        break;
    case 6:
        if(vef!=-1)
            system("cls");
        cout<<"Exit!";
        exit(0);
        break;
    default:
        if(vef!=-1)
            system("cls");
        cout<<"Wrong option!";
        goto retry;
        break;

    }
}

int main()
{
    hashmap h;
    if(!check_file())
        h.updatefile();
    int verify=loadfile(h);
    menu(h,verify);

}
```

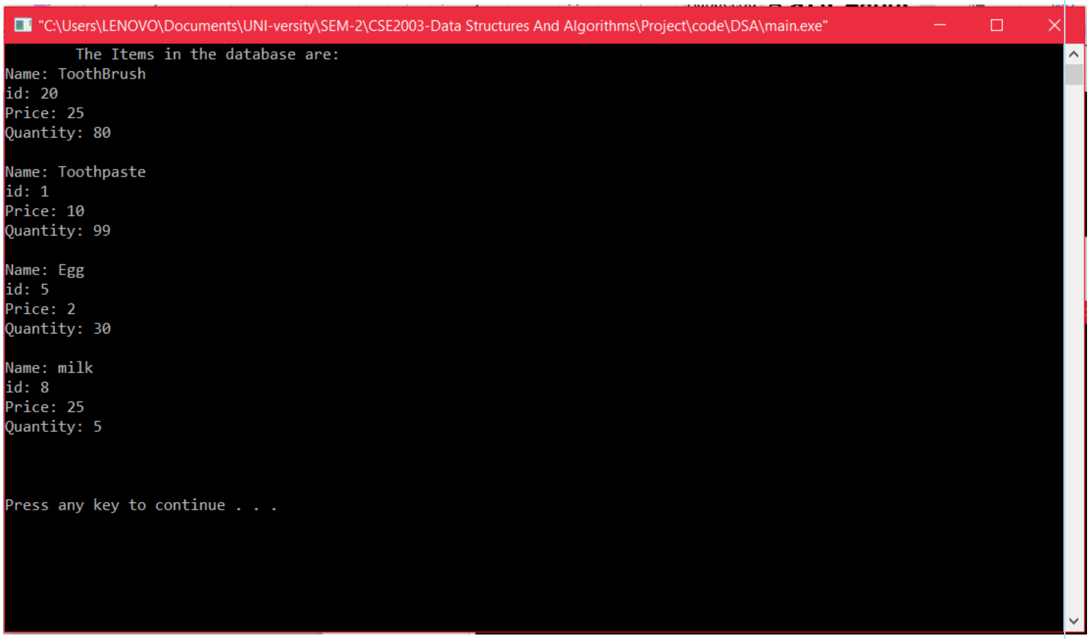
Result Analysis



"C:\Users\LENOVO\Documents\UNI-versity\SEM-2\CSE2003-Data Structures And Algorithms\Project\code\DSA\main.exe"

```
************************************************************
        ALLMART department store Management
Press:
1.To view items in the database
2.To add items into the database
3.Bill processing
4.Search for an item
5.To delete an item
6.Exit
  Enter your Choice: AUTO(2)

WARNING! The Database is empty or has not been created yet!

You need to update database first!

Add items to database

Enter the ID no of the product:
```

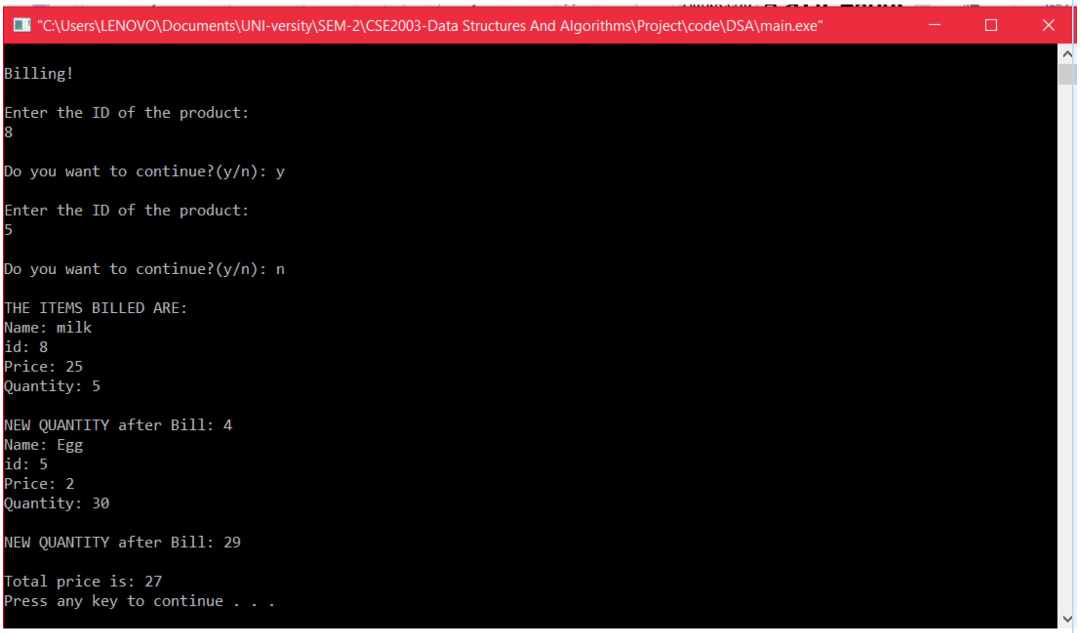
Initially, there is no items in the database as such the programs prompts the user to add an item or more into the database



"C:\Users\LENOVO\Documents\UNI-versity\SEM-2\CSE2003-Data Structures And Algorithms\Project\code\DSA\main.exe"

```
        The Items in the database are:
Name: ToothBrush
id: 20
Price: 25
Quantity: 80

Name: Toothpaste
id: 1
Price: 10
Quantity: 99

Name: Egg
id: 5
Price: 2
Quantity: 30

Name: milk
id: 8
Price: 25
Quantity: 5


Press any key to continue . . .
```

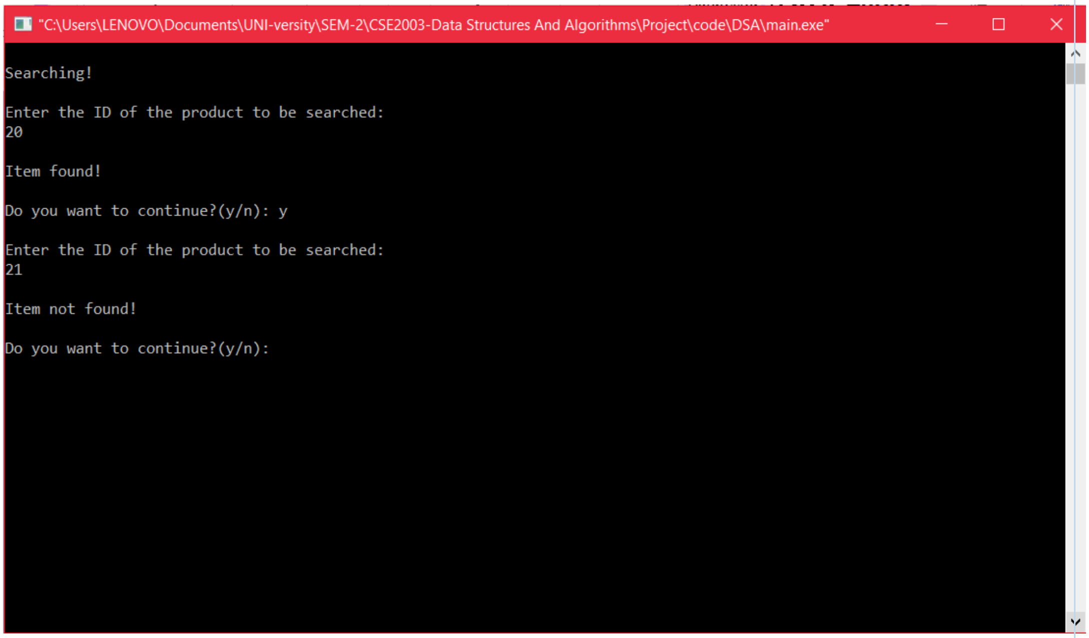
On adding several items to the database



"C:\Users\LENOVO\Documents\UNI-versity\SEM-2\CSE2003-Data Structures And Algorithms\Project\code\DSA\main.exe"

```
Billing!

Enter the ID of the product:
8

Do you want to continue?(y/n): y

Enter the ID of the product:
5

Do you want to continue?(y/n): n

THE ITEMS BILLED ARE:
Name: milk
id: 8
Price: 25
Quantity: 5

NEW QUANTITY after Bill: 4
Name: Egg
id: 5
Price: 2
Quantity: 30

NEW QUANTITY after Bill: 29

Total price is: 27
Press any key to continue . . .
```

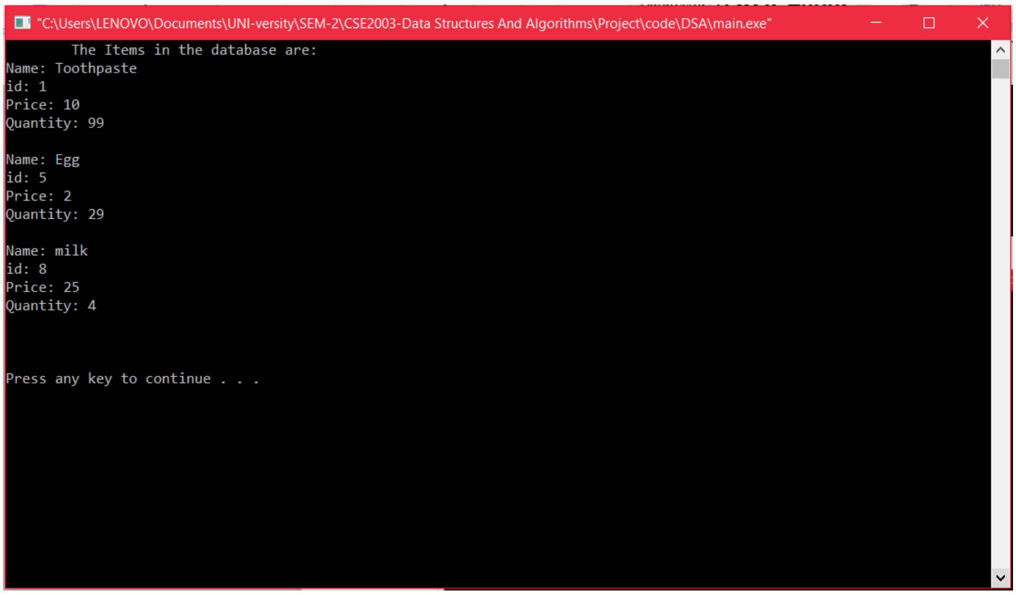
Billing the item, update the quantity and finds the total amount to be paid.



"C:\Users\LENOVO\Documents\UNI-versity\SEM-2\CSE2003-Data Structures And Algorithms\Project\code\DSA\main.exe"

```
Searching!

Enter the ID of the product to be searched:
20

Item found!

Do you want to continue?(y/n): y

Enter the ID of the product to be searched:
21

Item not found!

Do you want to continue?(y/n):
```

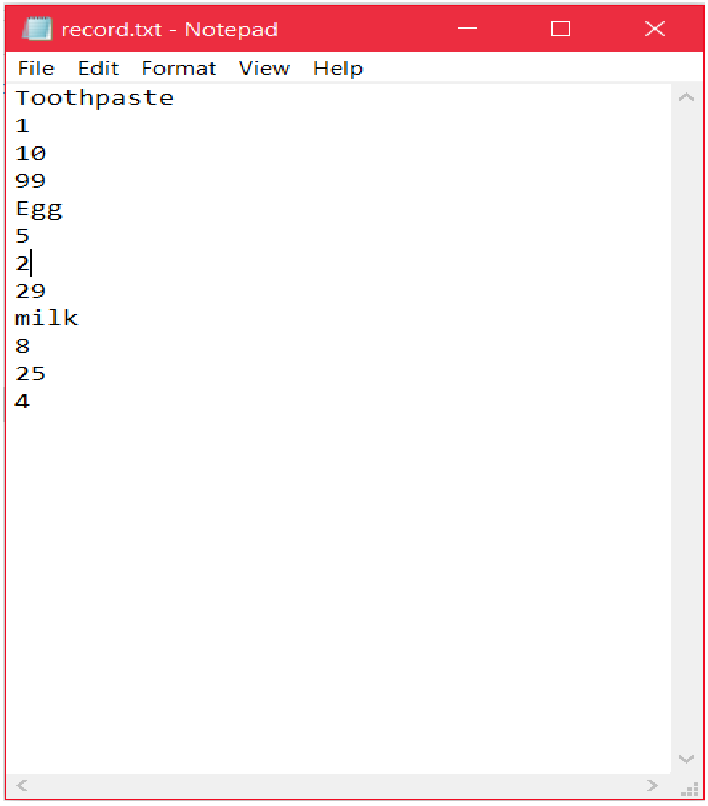
Searching for items in the inventory



"C:\Users\LENOVO\Documents\UNI-versity\SEM-2\CSE2003-Data Structures And Algorithms\Project\code\DSA\main.exe"

```
        The Items in the database are:
Name: Toothpaste
id: 1
Price: 10
Quantity: 99

Name: Egg
id: 5
Price: 2
Quantity: 29

Name: milk
id: 8
Price: 25
Quantity: 4



Press any key to continue . . .
```

Deletion, in this case node with ID no 20 has been deleted.



```
Toothpaste
1
10
99
Egg
5
2
29
milk
8
25
4
```

Final record.txt, i.e. the database in with the items are stored for later use.

Important Functions Explained:

1.

```
struct hashnode
{
      int id;
     string name;
     int quantity;
     float price;
     hashnode *next;
};
```

The node itself,i.e,the data is referenced using this definition of the node.

2.

```
hashmap()
{
    htable=new hashnode*[nmax];
    for(int i=0;i<nmax;i++)
    htable[i]=NULL;
}
```

The constructor of class hashmap that initialize all the elements in the array of pointers(htable) to NULL

3.

```
int hashfunc(int id)
{
    return id%nmax;
}
```

The hashing function used to find the index of the element to be added into the array of pointers. This function is simple and is prone to collisions, Especially with the max size of the array is small.

4.

```
void push(int idno,string name,float price,int quantity)
{
    int index=hashfunc(idno);
    hashnode *first;
    hashnode *prev;
    first=new hashnode;
    if(htable[index]==NULL)
    {
        prev=first;
```

```
       htable[index]=first;
       first->id=idno;
       first->name=name;
       first->price=price;
       first->quantity=quantity;
       first->next=NULL;
       }
   else
   {
      prev->next=first;
      first->next=NULL;
      prev=first;
      first->id=idno;
      first->name=name;
      first->price=price;
      first->quantity=quantity;
   }
}
```

This function is used to add nodes into the data structure, when the structure is empty the if statement holds true and thus creates the first node to with the htable[i] ;where i is the index found through the hashing function;is pointed to. Every other node added to htable[i] ,is added to the linked list instead.
In this way all multiples are grouped together greatly improving computational time.

5.

```
void pop(int idS)
{
    int index=hashfunc(idS);
    hashnode *prev=new hashnode;
    hashnode *bottom=htable[index];
    while(bottom!=NULL)
    {
        if(bottom->id==idS && bottom==htable[index])
        {
            htable[index]=NULL;
            delete bottom; break;
        }
        if(bottom->id!=idS)
        {
            prev=bottom;
            bottom=bottom->next;
        }
        else
```

```
       {
          prev->next=bottom->next;
          delete bottom; break;
       }
   }
}
```

This function is used to delete node in the data structure. The function first determines the index of htable using the hashing function, then traverses the linked list in order to find the node, to be deleted. If found then the previous node’s next is updated and the current node is deleted.

6.

```
void display()
{   hashnode *bottom=NULL;
    for(int i=0;i<nmax;i++)
    {
       if(htable[i]!=NULL)
       {
          bottom=htable[i];
          while(bottom!=NULL)
           {
               cout<<″Name: ″<<bottom->name<<’\n’<<″id: ″;
                <<bottom->id<<’\n’; cout<<″Price:
            ″<<bottom->price<<endl<<″Quantity:
            ″<<bottom->quantity<<″\n\n″;

              bottom=bottom->next;
             }
       }
    }
}
```

This function displays all the elements in the data structure.

6.

```
    void updatefile()
```

This function is used to update the database file, by transferring the contents of the current data structure into the txt file.

7.

```
    int loadfile(hashmap &s)
```

This function is used to load the database file into the data structure, by transferring the contents of the current file into the data structure using the push function.

Shortcomings of the Program

- Lack of GUI

  The program lacks a intuitive User interface, as such can be difficult to operate by someone is not well-versed with computers
- Prone to memory errors

  As the program uses txt file as database it can be prone to memory errors due to misplaced file pointers, thus causes the program to randomly crash at times.
- Lack of a sophisticated and specialized database programs

  The program does not employ modern database solutions such as SQL,oracle,etc. As such editing, maintaining and securing the database can be difficult.
- Longer runtime for operations when number of nodes(items) is increased greatly, the clutter increases in each node of the array(htable), thus takes longer to find and operate on the desired node. Can be decrease by increasing the max size of the array, however at the cost of memory efficiency.

Bibliography

Internet:

https://www.youtube.com/channel/UClEEsT7DkdVO_fkrBw0OTrA
(mycodeschool)

https://www.geeksforgeeks.org/

Books:

Data Structures and Algorithms Made Easy: Data Structures and Algorithmic Puzzles by:
Narasimha Karumanchi

Computer science with c++ by: Sumita Arora