Finding product in nearby store using real time database

Laksh Gupta, Aditya Agrawal

***Abstract*—**
We have made a program that tells about price and quantity of a product in nearby store. For this purpose, we have stored a sample dataset of 4 stores containing 10 products each. We have used binary file handling for storing and accessing the product data. As student, sometimes we need to find out the minimum price of a particular product in nearby stores or sometimes we need to find out if a particular product is available at a particular store or not. Our program will solve this problem by providing them with details they want.
In our program, we have given two options:

1. Backend
2. Frontend

One is useful for the store owner so that they can perform the following operations:

a) Enter a new product
b) Delete a product
c) View Inventory

And the second option is useful for customer. This option asks for the product name. After giving the product name, it displays all the stores which have that product along with its price and quantity in their store.

Literature Survey:
We wanted a way to find out the availability and price of a given product in stores near us, we searched for such apps and websites and found some apps like shopsavvy and smartprix but these apps only compare the price and availability of products available on online stores like amazon, flipkart, snapdeal etc.
We also found that there were some websites that did compare stores but they only included big stores like Walmart and Costco. They didn't include any local and nearby small stores. Also these apps were available in USA and UK, but not in India.

We took some ideas from these apps and websites and tried to execute this for local stores and shops in and near VIT. For example in VIT we have shops like Allmart, enzo and small shops inside campus. If we want to check that the required product is available in these shops or not, there is no way for us to check that. So we came up with a program that helps us solve this problem. We also searched for different ways to maintain and store the database of these stores and concluded that for our needs, a small database can be maintained via binary files. The data can be stored for each shop in the binary files and store each product details as an object of a class.

Binary files:
A binary file is a file stored in binary format. . Binary files differ from a text file as they are computer-readable but not human-readable .Binary files are usually thought of as being a sequence of bytes, which means the binary bits are grouped in eights. Binary files typically contain bytes that are intended to be interpreted as something other than text characters.
Files are a means to store data in a storage device and hence here we use them to maintain our database. C++ file handling provides a mechanism to store output of a program in a file and read from a file on the disk. we use <iostream> header file which provide
functions cin and cout to take input from console and write output to a console respectively. Now, we introduce one more header file <fstream> which provides data types or classes
(ifstream , ofstream , fstream) to read from a file and write to a file.

fstream enables us to perform both reading and writing in a file
Ifstream is for reading
Ofstream is for writing in a binary file

A file can be opened in different modes to perform read and write operations. Function to open a file i.e open() takes two arguments : char *filename and ios :: mode.

In this program we use binary files and we store the details of each product as an object of a class in 4 separate binary files .One for each store in our database.

And we perform the delete, search and read operations on the file.

Proposed model:

We made a c++ code which gives us two options, front-end and back-end

If we go to front end it asks for the product you want to search and searches for the entered product from the maintained database of four stores and displays the store, price and quantity left for the product in all the stores. It there is no such product available in any of the stores, then it displays the same (but there will not be any store name in the output).

In the back-end, it gives the user three options

- Add new product into the inventory
- Delete a product form the inventory
- View inventory

And we can perform these functions in any of the four stores.

We maintained a dataset for 4 stores and each store contains 10 products. Some of the products are overlapping in stores and some products are unique for each store.

The database is as follows:

```
mysql> select *from inventory;
+-------+--------------------+-------+----------+
| Store | Product            | Price | Quantity |
+-------+--------------------+-------+----------+
|     1 | ABC Shampoo        |    70 |       24 |
|     1 | PQR Soap           |    40 |       33 |
|     1 | Dettol             |    50 |       27 |
|     1 | Biscuit            |    10 |      170 |
|     1 | Maggie             |    12 |      220 |
|     1 | Colgate            |    48 |       17 |
|     1 | Surf Excel         |    99 |       69 |
|     1 | Butterflow Pen     |    10 |       54 |
|     1 | Fogg Deo           |   229 |        8 |
|     1 | Lays Orange        |    10 |       88 |
|     2 | Dettol             |    40 |       17 |
|     2 | Maggie             |     8 |       99 |
|     2 | Surf Excel         |   115 |       59 |
|     2 | Lays Orange        |     9 |       27 |
|     2 | Amul Milk          |    37 |       15 |
|     2 | Classmate Notebook |    40 |        9 |
|     2 | Soya Sticks        |    12 |       44 |
|     2 | Milk Shake         |    35 |       11 |
|     2 | Veg Puff           |    10 |       12 |
|     2 | Pepsi              |    35 |        9 |
|     3 | Maggie             |    15 |       29 |
|     3 | Pepsi              |    30 |       40 |
|     3 | Colgate            |    50 |       17 |
|     3 | ABC Shampoo        |    60 |       20 |
|     3 | Amul Milk          |    46 |       25 |
|     3 | Eggs               |     7 |       91 |
|     3 | Eggs Puff          |    14 |       39 |
|     3 | Cake               |   220 |        8 |
|     3 | Basmati Rice       |   710 |       12 |
|     3 | Biscuit            |    10 |        3 |
|     4 | Maggie             |     8 |       54 |
|     4 | Pepsi              |    37 |       30 |
|     4 | Fogg Deo           |   210 |        7 |
|     4 | PQR Soap           |    36 |       19 |
|     4 | Amul Butter        |   115 |       20 |
|     4 | Kissan Jam         |    40 |       31 |
|     4 | Water Bottle       |    18 |       17 |
|     4 | Haldiram Mixture   |    42 |       25 |
|     4 | Nescafe Icetea     |   135 |       11 |
|     4 | Kitkat             |    40 |       49 |
+-------+--------------------+-------+----------+
40 rows in set (0.00 sec)
```

Pseudo code:

Defining class product

{

function getdata to get the data from the user and store it in the class

function display_data to print the data stored in the class

function getname() to access the private data-member name;

function encoder() to convert the strings name and store to an integer array that holds the ascii value of each character in the strings so as to store the correct data in the binary file

```
int i;
for(i=0 ; store[i]!='\0' ; i++)
        codedstore[i]=int(store[i]);
ls=i;
for(i=0 ; name[i]!='\0' ; i++)
```

```
codedname[i]=int(name[i]);
ln=i;
```

function decoder() to convert the integer array that holds the ascii value of each character in the strings to strings so as to display the data to the user

```
int i;
for(i=0 ; i<ls ; i++)
        store[i]=(char)codedstore[i]
for(i=0 ; i<ln ; i++
    name[i]=(char)codedname[i];
```

\\ end of class product

Now we define functions to perform operations on the binary file

Function push(int i) to push the value in store no. i-1

Read the data from the user and store it in an object p of class product

We use if conditions to check the store no. and push it I the respective binary file of the store

```
ofstream out;
out.open("store[i-1].dat",ios::app|ios::binary);
out.write((char*)&p,sizeof(p));
out.close();
```

Function pop(int i) to delete the particular value form store no. i-1

First we enter the product name to be deleted

We use if conditions to check the store no. and delete it form the respective binary file of the store if it exists in the store

```
open file store[i].dat for reading
open file temp.dat for writing
int flag=0;
prod p;\\ to store the data read form the file
while there is data in store[i].dat
{
 Read data form it
Compare it with the data to be deleted
If data is found
flag=1;
else
write  the data in file temp.dat
 if(product was not found )
specified product not found \n";  fd.close(); ft.close();
    remove("store1.dat");
    rename("temp.dat","store1.dat");
     product deleted successfully\n";
```

A function display(int) which takes I as argument and displays the contents of a store i

```
ifstream in("store[i].dat", ios::binary);
while(!in.eof())
        in.read((char*)&p, sizeof(p));
        p.decoder();
        p.display_data();
        in.close()
```

now we declare 4 functions that take the string to be searched and searching for the product in the respective store

```
ifstream fin("store[].dat", ios::in|ios::binary)
 int f=0;
 while(!fin.eof())
{
   fin.read((char *)&p, sizeof(p));
   p.decoder();
   if(!serch.compare(p.getname()))
   {
      p.display_data();
      f=1;
      break;
   }
}
return f;
fin.close();
}
```

Now we define the main() function

int main()

Menu

1)Back-end

2)Front-end

3)Exit

4)Enter your choice

And further options for 1, 2 choices

Result:

We created a GitHub repository for our project and have uploaded the c++ code and necessary binary files there. The link to the repository is: https://github.com/laksh-gupta/file-handling-project

Acknowledgment

This project was made possible by the collective effort of the team. We want to thank Dr. Boominathan P, our mentor for his continuous guidance without which this project wouldn't be possible.