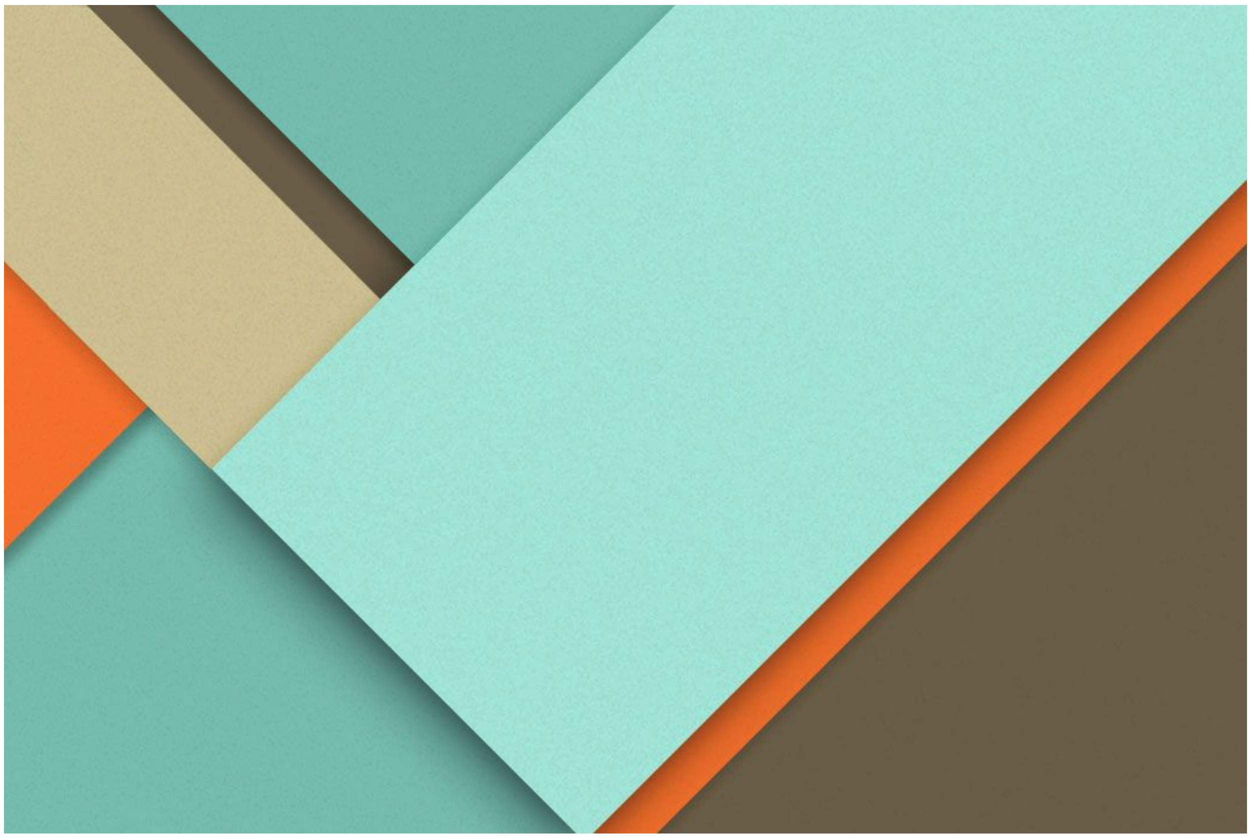


Online Structured Commerce Application

23.11.2017

Group 8
Vishakh S, B16CS038
Zaid Khan, B16CS040
Chinmay Garg, B16CS041
Saksham Banga, B16CS042

Contents

Abstract

This project deals with developing a commerce application for an online retailer. The main aim of this application is to provide a user friendly interface for virtual shopping, wherein the customers can choose to purchase a variety of products available.

The application was designed into two modules, the first one being for the customers who wish to buy the products. Second is for the vendor who can maintain and update the information pertaining to the articles and those of the customers. This application maintains the details of customers, vendors and products while allowing concurrent updates.

Acknowledgement

We would like to express our heartfelt gratitude to our Professor Dr. Anil Shukla for giving us this opportunity to develop this application and for supporting and motivating us at every stage. Our efforts would be incomplete without thanking him for sharing his pearls of wisdom with us during this course.

To Bjarne Stoustrup, for inventing the beautiful language of C++ and giving it selflessly to the world.

To E.Balagurusamy (author of Object Oriented Programming with C++, 6e, McGraw Hill Education) and Bruce Eckel (author of Thinking in C++, 2e, Pearson Publications) whose books, we have extensively used for reference during the course of this project.

To the God Almighty without whose blessings, this project would not have been possible.

Vishakh S, B16CS038

Zaid Khan, B16CS040

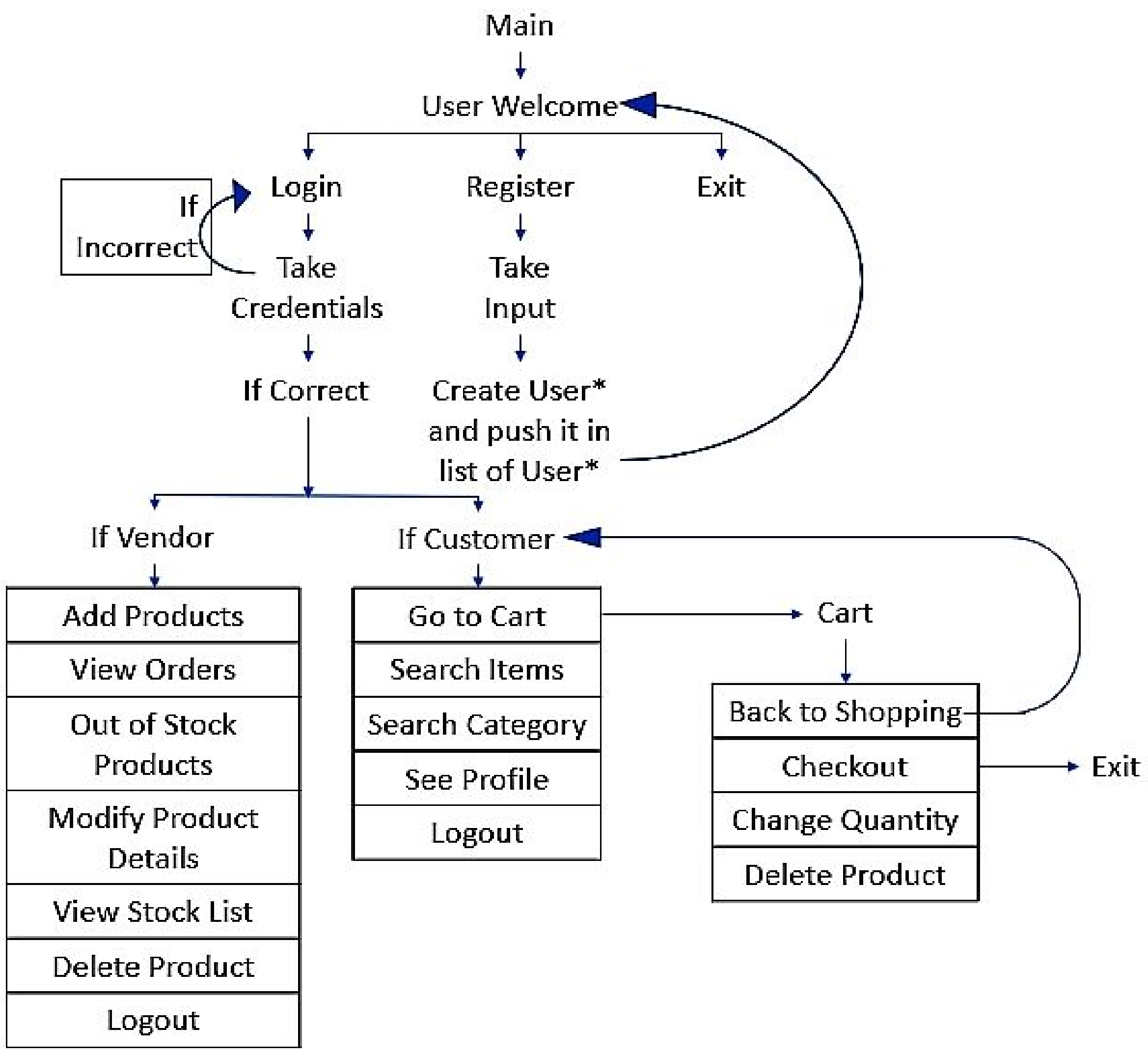
Chinmay Garg, B16CS041

Saksham Banga, B16CS042

Goals

Our goal is to program a form of electronics commerce which allows consumers to directly and conveniently buy goods or services from a seller through an accessible platform. Consumers find a product of interest by visiting the portal and searching it through the stocklist.

Technical Flow



Concepts of OOPs used in the Project

1. Data Abstraction

Abstraction means displaying only essential information and hiding the details. Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.

i. Abstraction using Classes:

We can implement Abstraction in C++ using classes. Class helps us to group data members and member functions using available access specifiers. A Class can decide which data member will be visible to outside world and which is not.

ii. Abstraction in Header files:

One more type of abstraction in C++ can be header files. For example, consider the pow() method present in math.h header file. Whenever we need to calculate power of a number, we simply call the function pow() present in the math.h header file and pass the numbers as arguments without knowing the underlying algorithm according to which the function is actually calculating power of numbers.

iii. Abstraction using access specifiers:

Access specifiers are the main pillar of implementing abstraction in C++. We can use access specifiers to enforce restrictions on class members. For example:

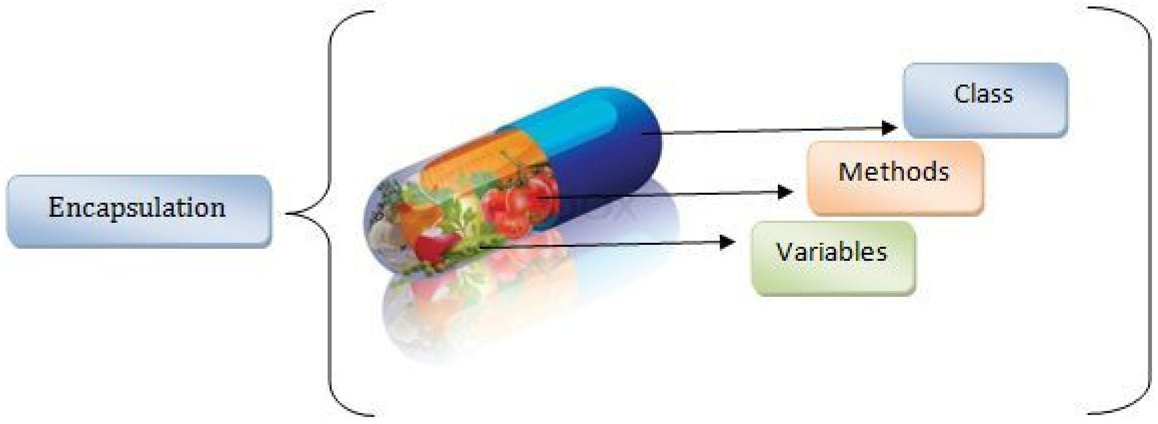
- Members declared as public in a class, can be accessed from anywhere in the program.
- Members declared as protected can be accessed only within the class and within the derived classes of this class.
- Members declared as private in a class, can be accessed only from within the class. They are not allowed to be accessed from any part of code outside the class.

We can easily implement abstraction using the above two features provided by access specifiers. Say, the members that defines the internal implementation can be marked as private in a class. And the important information needed to be given to the outside world can be marked as public. And these public members can access the private members as they are inside the class.

Consider an example of abstraction used in our project*. A consumer only knows and uses the functionalities provided by the programmer to him. He is unaware of the internal implementation of it. This is what abstraction is.*

2. Encapsulation

Encapsulation is defined as wrapping up of data and information under a single unit. In Object Oriented Programming, Encapsulation is defined as binding together the data and the functions that manipulates them.



- Consider a real life example of encapsulation, in a company there are different sections like the accounts section, finance section, sales section etc. The finance section handles all the financial transactions and keep records of all the data related to finance. Similarly the sales section handles all the sales related activities and keep records of all the sales. Now there may arise a situation when for some reason an official from finance section needs all the data about sales in a particular month. In this case, he is not allowed to directly access the data of sales section. He will first have to contact some other officer in the sales section and then request him to give the particular data. This is what encapsulation is. Here the data of sales section and the employees that can manipulate them are wrapped under a single name “sales section”.

Encapsulation also lead to data abstraction or hiding. As using encapsulation also hides the data. In the above example the data of any of the section like sales, finance or accounts is hidden from any other section.

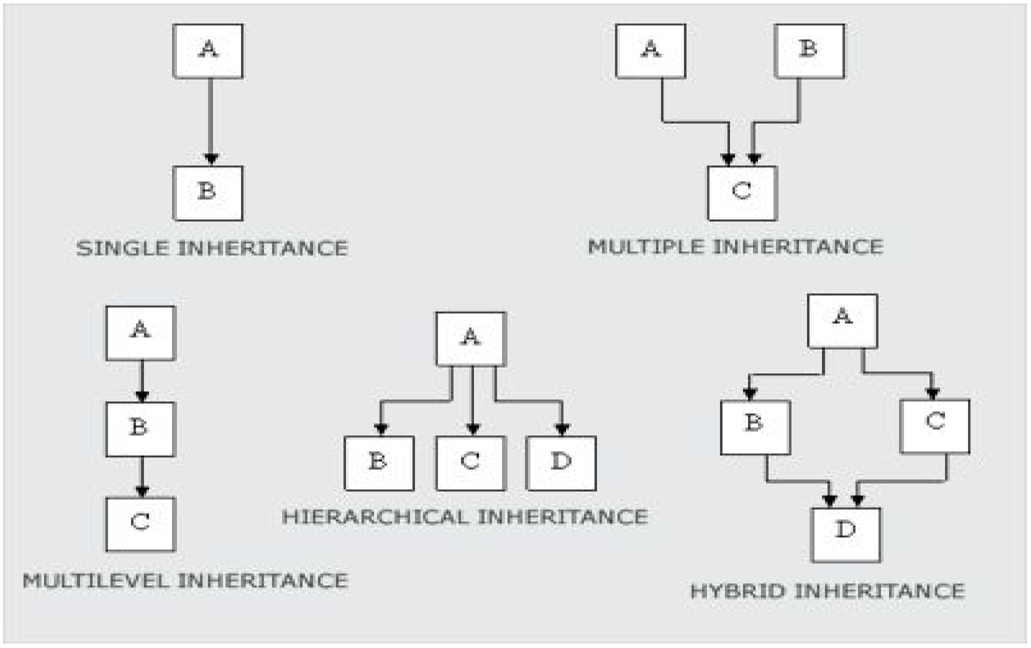
In C++ encapsulation can be implemented using Class and access modifiers.

3. Inheritance

Inheritance is the process by which objects of one class acquire the properties of objects of another class. It supports the concept of hierarchical classification. Inheritance provides reusability. This means that we can add additional features to an existing class without modifying it.

Considering HumanBeing a class, which has properties like hands, legs, eyes etc, and functions like walk, talk, eat, see etc. Male and Female are also classes, but most of the properties and functions are included in HumanBeing, hence they can inherit everything from class HumanBeing using the concept of Inheritance.

Shown below are the different types of Inheritance.



4. Polymorphism

Polymorphism means ability to take more than one form. An operation may exhibit different behaviors in different instances. The behavior depends upon the types of data used in the operation.

C++ supports operator overloading and function overloading.

Operator overloading is the process of making an operator to exhibit different behaviors in different instances is known as operator overloading.

Function overloading is using a single function name to perform different types of tasks.

Polymorphism is extensively used in implementing inheritance.

If we walk using our hands, and not legs, here we will change the parts used to perform something. Hence this is called **Overloading**.

And if there is a defined way of walking, but I wish to walk differently, but using my legs, like everyone else. Then I can walk like I want, this will be called as **Overriding**.

5. Object-Oriented Messaging

Messaging is how work gets done in an OO system. Understanding messaging is a key part of being able to visualize how an OO program actually executes, and the relationship between the abstractions (objects) in an OO program.

A message has four parts:

- identity of the recipient object
- code to be executed by the recipient
- arguments for the code
- return value

The code to be executed by an object when it is sent a message is known variously as a **method**, or a ***function***. Method is preferable.

Messaging an object may cause it to change state. The Elevator object will move when it is otherwise idle, and a user presses a floor button. But note carefully that what an object does when messaged depends on the state it was in when messaged. Thus the state of an object is a consequence of the history of messages it has received.

Objects messaging each other are what gets work done in an OO system. Rumbaugh says that two objects which can message each other have a "link" between them.

Remember, an object can't message another object without first having its name. So, how does an object find out the name of an object it wants to message?

- Name is global
- Name is part of the class of the messaging object
- Name is passed as a parameter to some operation
- Name is locally declared within object's method
- Name is provided as return value from message

How objects relate to each other depends on how they collaborate.

- **Actor** operates on other objects, but is never operated upon.
- **Server** is operated on by other objects, but never operates on other objects.
- **Agent** both and actor and a server.

6. Modularity

Modularity is closely tied with encapsulation; think of modularity as a way of mapping encapsulated abstractions into real, physical modules.

Modularity refers to the concept of making multiple modules first and then linking and combining them to form a complete system. Modularity enables re-usability and minimizes duplication. In addition to re-usability, modularity also makes it easier to fix problems as bugs can be traced to specific system modules, thus limiting the scope of detailed error searching.

The C/C++ convention is to create two files for each class: a header file (.h suffix) for the class interface, and an implementation file (.c, .cp, .cpp, .C suffix) for the code of the class.

Booch gives two goals for defining modules. Make a module cohesive (shared data structures, similar classes) with an interface that allows for minimal inter-module coupling.

Other considerations: teamwork, security, documentation.

Important to remember that the decisions concerning modularity are more physical issues, whereas the encapsulation of abstractions are logical issues of design.

It is possible to "over modularize". This can increase documentation costs and make it hard to find information.

Data Structures Used

1. **Map**

   Maps are associative containers that store elements in a mapped fashion. Each element has a key value and a mapped value. No two mapped values can have same key values.

2. **Pair**

   This class couples together a pair of values, which may be of different types (T1 and T2). The individual values can be accessed through its public members first and second.

3. **Vector**

   Vectors are sequence containers representing arrays that can change in size.

4. **Array**

   An array is a series of elements of the same type placed in contiguous memory locations that can be individually referenced by adding an index to a unique identifier.

5. **String**

   Strings are objects that represent sequences of characters.

6. **Enumeration**

   An *enumeration* is a distinct type whose value is restricted to a range of values (see below for details), which may include several explicitly named constants ("*enumerators*"). The values of the constants are values of an integral type known as the *underlying type* of the enumeration.

Major Tools Used

- Functions
- Classes and Objects
- Constructors and Destructors
- Operator overloading
- Inheritance
- Pointers, Virtual Functions and Polymorphism
- Standard Template Library
- Namespace Scope

The Five Pillars of our Code

1. User
2. Vendor
3. Customer
4. Stocklist
5. Cart

chorbazar
USER
VENDOR
CUSTOMER
STOCKLIST
CART

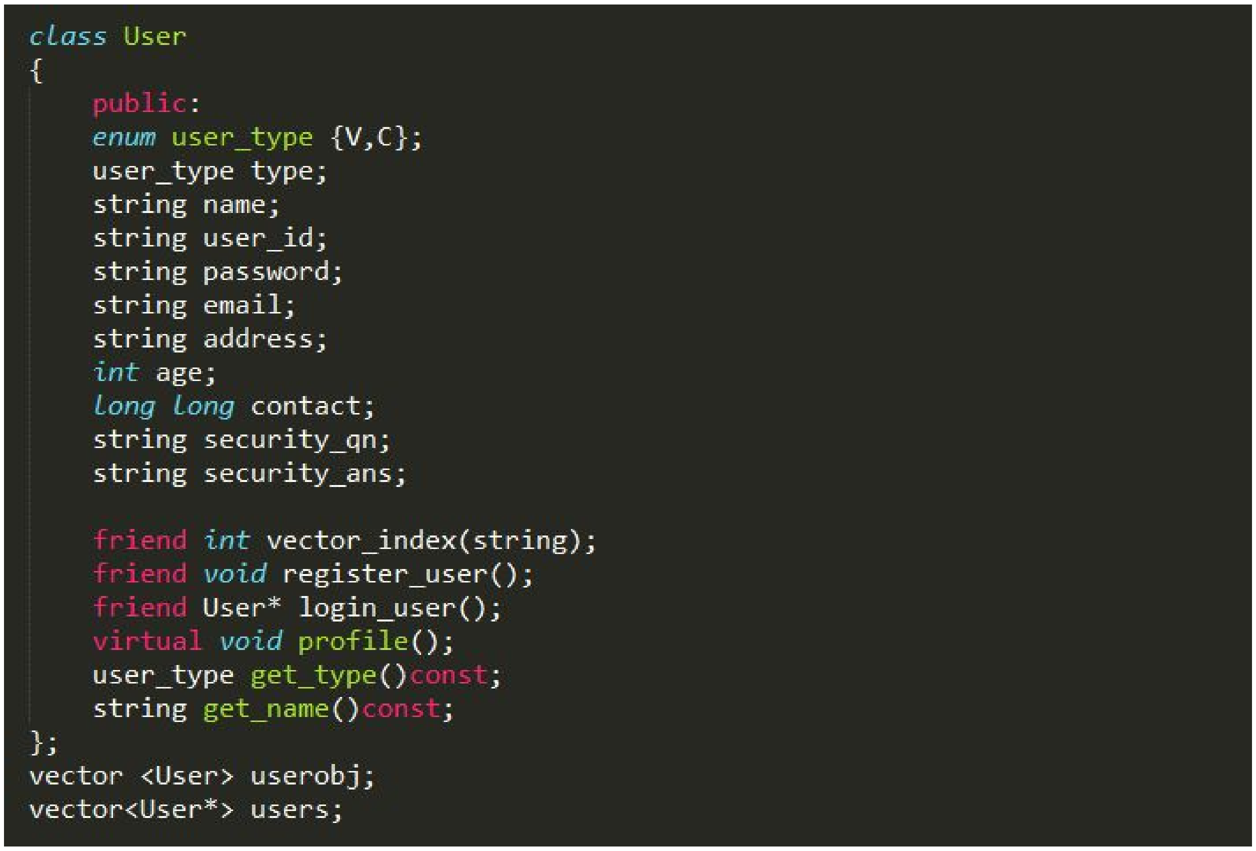
1. USER

Overview

This section deals with the class User. This class has in it defined, the basic information about any user,.i.e. both vendor and customer. The User class is inherited by both Vendor and Customer classes. Every user is stored in a vector of User class Pointers.

Information stored in User class

1. User type, a type field indicating whether he/she is a vendor or customer.
2. Name of the user.
3. User's user_id for unique identification.
4. Password for user's account.
5. Email address of the user.
6. Shipping address of the user.
7. Age of the user.
8. Contact number of the user.
9. Security question and its answer for the user's account, in case he/she forgets the password to his/her account.



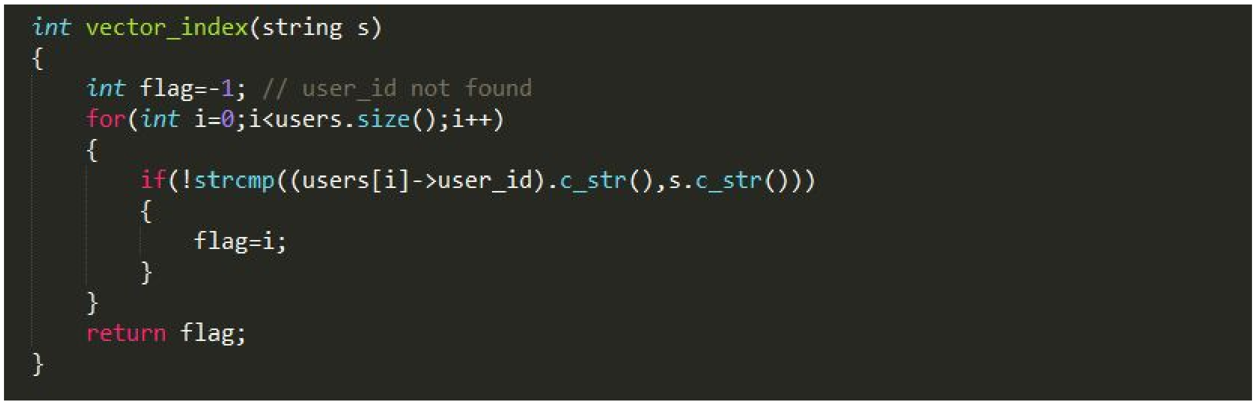
```
class User
{
    public:
    enum user_type {V,C};
    user_type type;
    string name;
    string user_id;
    string password;
    string email;
    string address;
    int age;
    long long contact;
    string security_qn;
    string security_ans;

    friend int vector_index(string);
    friend void register_user();
    friend User* login_user();
    virtual void profile();
    user_type get_type()const;
    string get_name()const;
};
vector <User> userobj;
vector<User*> users;
```

Functions declared for the User

A. Inside User Class

1. friend int vector_index(string);

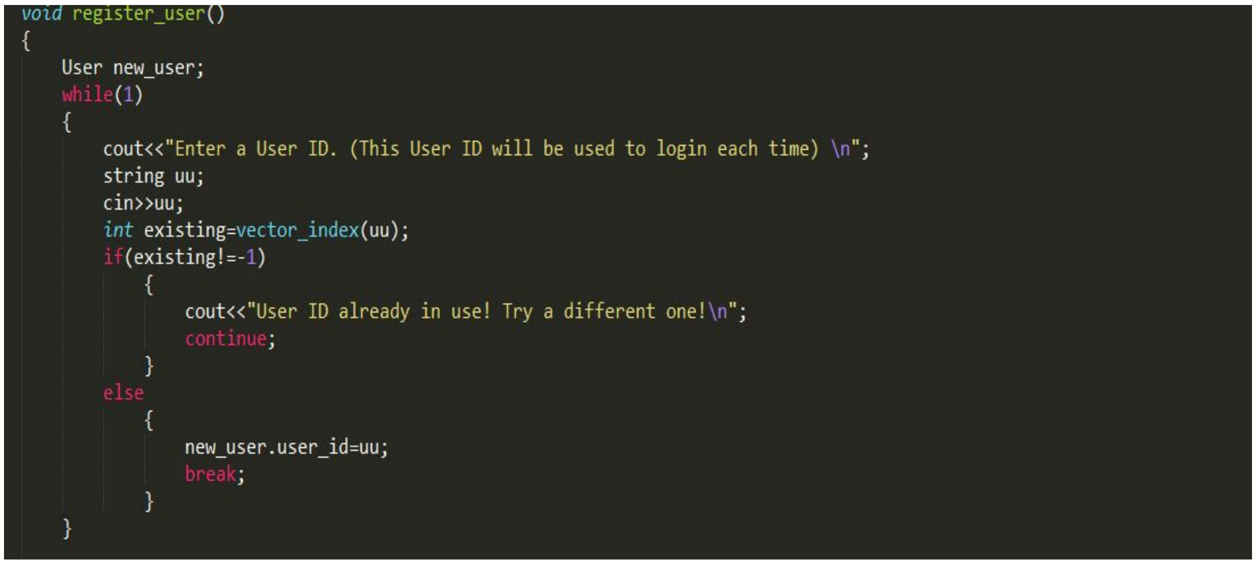
It checks whether the string (new user's user_id) given as input is already in use with a different account or not. If it is not in use, then the function returns -1, else it returns the value of the index where it stored in the vector of users.



```
int vector_index(string s)
{
    int flag=-1; // user_id not found
    for(int i=0;i<users.size();i++)
    {
        if(!strcmp((users[i]->user_id).c_str(),s.c_str()))
        {
            flag=i;
        }
    }
    return flag;
}
```

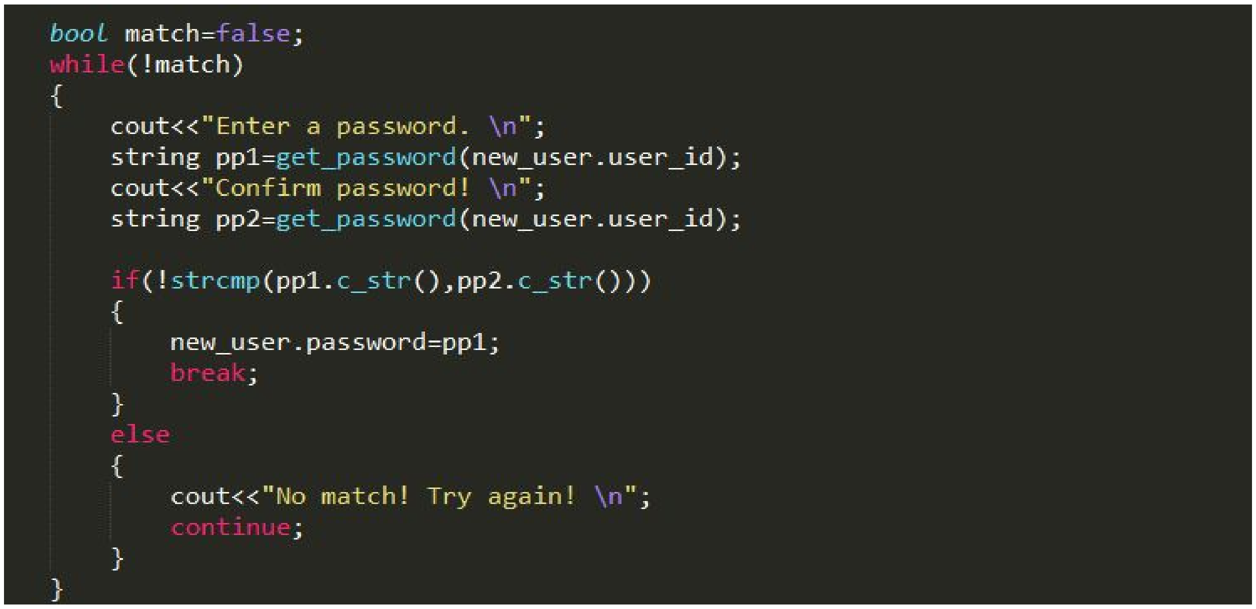
2. friend void register_user();

As the name suggests it registers a new user. It is the function which accepts values to the different fields required for registration. It also ensures that no two users have same user_id. It also calls the Captcha() function in order to ensure that the registration is being done manually and not by a robot.



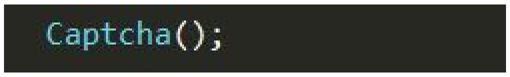
```
void register_user()
{
    User new_user;
    while(1)
    {
        cout<<"Enter a User ID. (This User ID will be used to login each time) \n";
        string uu;
        cin>>uu;
        int existing=vector_index(uu);
        if(existing!=-1)
            {
                cout<<"User ID already in use! Try a different one!\n";
                continue;
            }
        else
            {
                new_user.user_id=uu;
                break;
            }
    }
```

Accepting a user_id and checking if it is already in use



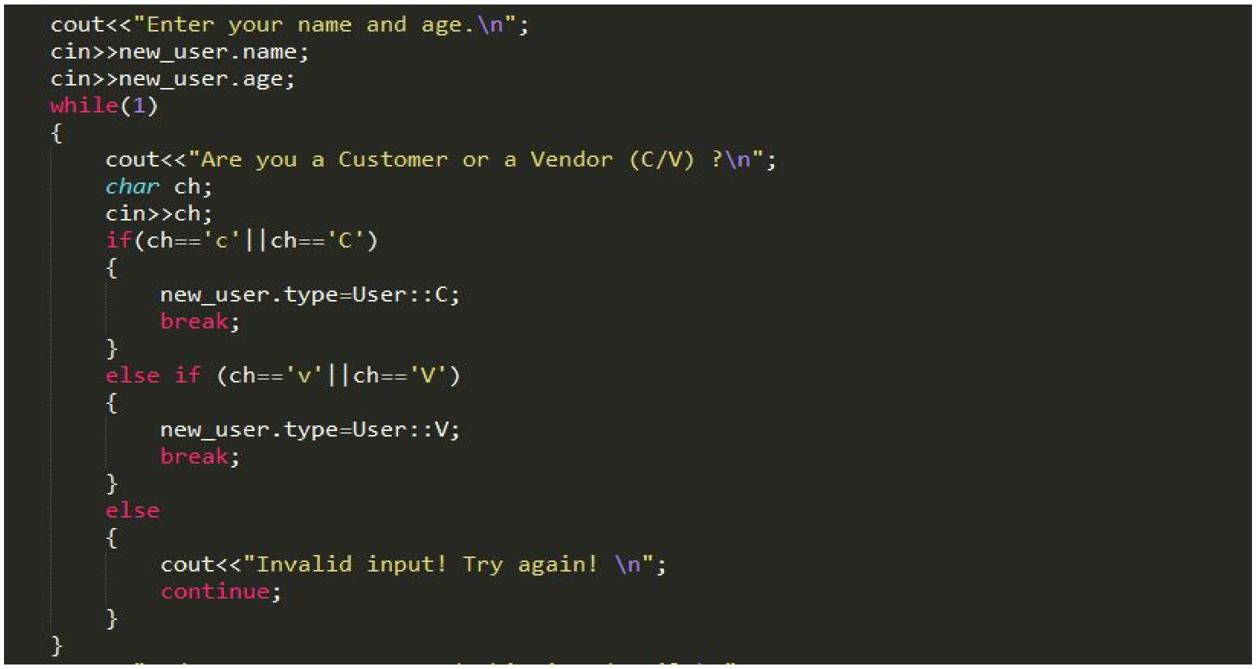
```
bool match=false;
while(!match)
{
    cout<<"Enter a password. \n";
    string pp1=get_password(new_user.user_id);
    cout<<"Confirm password! \n";
    string pp2=get_password(new_user.user_id);

    if(!strcmp(pp1.c_str(),pp2.c_str()))
    {
        new_user.password=pp1;
        break;
    }
    else
    {
        cout<<"No match! Try again! \n";
        continue;
    }
}
```

Accept a password and make the user retype it for avoiding typographical errors



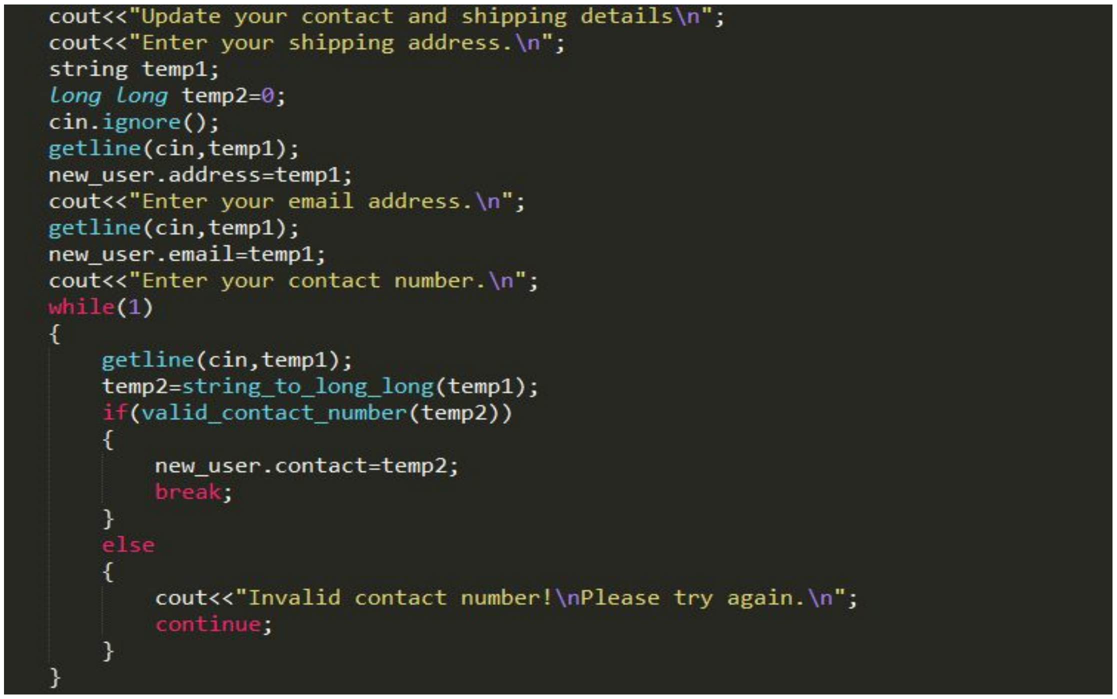
```
Captcha();
```

Captcha verification



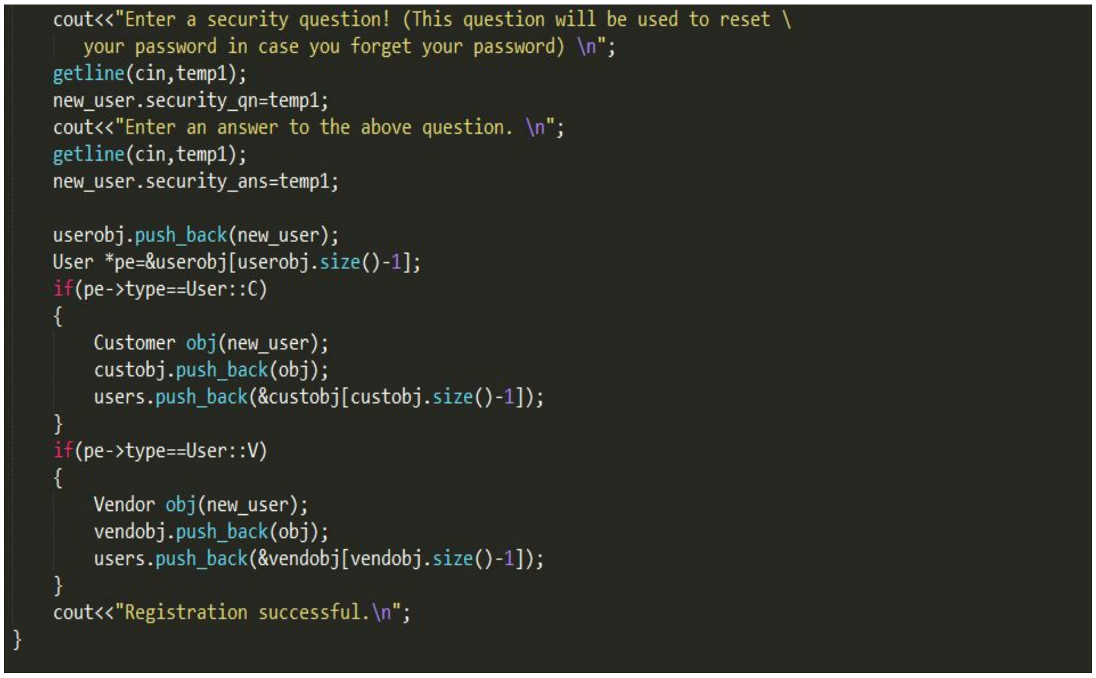
```
cout<<"Enter your name and age.\n";
cin>>new_user.name;
cin>>new_user.age;
while(1)
{
    cout<<"Are you a Customer or a Vendor (C/V) ?\n";
    char ch;
    cin>>ch;
    if(ch=='c'||ch=='C')
    {
        new_user.type=User::C;
        break;
    }
    else if (ch=='v'||ch=='V')
    {
        new_user.type=User::V;
        break;
    }
    else
    {
        cout<<"Invalid input! Try again! \n";
        continue;
    }
}
```

Setting the type field



```
    cout<<"Update your contact and shipping details\n";
    cout<<"Enter your shipping address.\n";
    string temp1;
    long long temp2=0;
    cin.ignore();
    getline(cin,temp1);
    new_user.address=temp1;
    cout<<"Enter your email address.\n";
    getline(cin,temp1);
    new_user.email=temp1;
    cout<<"Enter your contact number.\n";
    while(1)
    {
        getline(cin,temp1);
        temp2=string_to_long_long(temp1);
        if(valid_contact_number(temp2))
        {
            new_user.contact=temp2;
            break;
        }
        else
        {
            cout<<"Invalid contact number!\nPlease try again.\n";
            continue;
        }
    }
```

Other details

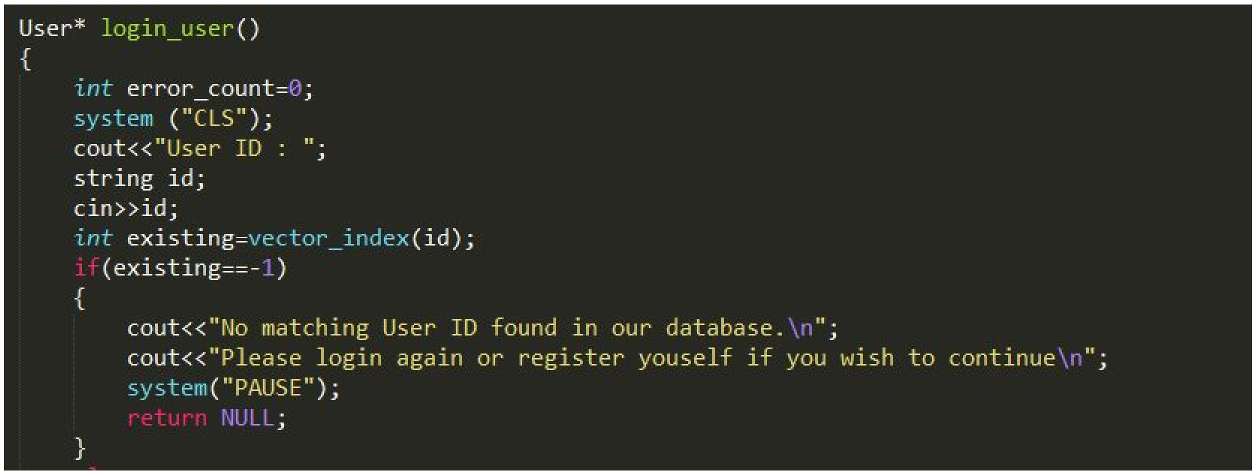


```
    cout<<"Enter a security question! (This question will be used to reset \
       your password in case you forget your password) \n";
    getline(cin,temp1);
    new_user.security_qn=temp1;
    cout<<"Enter an answer to the above question. \n";
    getline(cin,temp1);
    new_user.security_ans=temp1;

    userobj.push_back(new_user);
    User *pe=&userobj[userobj.size()-1];
    if(pe->type==User::C)
    {
        Customer obj(new_user);
        custobj.push_back(obj);
        users.push_back(&custobj[custobj.size()-1]);
    }
    if(pe->type==User::V)
    {
        Vendor obj(new_user);
        vendobj.push_back(obj);
        users.push_back(&vendobj[vendobj.size()-1]);
    }
    cout<<"Registration successful.\n";
}
```

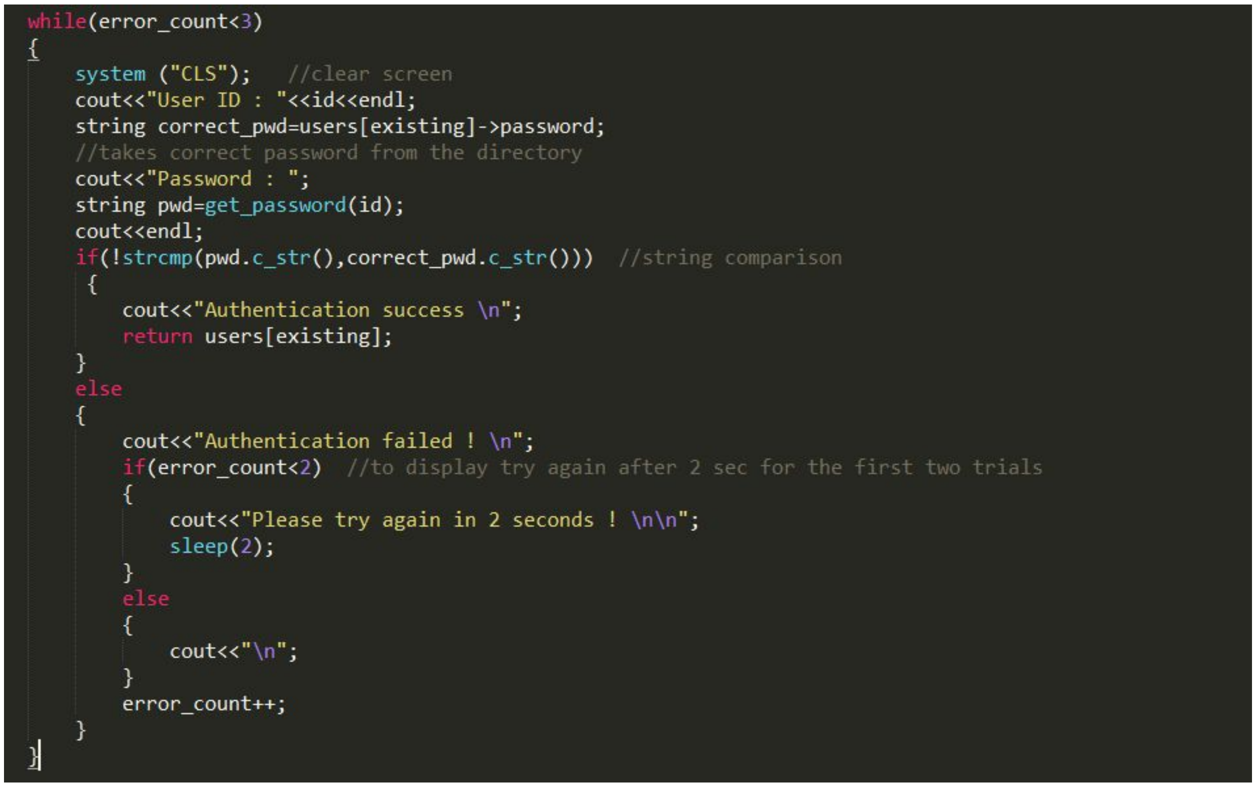
Pushing into the vector in case of successful registration

3. friend User* login_user();

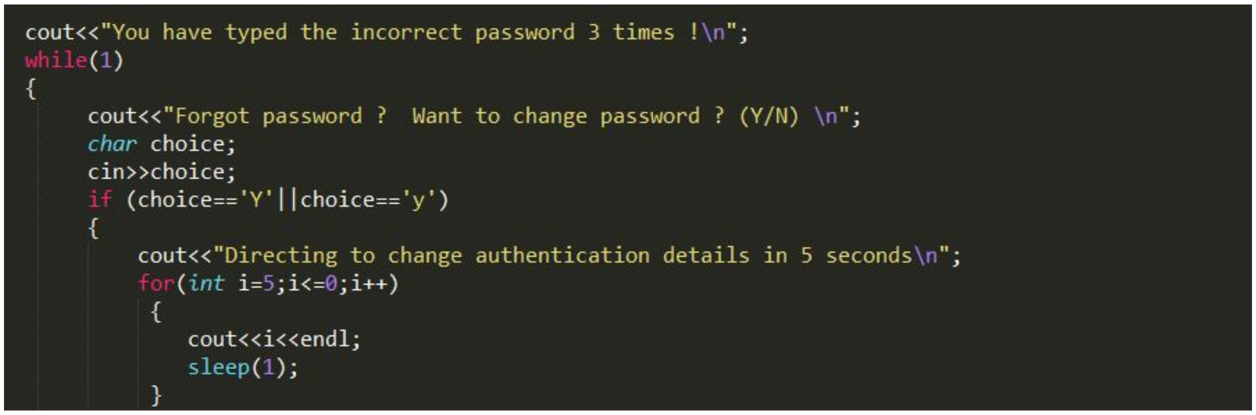
It is the function called before accessing the stocklist. It logins the user by matching the inputs he gives with those he entered at the time of registration. In case he has forgotten the password, the function gives him a chance to change the password if he answers the security question correctly.



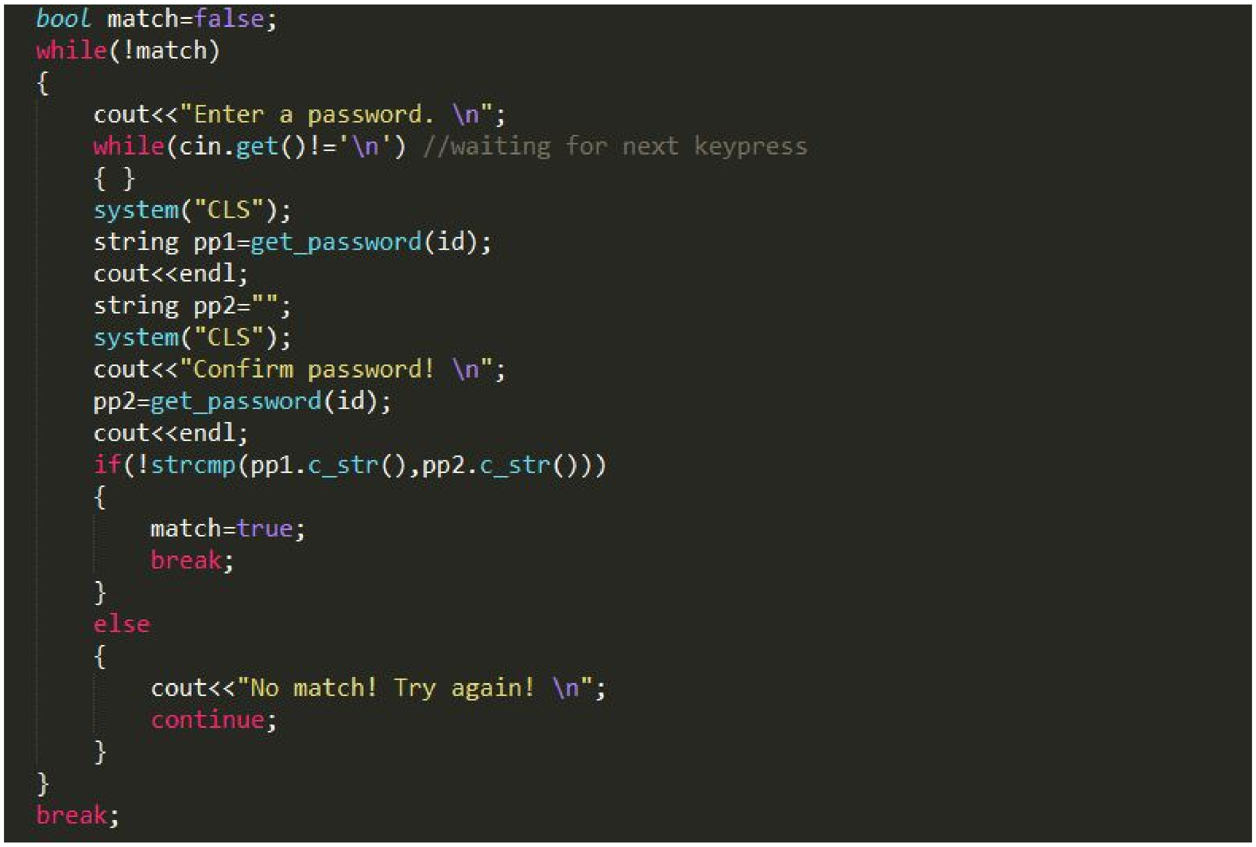
```
User* login_user()
{
    int error_count=0;
    system ("CLS");
    cout<<"User ID : ";
    string id;
    cin>>id;
    int existing=vector_index(id);
    if(existing==-1)
    {
        cout<<"No matching User ID found in our database.\n";
        cout<<"Please login again or register youself if you wish to continue\n";
        system("PAUSE");
        return NULL;
    }
```



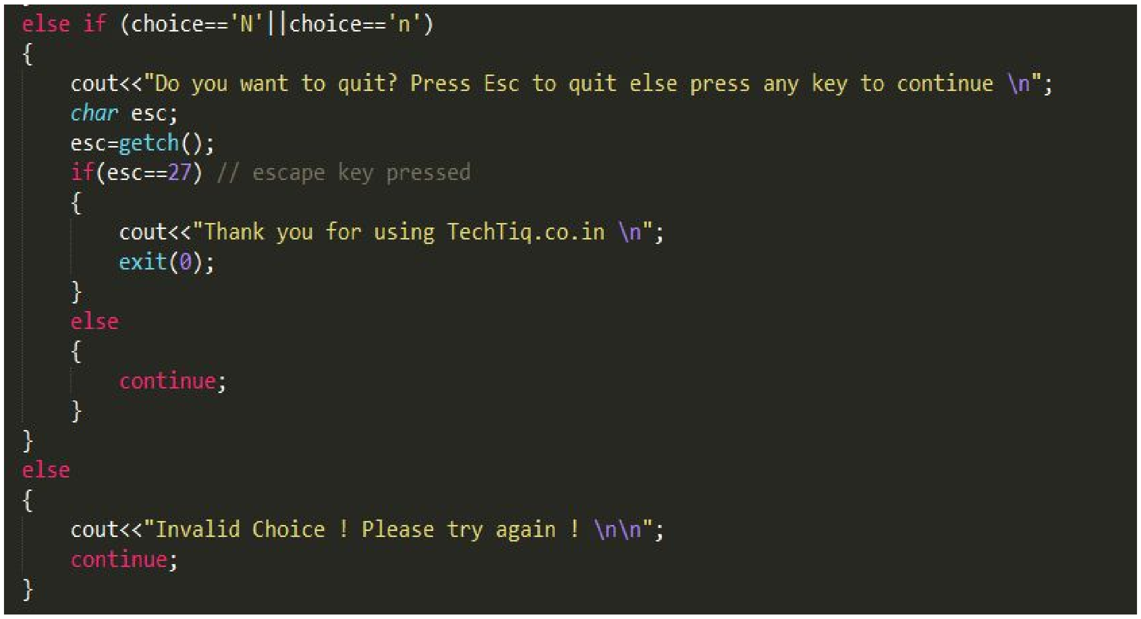
```
while(error_count<3)
{
    system ("CLS");   //clear screen
    cout<<"User ID : "<<id<<endl;
    string correct_pwd=users[existing]->password;
    //takes correct password from the directory
    cout<<"Password : ";
    string pwd=get_password(id);
    cout<<endl;
    if(!strcmp(pwd.c_str(),correct_pwd.c_str()))  //string comparison
     {
        cout<<"Authentication success \n";
        return users[existing];
    }
    else
    {
        cout<<"Authentication failed ! \n";
        if(error_count<2)  //to display try again after 2 sec for the first two trials
        {
            cout<<"Please try again in 2 seconds ! \n\n";
            sleep(2);
        }
        else
        {
            cout<<"\n";
        }
        error_count++;
    }
}
```



```
cout<<"You have typed the incorrect password 3 times !\n";
while(1)
{
     cout<<"Forgot password ?  Want to change password ? (Y/N) \n";
     char choice;
     cin>>choice;
     if (choice=='Y'||choice=='y')
     {
         cout<<"Directing to change authentication details in 5 seconds\n";
         for(int i=5;i<=0;i++)
          {
             cout<<i<<endl;
             sleep(1);
          }
```



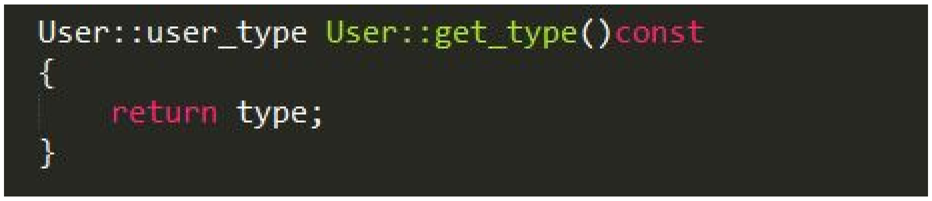
```
bool match=false;
while(!match)
{
    cout<<"Enter a password. \n";
    while(cin.get()!='\n') //waiting for next keypress
    { }
    system("CLS");
    string pp1=get_password(id);
    cout<<endl;
    string pp2="";
    system("CLS");
    cout<<"Confirm password! \n";
    pp2=get_password(id);
    cout<<endl;
    if(!strcmp(pp1.c_str(),pp2.c_str()))
    {
        match=true;
        break;
    }
    else
    {
        cout<<"No match! Try again! \n";
        continue;
    }
}
break;
```



```
else if (choice=='N'||choice=='n')
{
    cout<<"Do you want to quit? Press Esc to quit else press any key to continue \n";
    char esc;
    esc=getch();
    if(esc==27) // escape key pressed
    {
        cout<<"Thank you for using TechTiq.co.in \n";
        exit(0);
    }
    else
    {
        continue;
    }
}
else
{
    cout<<"Invalid Choice ! Please try again ! \n\n";
    continue;
}
```

4. user_type get_type() const;

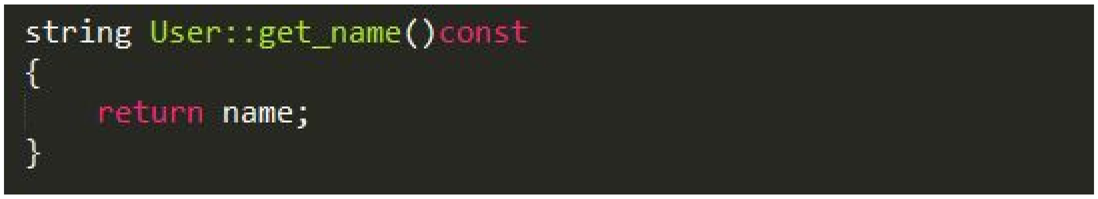
This function returns the type of the user, i.e., whether he is a vendor or a customer.



```
User::user_type User::get_type()const
{
    return type;
}
```

5. string get_name()const

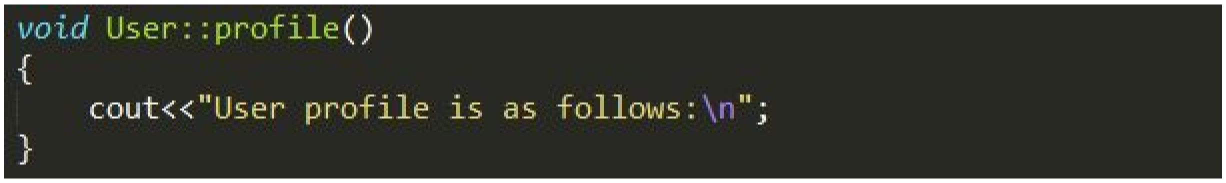
This is a getter function which return the name of the user.



```
string User::get_name()const
{
    return name;
}
```

6. virtual void profile();

This is a virtual function which which will be redefined further in the derived classes of the User class.

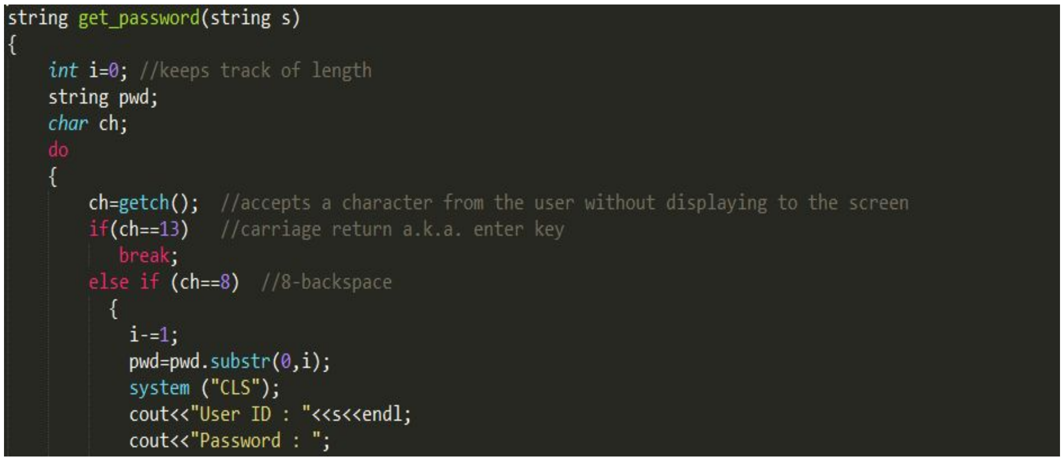


```
void User::profile()
{
    cout<<"User profile is as follows:\n";
}
```

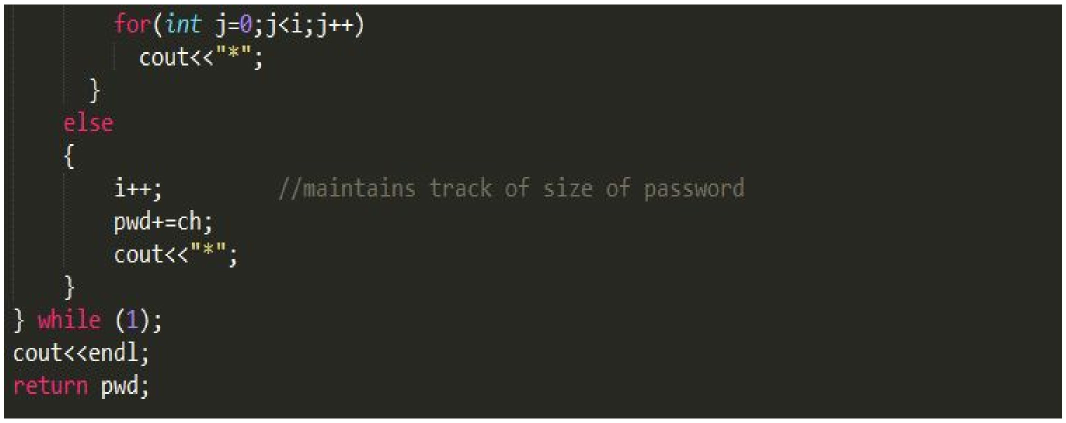
B. Outside User Class

1. string get_password(string s);

This function accepts the input character by character from the user without displaying them to the screen. Here, s is the user id which is passed to print the user_id again to the screen in the event of a backspace press.



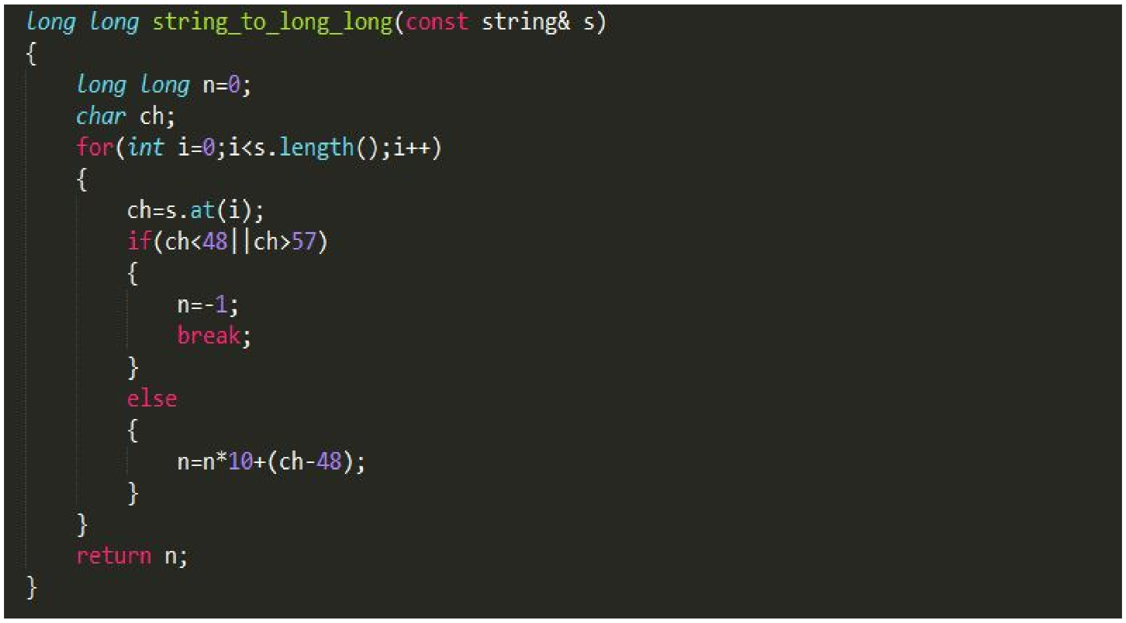
```
string get_password(string s)
{
    int i=0; //keeps track of length
    string pwd;
    char ch;
    do
    {
        ch=getch();  //accepts a character from the user without displaying to the screen
        if(ch==13)   //carriage return a.k.a. enter key
           break;
        else if (ch==8)  //8-backspace
          {
            i-=1;
            pwd=pwd.substr(0,i);
            system ("CLS");
            cout<<"User ID : "<<s<<endl;
            cout<<"Password : ";
```



```
            for(int j=0;j<i;j++)
              cout<<"*";
          }
        else
        {
            i++;          //maintains track of size of password
            pwd+=ch;
            cout<<"*";
        }
} while (1);
cout<<endl;
return pwd;
```

2. long long string_to_long_long(const string& s);

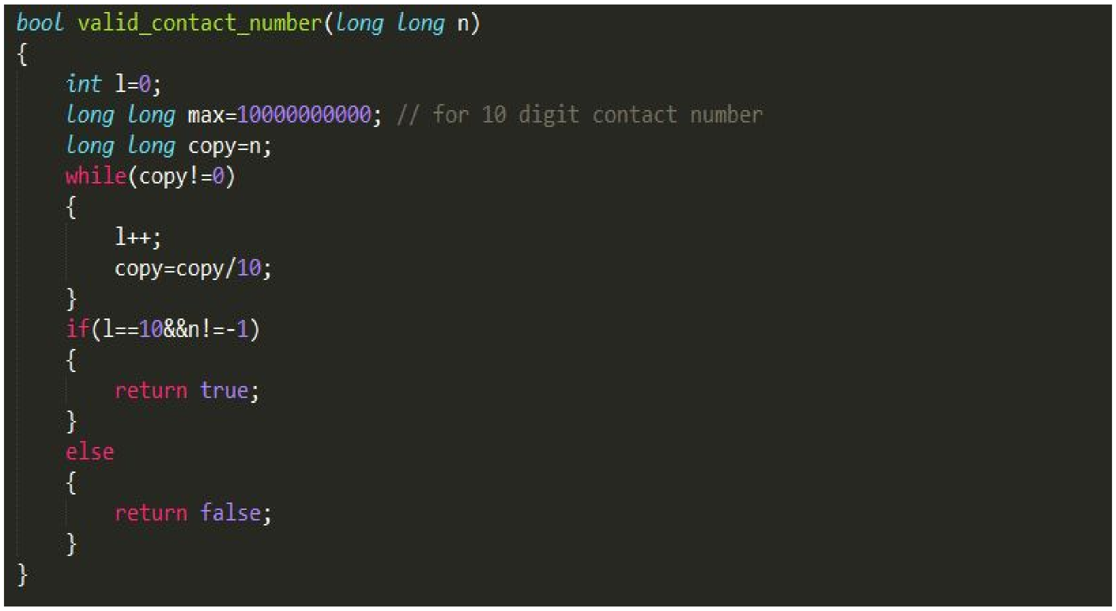
This is the function to convert the string to long long. This function returns a -1 if the user inputs an invalid number (i.e., if some character other than a digit occurs in the string), else it returns the contact number (which was stored in the function).



```
long long string_to_long_long(const string& s)
{
    long long n=0;
    char ch;
    for(int i=0;i<s.length();i++)
    {
        ch=s.at(i);
        if(ch<48||ch>57)
        {
            n=-1;
            break;
        }
        else
        {
            n=n*10+(ch-48);
        }
    }
    return n;
}
```

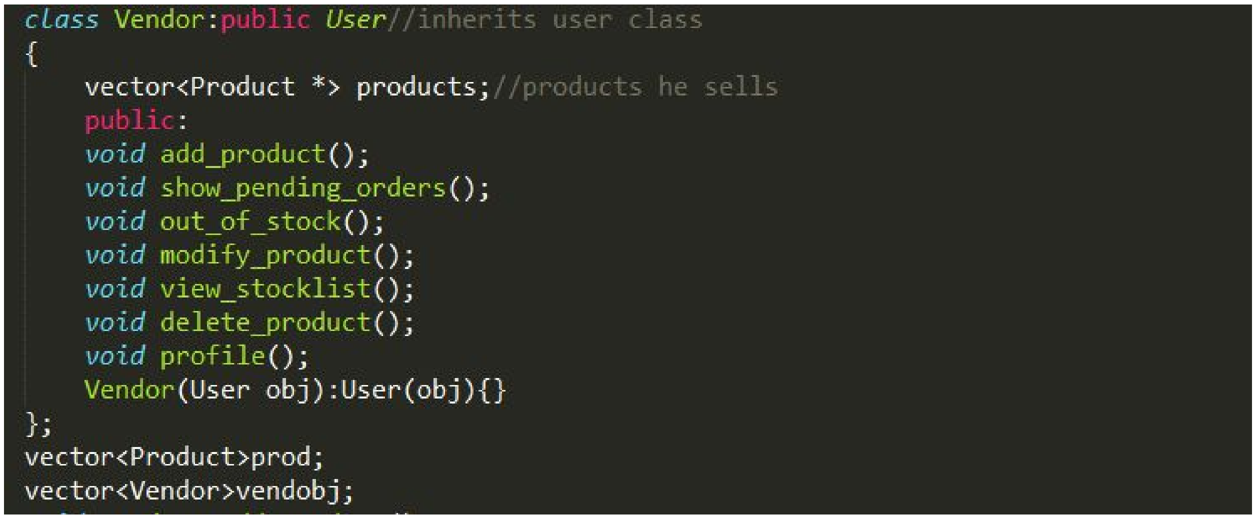
3. bool valid_contact_number(long long n);

It checks if the contact number is valid or not. The code is valid only if it has exactly 10 digits.



```
bool valid_contact_number(long long n)
{
    int l=0;
    long long max=10000000000; // for 10 digit contact number
    long long copy=n;
    while(copy!=0)
    {
        l++;
        copy=copy/10;
    }
    if(l==10&&n!=-1)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

2. VENDOR



```cpp
class Vendor:public User//inherits user class
{
    vector<Product *> products;//products he sells
    public:
    void add_product();
    void show_pending_orders();
    void out_of_stock();
    void modify_product();
    void view_stocklist();
    void delete_product();
    void profile();
    Vendor(User obj):User(obj){}
};
vector<Product>prod;
vector<Vendor>vendobj;
```

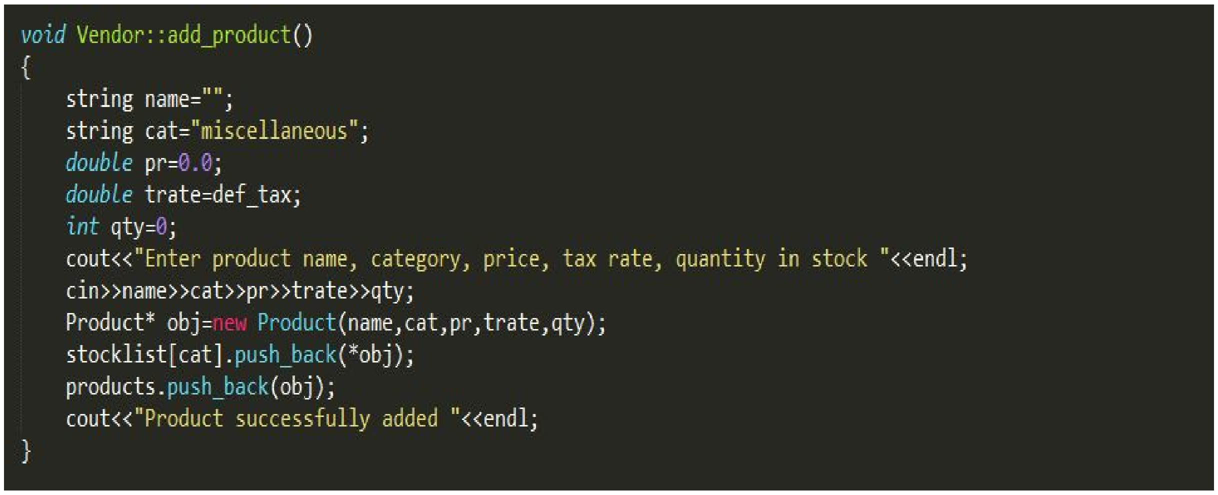
Overview

All the vendor operations can be summarised under one class, i.e., the Vendor class. A vendor inherits the User class.

Functions for Vendor class

1. **void add_product**

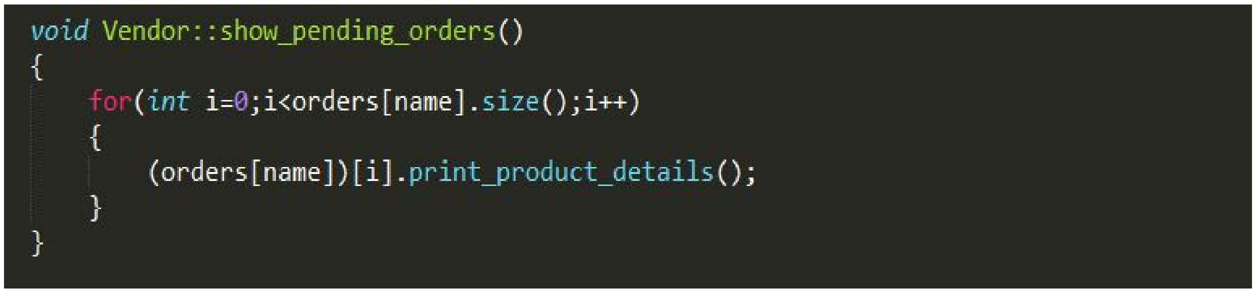
   Basic function to add products to the stocklist and simultaneously add them to the Vector of pointer to product objects.



```cpp
void Vendor::add_product()
{
    string name="";
    string cat="miscellaneous";
    double pr=0.0;
    double trate=def_tax;
    int qty=0;
    cout<<"Enter product name, category, price, tax rate, quantity in stock "<<endl;
    cin>>name>>cat>>pr>>trate>>qty;
    Product* obj=new Product(name,cat,pr,trate,qty);
    stocklist[cat].push_back(*obj);
    products.push_back(obj);
    cout<<"Product successfully added "<<endl;
}
```

2. **void show_pending_orders();**

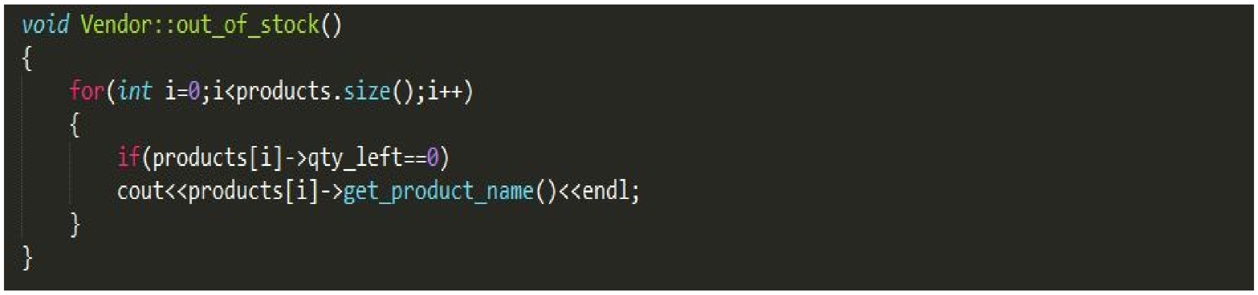
Prints the list of pending orders, the vendor has to deliver



```
void Vendor::show_pending_orders()
{
    for(int i=0;i<orders[name].size();i++)
    {
        (orders[name])[i].print_product_details();
    }
}
```

3. **void out_of_stock();**

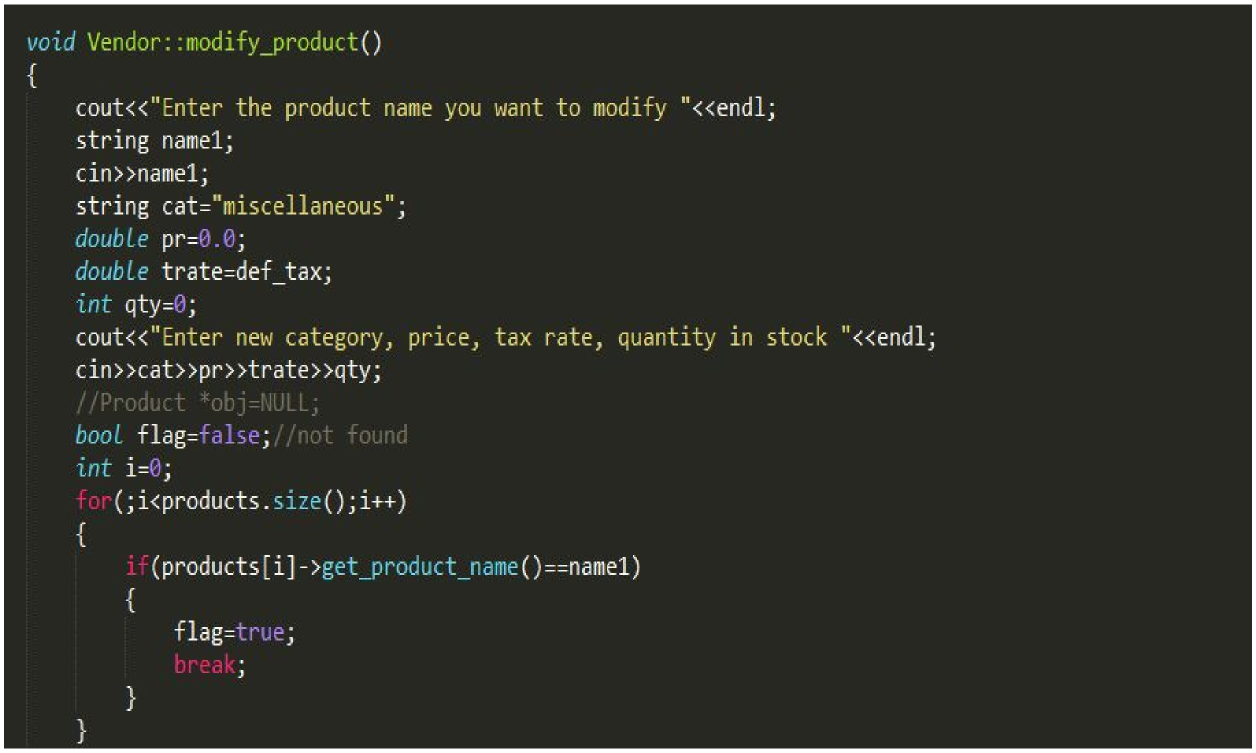
If the quantity of a product is zero, print it.



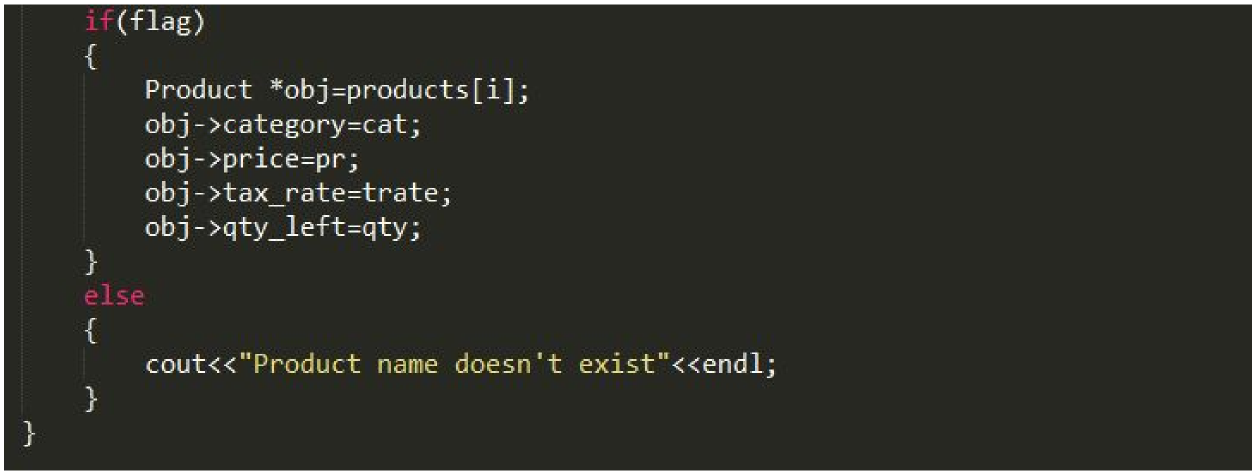
```
void Vendor::out_of_stock()
{
    for(int i=0;i<products.size();i++)
    {
        if(products[i]->qty_left==0)
        cout<<products[i]->get_product_name()<<endl;
    }
}
```

4. void modify_product();

Modifies product details

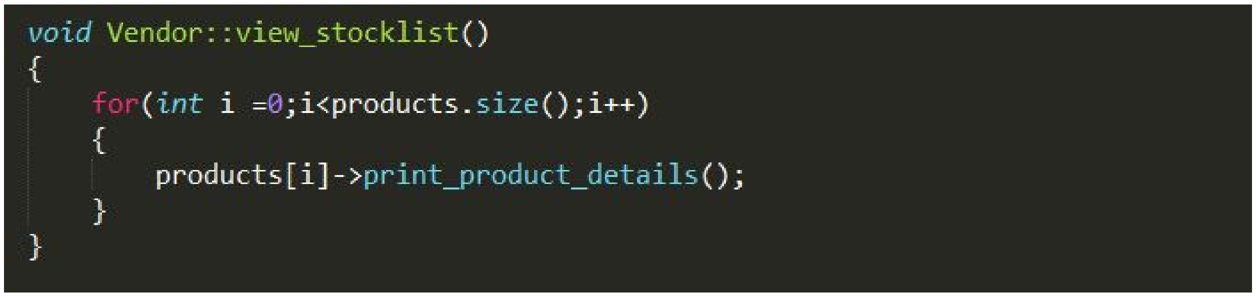


```cpp
void Vendor::modify_product()
{
    cout<<"Enter the product name you want to modify "<<endl;
    string name1;
    cin>>name1;
    string cat="miscellaneous";
    double pr=0.0;
    double trate=def_tax;
    int qty=0;
    cout<<"Enter new category, price, tax rate, quantity in stock "<<endl;
    cin>>cat>>pr>>trate>>qty;
    //Product *obj=NULL;
    bool flag=false;//not found
    int i=0;
    for(;i<products.size();i++)
    {
        if(products[i]->get_product_name()==name1)
        {
            flag=true;
            break;
        }
    }
```



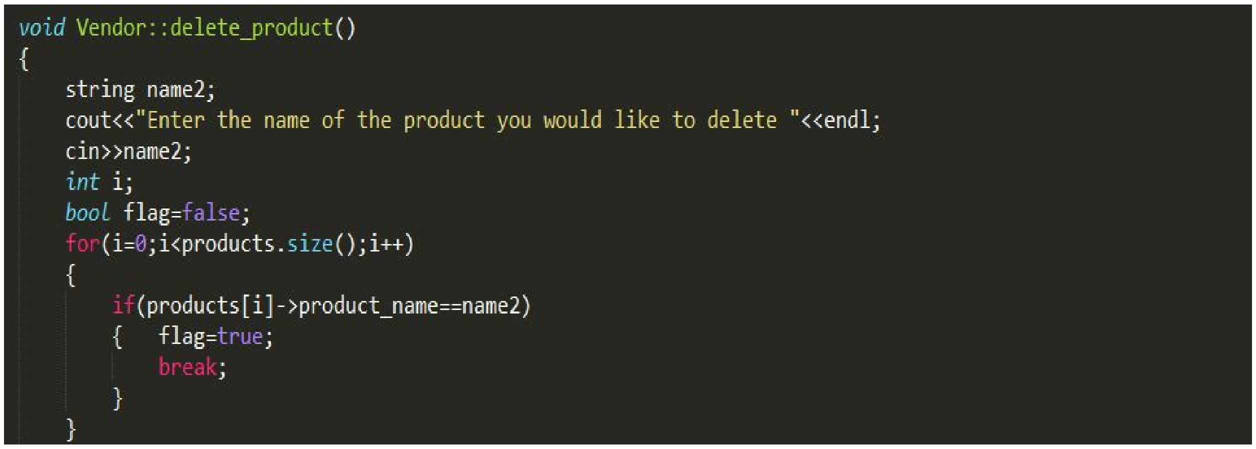
```cpp
    if(flag)
    {
        Product *obj=products[i];
        obj->category=cat;
        obj->price=pr;
        obj->tax_rate=trate;
        obj->qty_left=qty;
    }
    else
    {
        cout<<"Product name doesn't exist"<<endl;
    }
}
```

5. void view_stocklist();

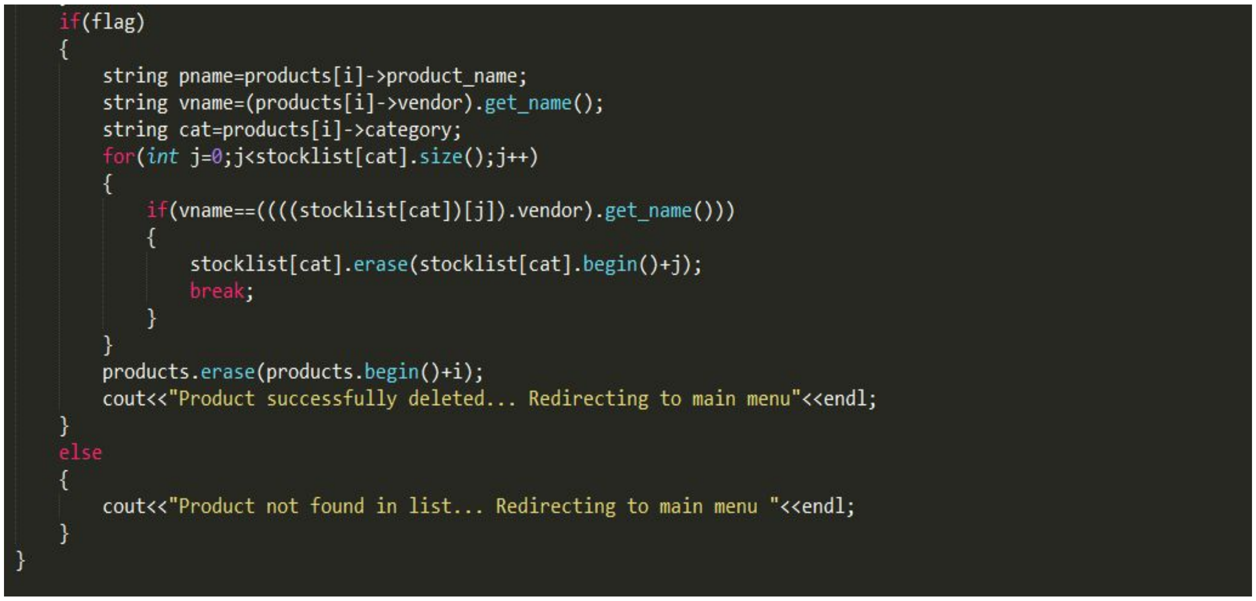


```
void Vendor::view_stocklist()
{
        for(int i =0;i<products.size();i++)
        {
                products[i]->print_product_details();
        }
}
```

6. void delete_product();



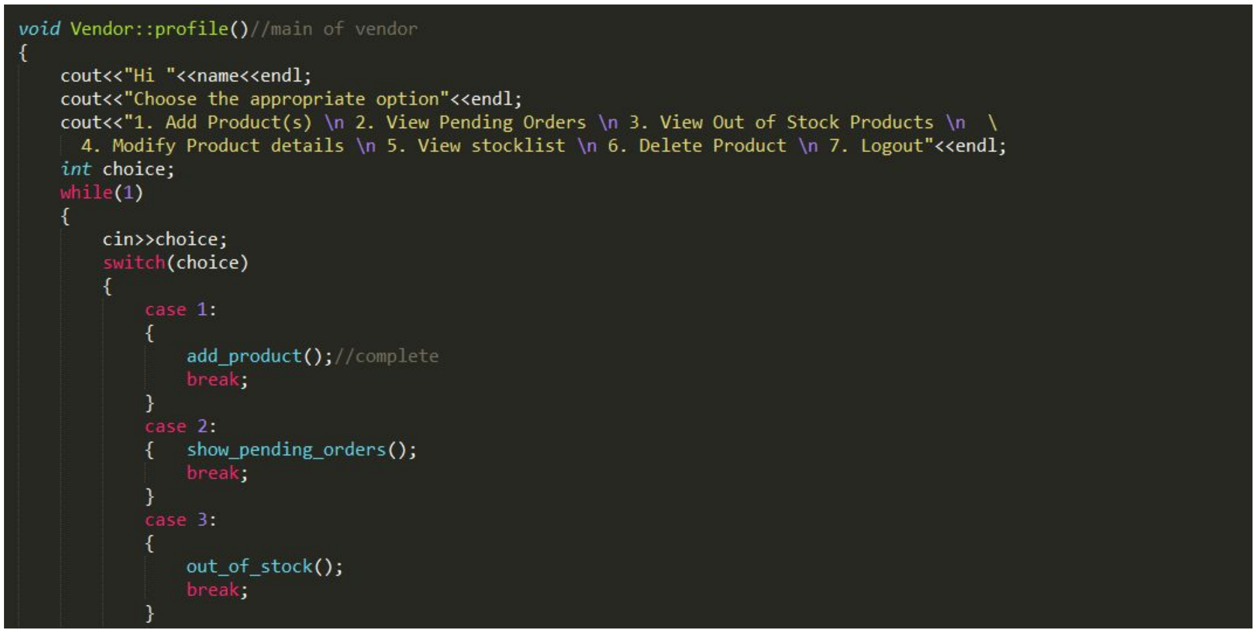
```
void Vendor::delete_product()
{
    string name2;
    cout<<"Enter the name of the product you would like to delete "<<endl;
    cin>>name2;
    int i;
    bool flag=false;
    for(i=0;i<products.size();i++)
    {
        if(products[i]->product_name==name2)
        {   flag=true;
            break;
        }
    }
```



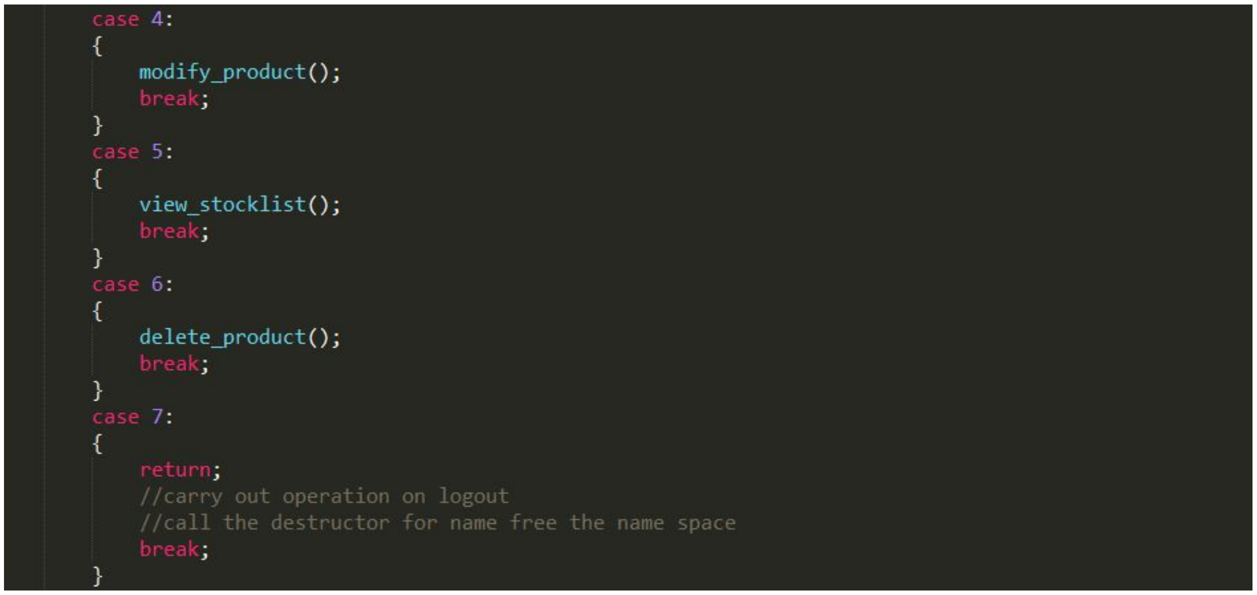
```
    if(flag)
    {
        string pname=products[i]->product_name;
        string vname=(products[i]->vendor).get_name();
        string cat=products[i]->category;
        for(int j=0;j<stocklist[cat].size();j++)
        {
            if(vname==((((stocklist[cat])[j]).vendor).get_name()))
            {
                stocklist[cat].erase(stocklist[cat].begin()+j);
                break;
            }
        }
        products.erase(products.begin()+i);
        cout<<"Product successfully deleted... Redirecting to main menu"<<endl;
    }
    else
    {
        cout<<"Product not found in list... Redirecting to main menu "<<endl;
    }
}
```

7. void profile(string name);

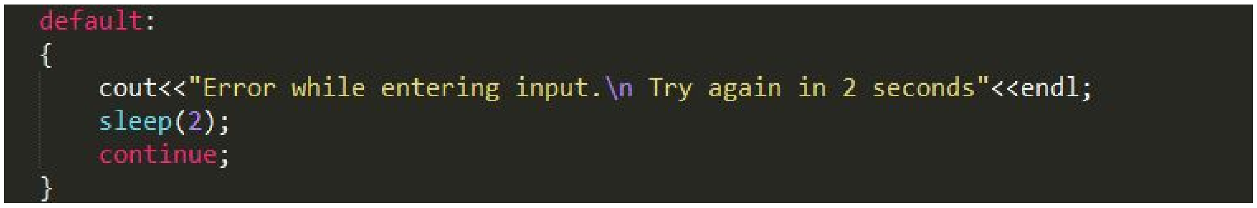
This is the driver function (declared virtual) which calls all other functions



```
void Vendor::profile()//main of vendor
{
    cout<<"Hi "<<name<<endl;
    cout<<"Choose the appropriate option"<<endl;
    cout<<"1. Add Product(s) \n 2. View Pending Orders \n 3. View Out of Stock Products \n  \
      4. Modify Product details \n 5. View stocklist \n 6. Delete Product \n 7. Logout"<<endl;
    int choice;
    while(1)
    {
        cin>>choice;
        switch(choice)
        {
            case 1:
            {
                add_product();//complete
                break;
            }
            case 2:
            {   show_pending_orders();
                break;
            }
            case 3:
            {
                out_of_stock();
                break;
            }
```



```
            case 4:
            {
                modify_product();
                break;
            }
            case 5:
            {
                view_stocklist();
                break;
            }
            case 6:
            {
                delete_product();
                break;
            }
            case 7:
            {
                return;
                //carry out operation on logout
                //call the destructor for name free the name space
                break;
            }
```

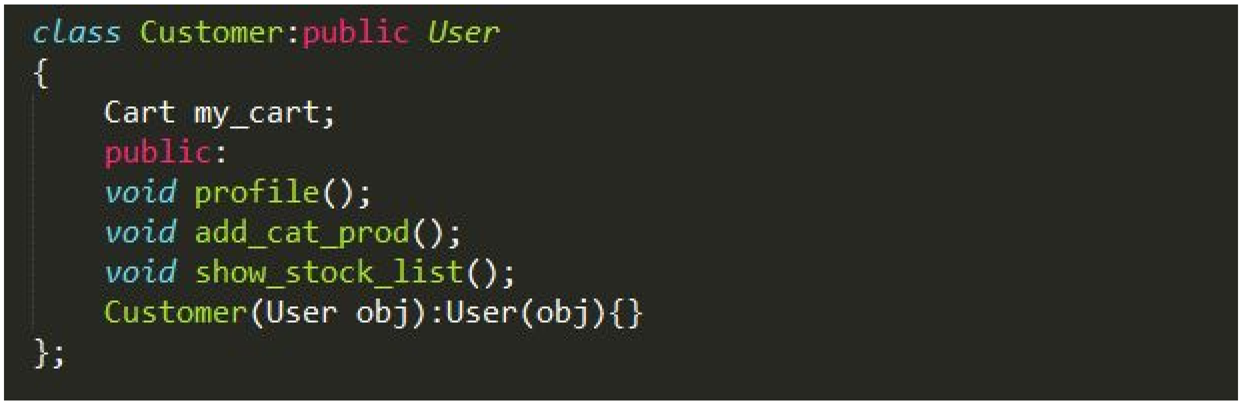


```
default:
{
    cout<<"Error while entering input.\n Try again in 2 seconds"<<endl;
    sleep(2);
    continue;
}
```

3. CUSTOMER

Overview

This part of the code is woven around the Customer class which inherits the class User. Each object of this class an object of class Cart: *Cart my_cart*.

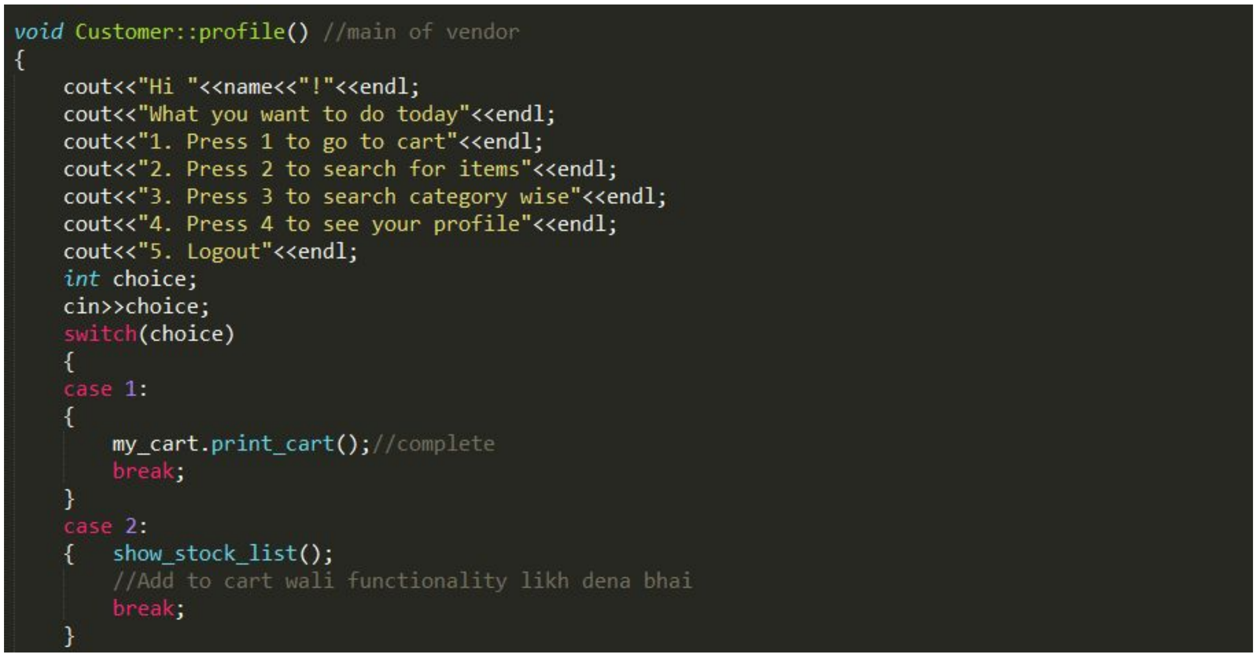


```
class Customer:public User
{
    Cart my_cart;
    public:
    void profile();
    void add_cat_prod();
    void show_stock_list();
    Customer(User obj):User(obj){}
};
```

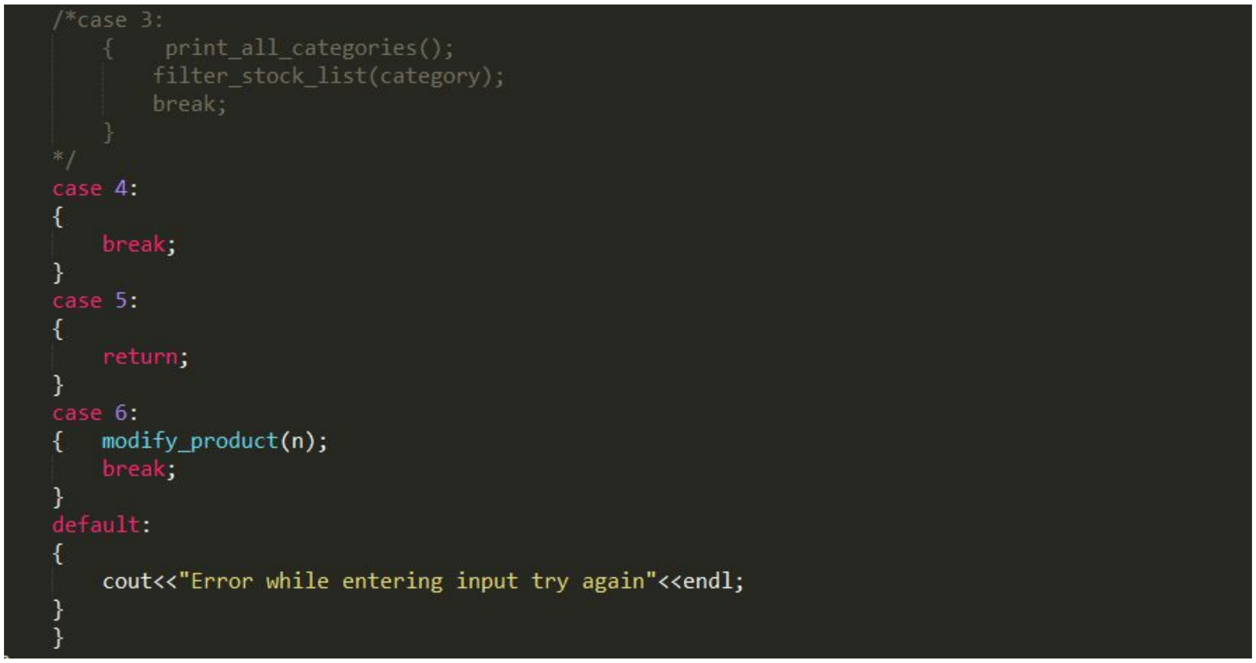
Functions for Customer class

1. **void profile();**

   Basically it's the driver function which drive all the functions means to say that it just calls others function in the class. profile () is a virtual function so that it can be used easily.



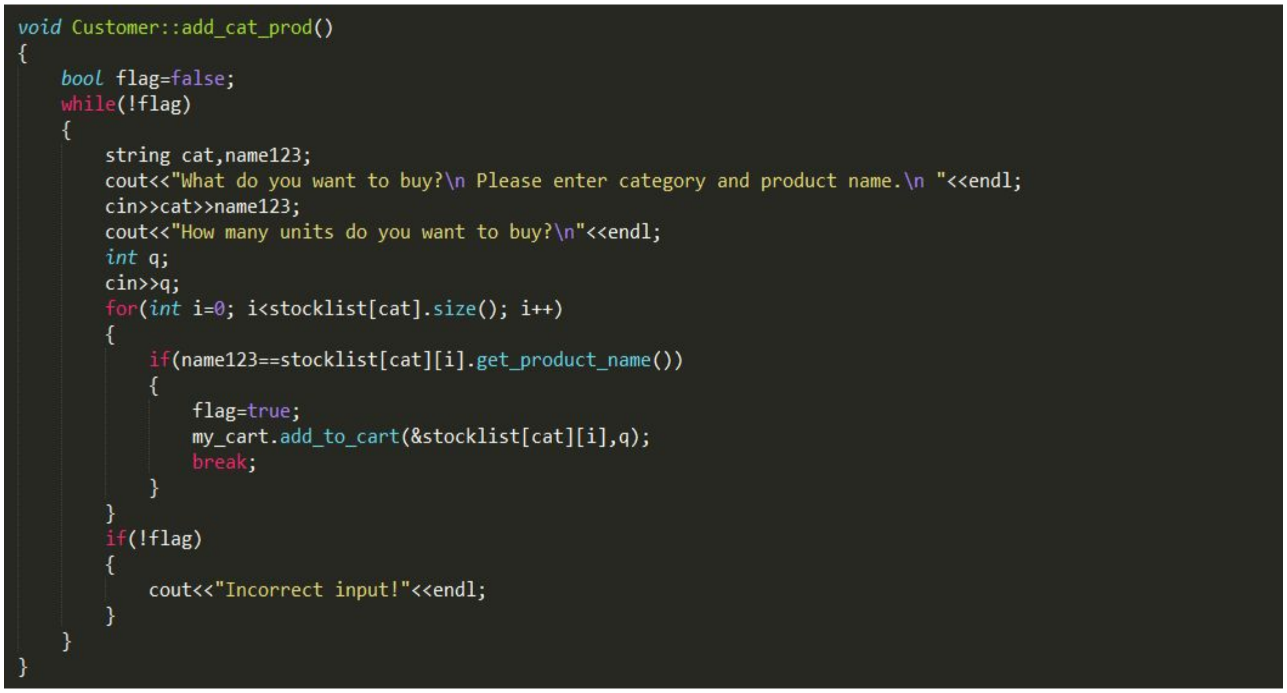
```
void Customer::profile() //main of vendor
{
    cout<<"Hi "<<name<<"!"<<endl;
    cout<<"What you want to do today"<<endl;
    cout<<"1. Press 1 to go to cart"<<endl;
    cout<<"2. Press 2 to search for items"<<endl;
    cout<<"3. Press 3 to search category wise"<<endl;
    cout<<"4. Press 4 to see your profile"<<endl;
    cout<<"5. Logout"<<endl;
    int choice;
    cin>>choice;
    switch(choice)
    {
    case 1:
    {
        my_cart.print_cart();//complete
        break;
    }
    case 2:
    {   show_stock_list();
        //Add to cart wali functionality likh dena bhai
        break;
    }
```



```
/*case 3:
    {    print_all_categories();
        filter_stock_list(category);
        break;
    }
*/
case 4:
{
    break;
}
case 5:
{
    return;
}
case 6:
{   modify_product(n);
    break;
}
default:
{
    cout<<"Error while entering input try again"<<endl;
}
}
```

2. void add_cat_prod();

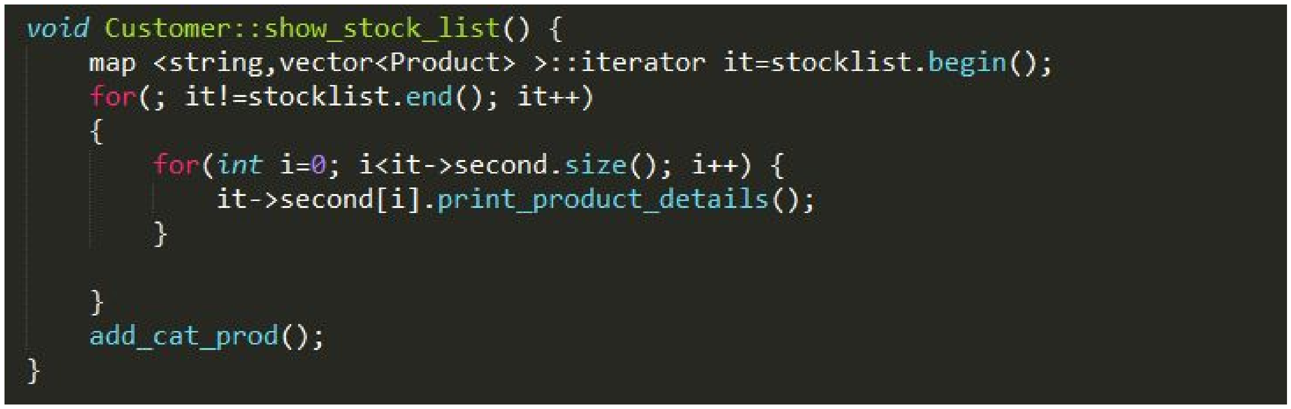
This function asks for the category and name of the product you wish to add to your cart and also its quantity. And add those items into the cart.



```
void Customer::add_cat_prod()
{
    bool flag=false;
    while(!flag)
    {
        string cat,name123;
        cout<<"What do you want to buy?\n Please enter category and product name.\n "<<endl;
        cin>>cat>>name123;
        cout<<"How many units do you want to buy?\n"<<endl;
        int q;
        cin>>q;
        for(int i=0; i<stocklist[cat].size(); i++)
        {
            if(name123==stocklist[cat][i].get_product_name())
            {
                flag=true;
                my_cart.add_to_cart(&stocklist[cat][i],q);
                break;
            }
        }
        if(!flag)
        {
            cout<<"Incorrect input!"<<endl;
        }
    }
}
```

3. void show_stock_list();

This function prints the complete stock list. And also it ask customer to add items into the cart. As add_cat_prod() is called inside it.

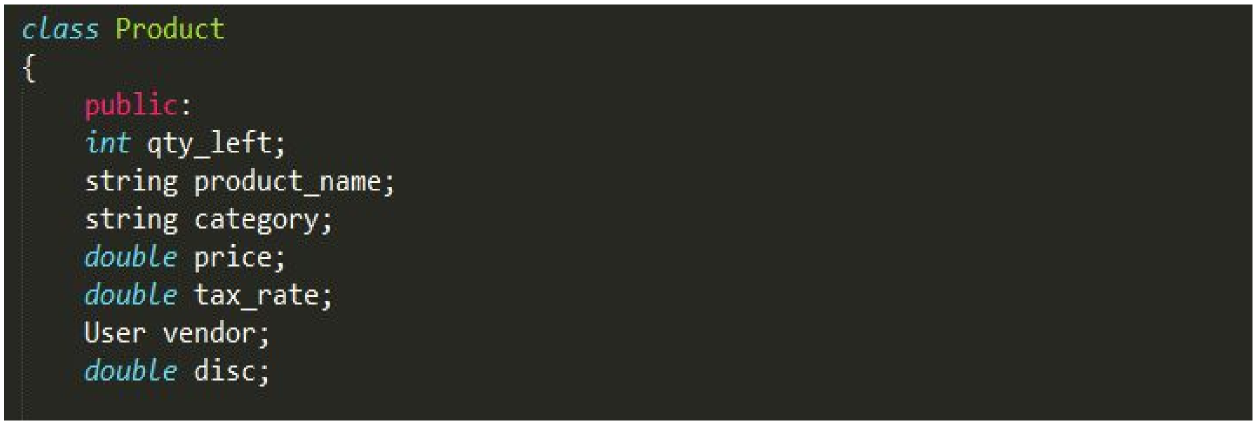


```
void Customer::show_stock_list() {
    map <string,vector<Product> >::iterator it=stocklist.begin();
    for(; it!=stocklist.end(); it++)
    {
        for(int i=0; i<it->second.size(); i++) {
            it->second[i].print_product_details();
        }

    }
    add_cat_prod();
}
```

4. STOCKLIST

Overview

This part contains the *class Product* and *map<string,vector<Product> > stocklist;*.



```
class Product
{
    public:
    int qty_left;
    string product_name;
    string category;
    double price;
    double tax_rate;
    User vendor;
    double disc;
```

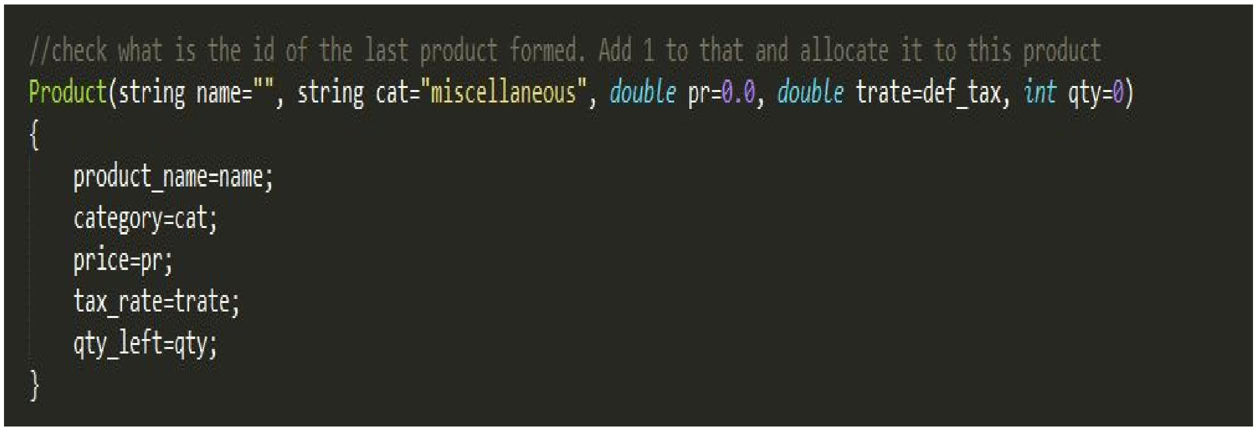
Information stored in Product class

1. Quantity left with the vendor of that product.
2. Name of the product.
3. Name of the category.
4. Price of the product.
5. Current tax rate on that product.
6. Details of vendor selling that product.
7. Discount on that product.

Functions for Product class

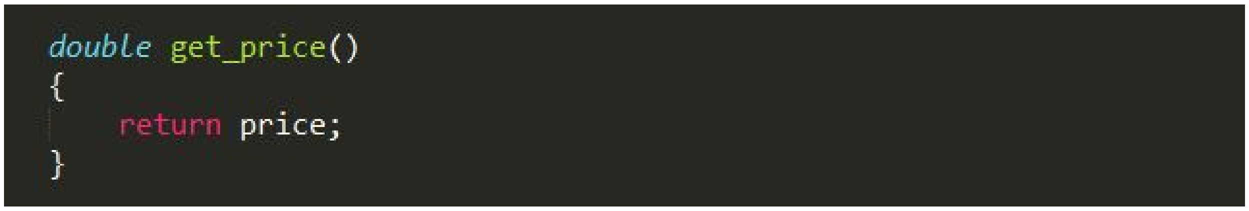
1. **Product(string, string, double, double, int);**

It is the constructor to the class Product.



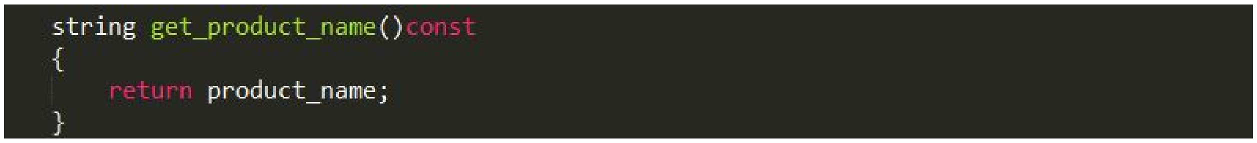
```
//check what is the id of the last product formed. Add 1 to that and allocate it to this product
Product(string name="", string cat="miscellaneous", double pr=0.0, double trate=def_tax, int qty=0)
{
    product_name=name;
    category=cat;
    price=pr;
    tax_rate=trate;
    qty_left=qty;
}
```

2. **double get_price();**



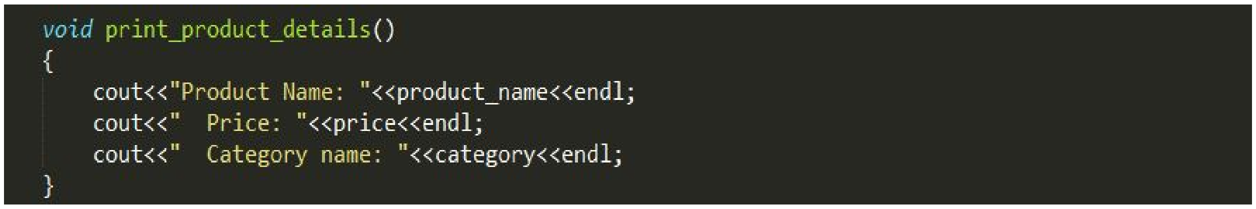
```
double get_price()
{
    return price;
}
```

3. **string get_product_name() const;**



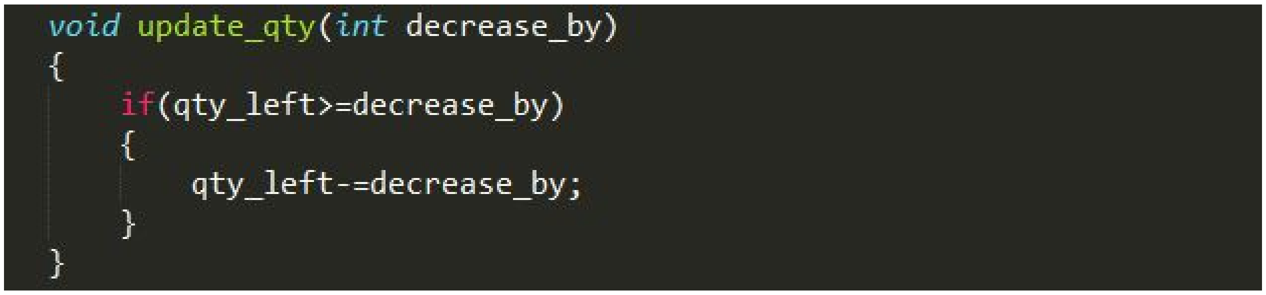
```
string get_product_name()const
{
    return product_name;
}
```

4. **void print_product_details();**



```
void print_product_details()
{
    cout<<"Product Name: "<<product_name<<endl;
    cout<<"  Price: "<<price<<endl;
    cout<<"  Category name: "<<category<<endl;
}
```

5. **void update_qty(int);**

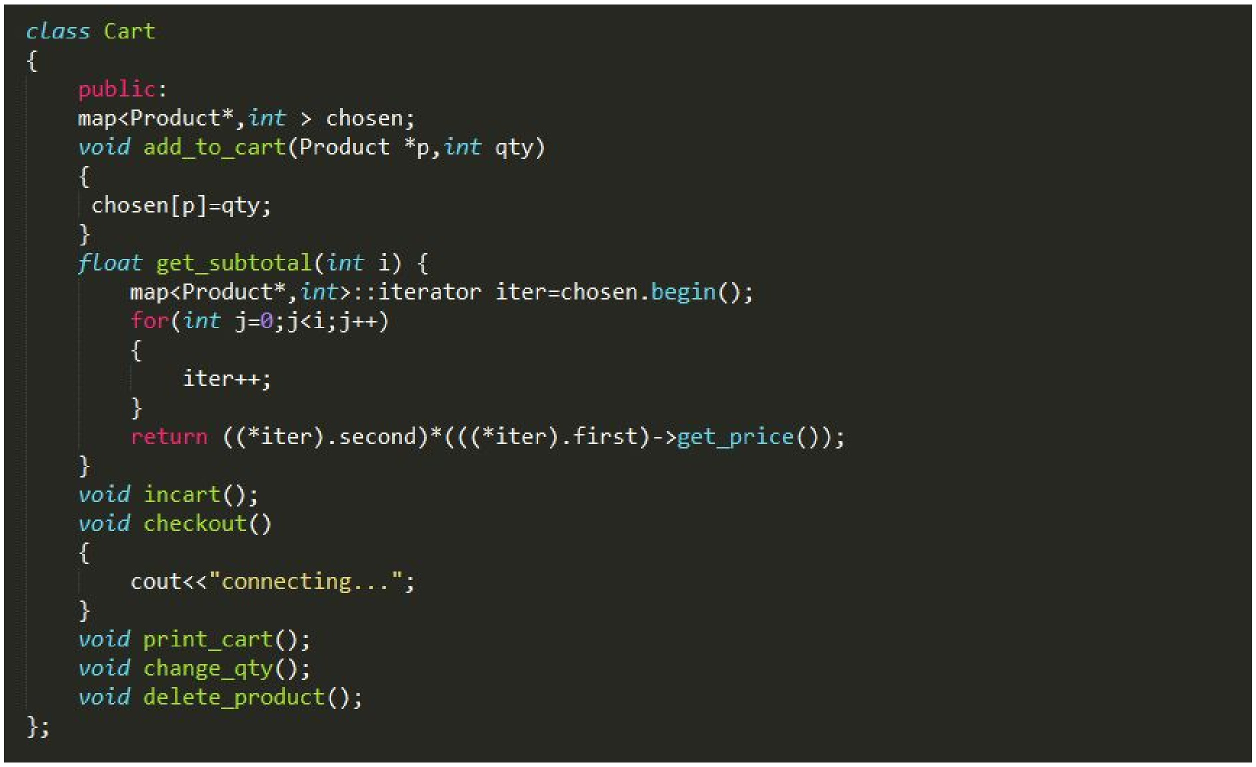


```
void update_qty(int decrease_by)
{
    if(qty_left>=decrease_by)
    {
        qty_left-=decrease_by;
    }
}
```

5. CART

Overview

The cart allows the customers to select items for eventual purchase. This part of our code is built around the Cart class. The selected items are stored in a map *map<Product*,int> chosen;*.



```
class Cart
{
    public:
    map<Product*,int > chosen;
    void add_to_cart(Product *p,int qty)
    {
     chosen[p]=qty;
    }
    float get_subtotal(int i) {
        map<Product*,int>::iterator iter=chosen.begin();
        for(int j=0;j<i;j++)
        {
            iter++;
        }
        return ((*iter).second)*(((*iter).first)->get_price());
    }
    void incart();
    void checkout()
    {
        cout<<"connecting...";
    }
    void print_cart();
    void change_qty();
    void delete_product();
};
```

Functions for Cart class

1. **void add_to_cart(Product*, int);**

It takes a pointer to an object of class Product and an integer as input. To insert them into the map it uses operator[] in order to overwrite any possibly existing element, this ensures that we have only one element in the map for same product even if we add the product to the cart multiple times. The required in the input the quantity of the product the customer wishes to buy.

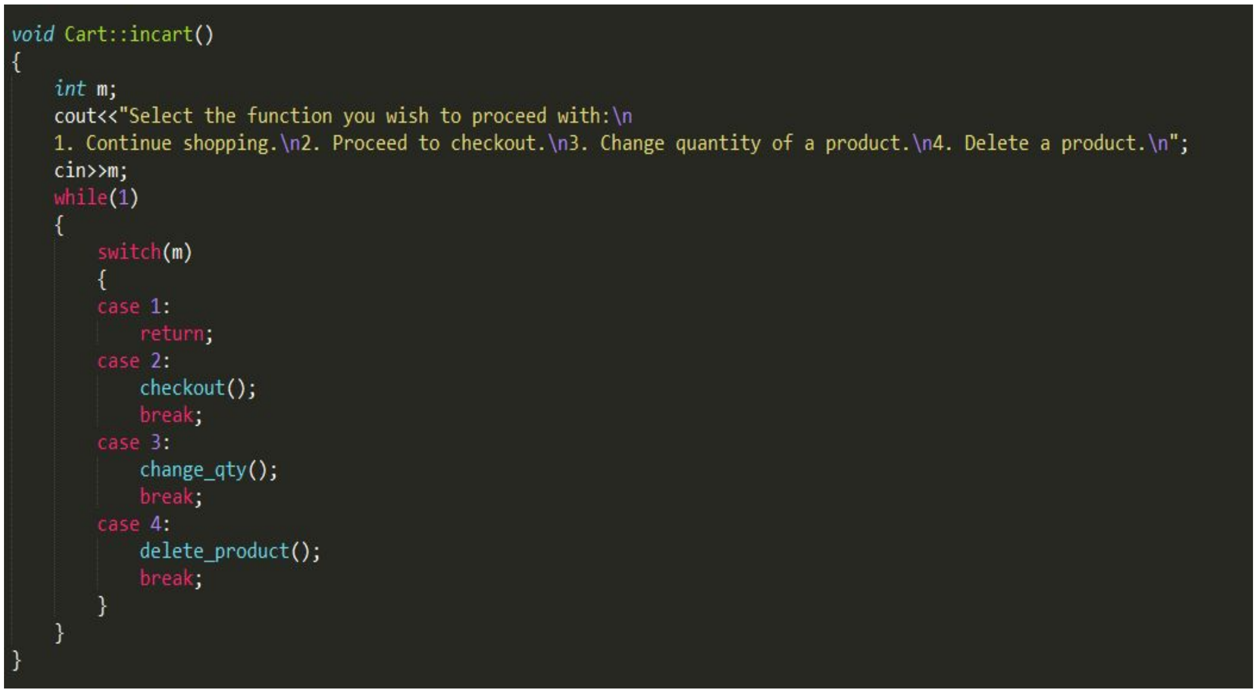
2. **float get_subtotal(int);**

This function uses the iterator to traverse through the elements of the map. It returns the subtotal for the product stored in the i'th element (i is the input) of the map by multiplying the quantity of that product with its price.

3. void incart();

This function simply contains a switch case and asks the customer to choose between four different actions they want to proceed with, namely,

a. Continue shopping
b. Proceed to checkout
c. Change quantity of a product
d. Delete a product



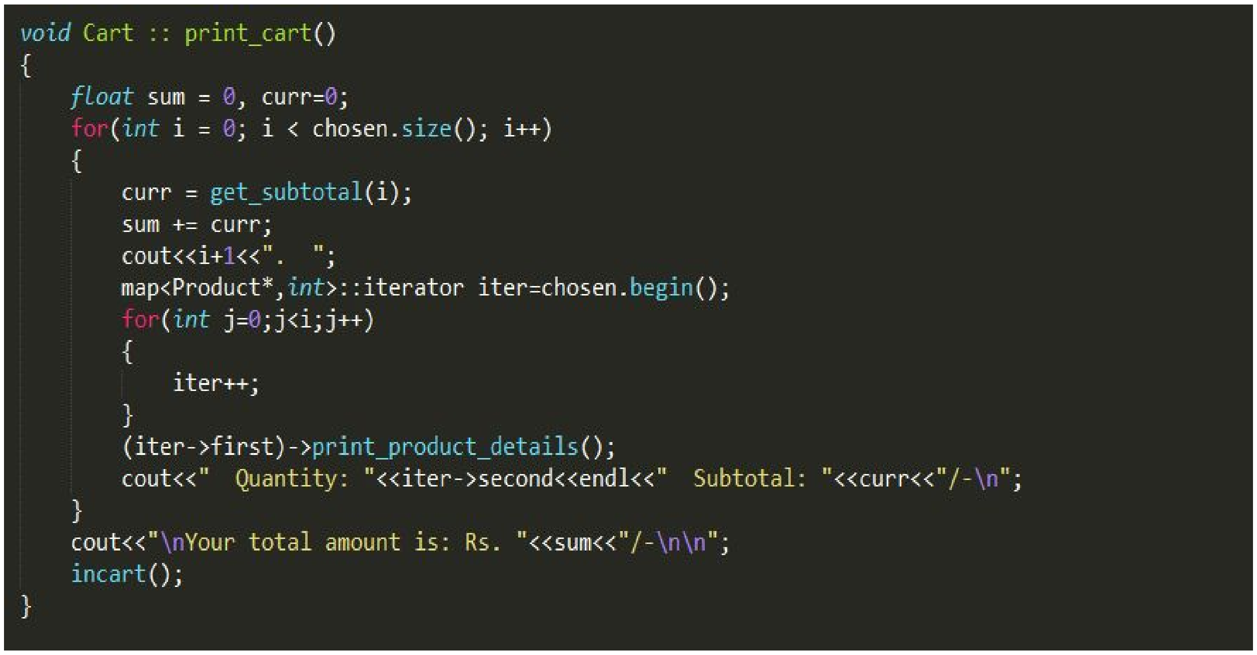
```
void Cart::incart()
{
    int m;
    cout<<"Select the function you wish to proceed with:\n
    1. Continue shopping.\n2. Proceed to checkout.\n3. Change quantity of a product.\n4. Delete a product.\n";
    cin>>m;
    while(1)
    {
        switch(m)
        {
        case 1:
            return;
        case 2:
            checkout();
            break;
        case 3:
            change_qty();
            break;
        case 4:
            delete_product();
            break;
        }
    }
}
```

4. void checkout();

This function takes the customer out of the program and shows a connecting message.

5. void print_cart();

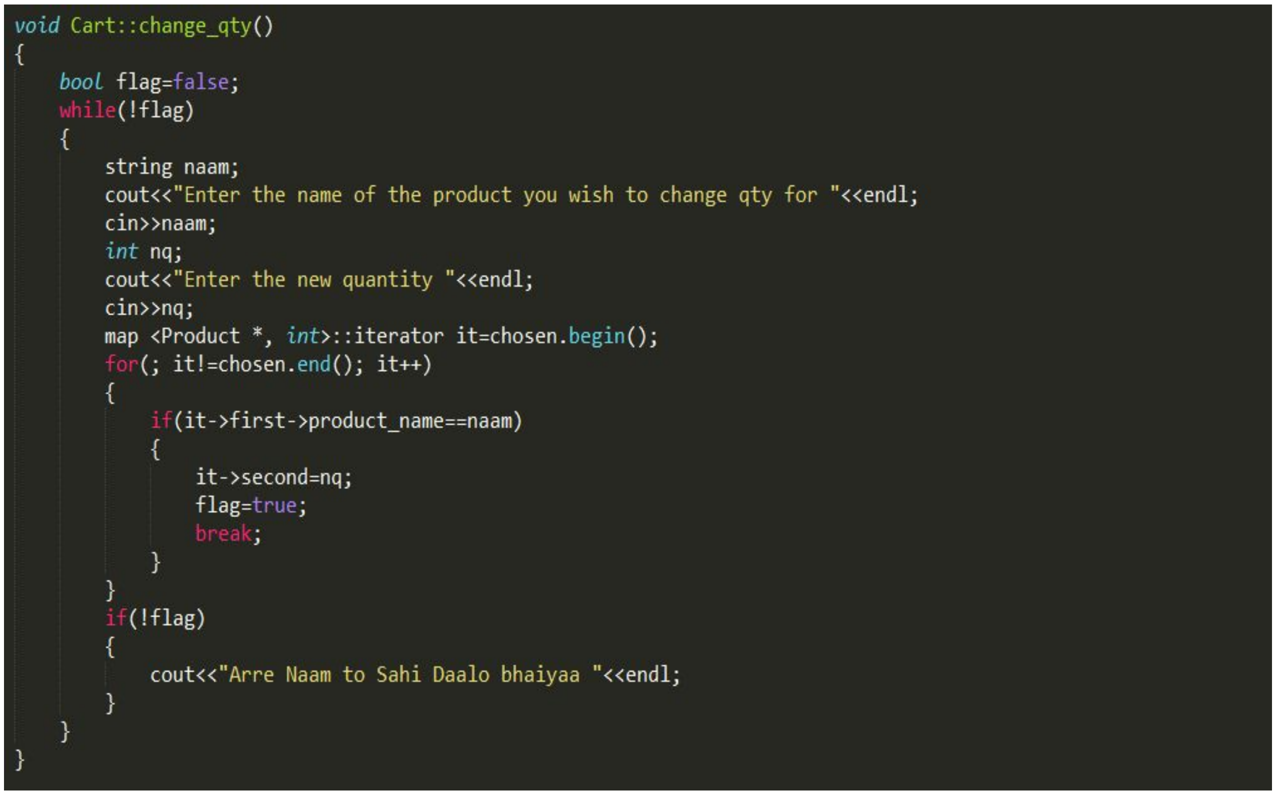
This function prints the details of all the products along with their quantity selected and the subtotal. It also prints the total amount the customer needs to pay if he buys the whole cart.



```
void Cart :: print_cart()
{
    float sum = 0, curr=0;
    for(int i = 0; i < chosen.size(); i++)
    {
        curr = get_subtotal(i);
        sum += curr;
        cout<<i+1<<".  ";
        map<Product*,int>::iterator iter=chosen.begin();
        for(int j=0;j<i;j++)
        {
            iter++;
        }
        (iter->first)->print_product_details();
        cout<<"  Quantity: "<<iter->second<<endl<<"  Subtotal: "<<curr<<"/-\n";
    }
    cout<<"\nYour total amount is: Rs. "<<sum<<"/-\n\n";
    incart();
}
```

6. void change_qty();

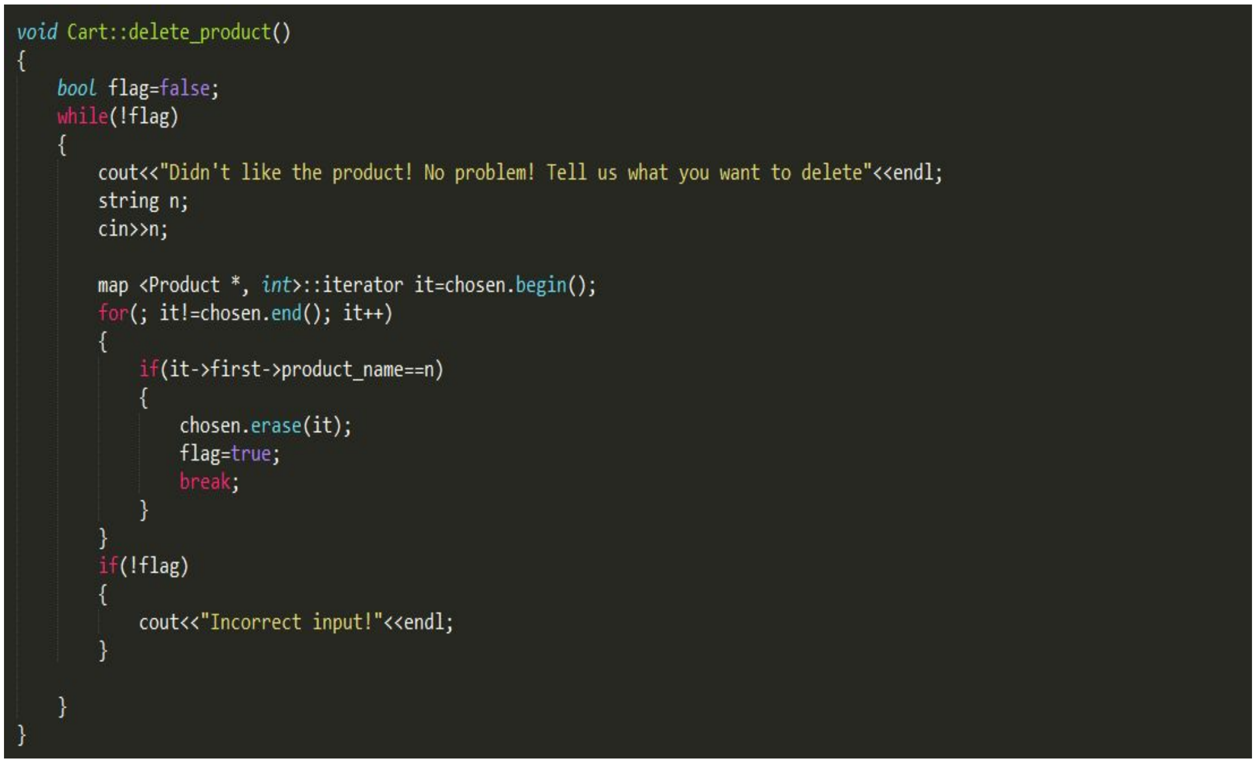
It first asks the name of the product whose quantity the customer wishes to change. Then if the product exists in the cart, it updates its quantity by taking input from the customer, else it asks to enter the name of product again.



```
void Cart::change_qty()
{
    bool flag=false;
    while(!flag)
    {
        string naam;
        cout<<"Enter the name of the product you wish to change qty for "<<endl;
        cin>>naam;
        int nq;
        cout<<"Enter the new quantity "<<endl;
        cin>>nq;
        map <Product *, int>::iterator it=chosen.begin();
        for(; it!=chosen.end(); it++)
        {
            if(it->first->product_name==naam)
            {
                it->second=nq;
                flag=true;
                break;
            }
        }
        if(!flag)
        {
            cout<<"Arre Naam to Sahi Daalo bhaiyaa "<<endl;
        }
    }
}
```

7. void delete_product();

If the customer confirms that he wishes to delete a product from his cart, the function asks the name of the product and then if the product exists, it is erased from his cart.



```cpp
void Cart::delete_product()
{
    bool flag=false;
    while(!flag)
    {
        cout<<"Didn't like the product! No problem! Tell us what you want to delete"<<endl;
        string n;
        cin>>n;

        map <Product *, int>::iterator it=chosen.begin();
        for(; it!=chosen.end(); it++)
        {
            if(it->first->product_name==n)
            {
                chosen.erase(it);
                flag=true;
                break;
            }
        }
        if(!flag)
        {
            cout<<"Incorrect input!"<<endl;
        }

    }
}
```

CAPTCHA: the bonus

A Brief Introduction

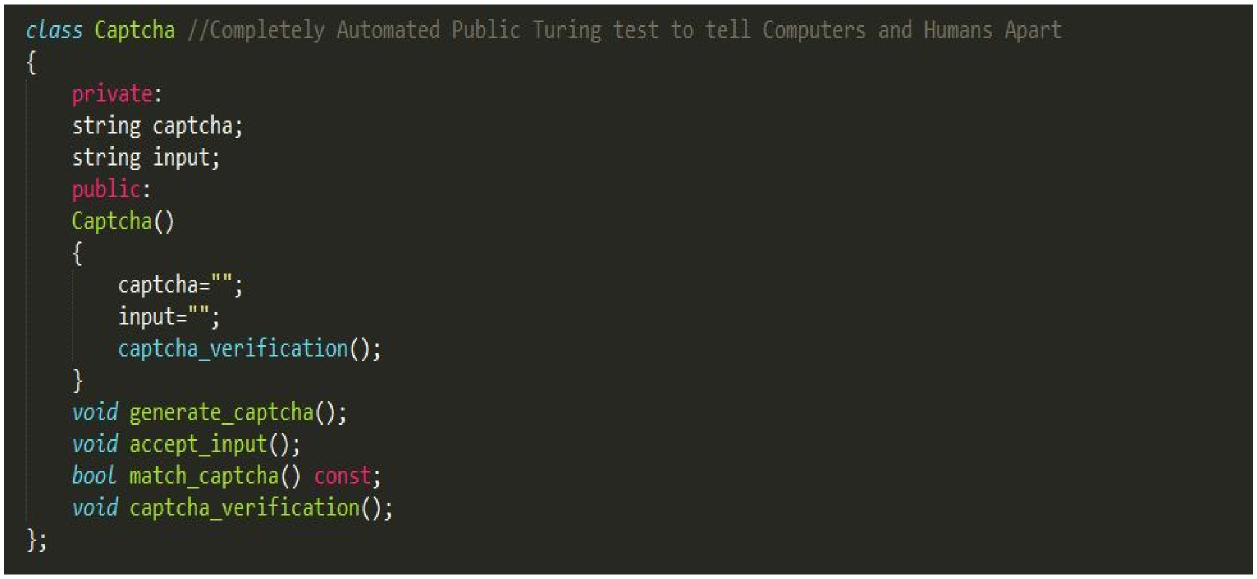
A CAPTCHA (an acronym for "Completely Automated Public Turing test to tell Computers and Humans Apart") is a type of challenge-response test used in computing to determine whether or not the user is human.

The term was coined in 2003 by Luis von Ahn, Manuel Blum, Nicholas J. Hopper, and John Langford. The most common type of CAPTCHA was first invented in 1997 by two groups working in parallel: (1) Mark D. Lillibridge, Martin Abadi, Krishna Bharat, and Andrei Z. Broder; and (2) Eran Reshef, Gili Raanan and Eilon Solan.

CAPTCHA requires that the user type the letters of a distorted image or the characters in a string sequence (with alphanumeric characters), sometimes with the addition of an obscured sequence of letters or digits that appears on the screen. Because the test is administered by a computer, in contrast to the standard Turing test that is administered by a human, a CAPTCHA is sometimes described as a reverse Turing test.

Captcha Class

Class Captcha is defined to implement the functionality of displaying a random captcha to the screen, accepting a captcha from the user, comparing the user input with the captcha actually displayed and directing him accordingly.

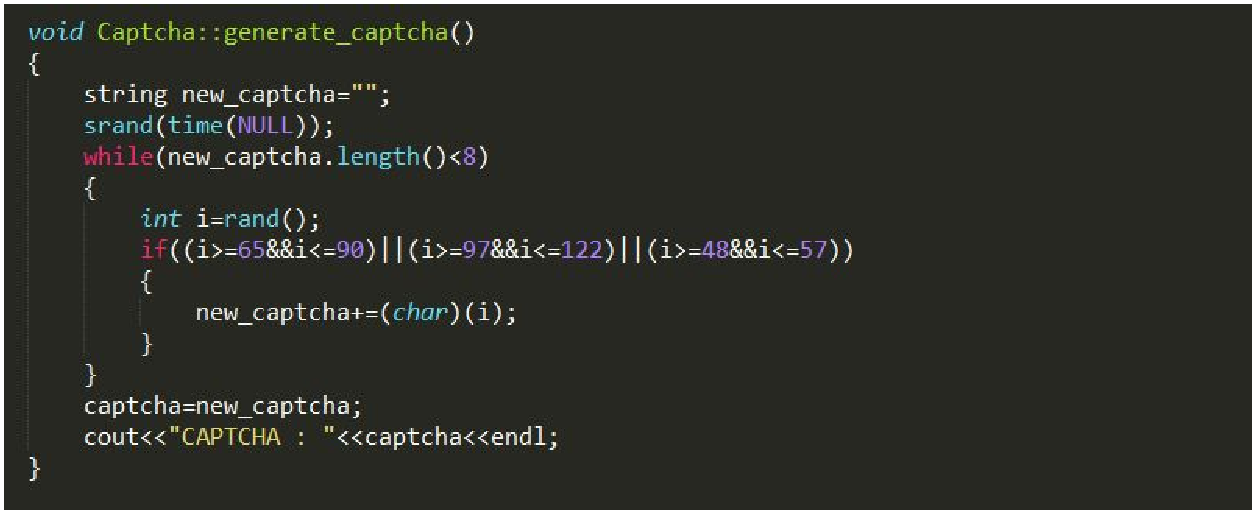


```
class Captcha //Completely Automated Public Turing test to tell Computers and Humans Apart
{
    private:
    string captcha;
    string input;
    public:
    Captcha()
    {
        captcha="";
        input="";
        captcha_verification();
    }
    void generate_captcha();
    void accept_input();
    bool match_captcha() const;
    void captcha_verification();
};
```

The captcha class definition

1. **void generate_captcha ()**

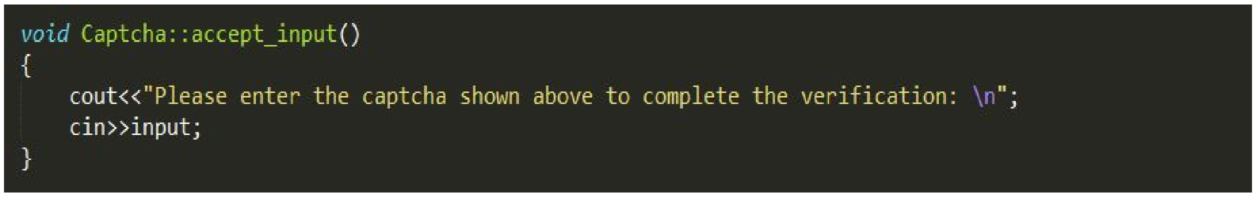
The code snippet shown below generates and displays a random captcha. Using rand() function of the stdlib.h header file, this snippet generates a random value i and adds it to the captcha string if its ASCII value lies in the alphanumeric range.



```
void Captcha::generate_captcha()
{
    string new_captcha="";
    srand(time(NULL));
    while(new_captcha.length()<8)
    {
        int i=rand();
        if((i>=65&&i<=90)||(i>=97&&i<=122)||(i>=48&&i<=57))
        {
            new_captcha+=(char)(i);
        }
    }
    captcha=new_captcha;
    cout<<"CAPTCHA : "<<captcha<<endl;
}
```

2. void accept_input()

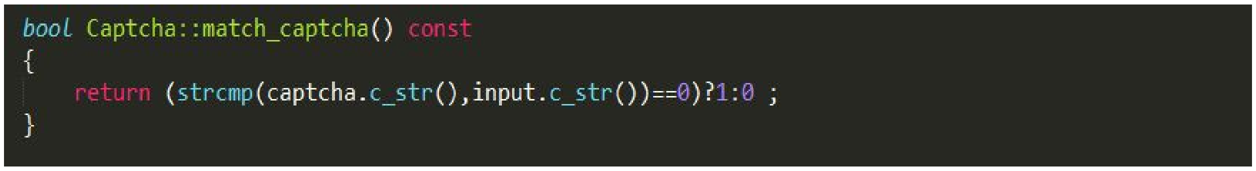
This snippet accepts an input captcha from the user.



```
void Captcha::accept_input()
{
    cout<<"Please enter the captcha shown above to complete the verification: \n";
    cin>>input;
}
```

3. bool match_captcha() const

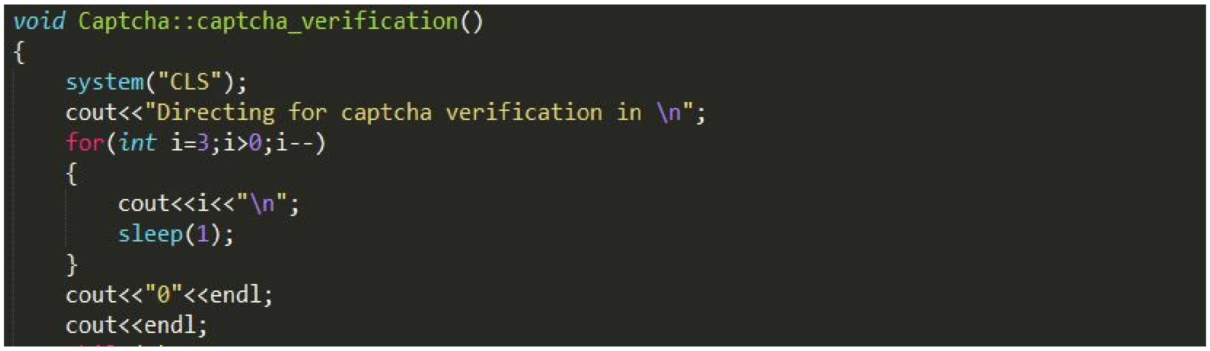
This part of the code compares the actual captcha with the inputted one.



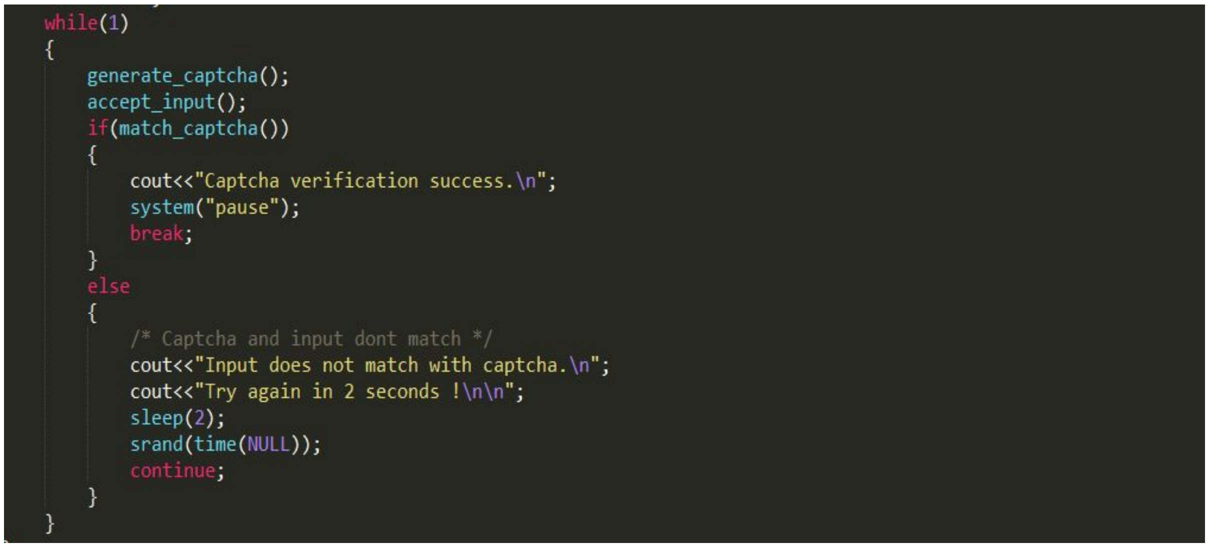
```
bool Captcha::match_captcha() const
{
    return (strcmp(captcha.c_str(),input.c_str())==0)?1:0 ;
}
```

4. void captcha_verification()

The initial part of the snippet contains a timer of 3 seconds. In the latter half, it call the other functions accordingly to generate a random captcha, accept the input and perform the comparison. If the input and true captcha don't match, the user is asked to try again with a different generated captcha.



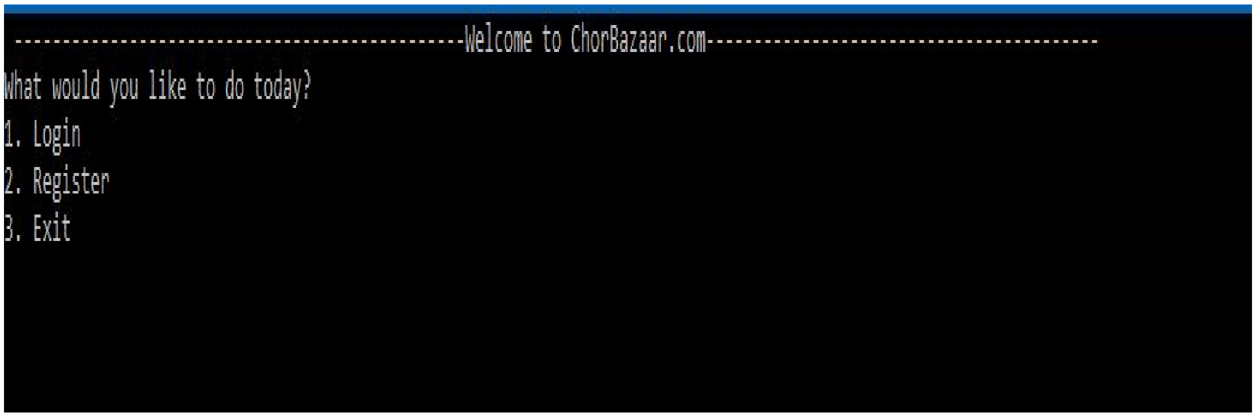
```
void Captcha::captcha_verification()
{
    system("CLS");
    cout<<"Directing for captcha verification in \n";
    for(int i=3;i>0;i--)
    {
        cout<<i<<"\n";
        sleep(1);
    }
    cout<<"0"<<endl;
    cout<<endl;
```



```
    while(1)
    {
        generate_captcha();
        accept_input();
        if(match_captcha())
        {
            cout<<"Captcha verification success.\n";
            system("pause");
            break;
        }
        else
        {
            /* Captcha and input dont match */
            cout<<"Input does not match with captcha.\n";
            cout<<"Try again in 2 seconds !\n\n";
            sleep(2);
            srand(time(NULL));
            continue;
        }
    }
```

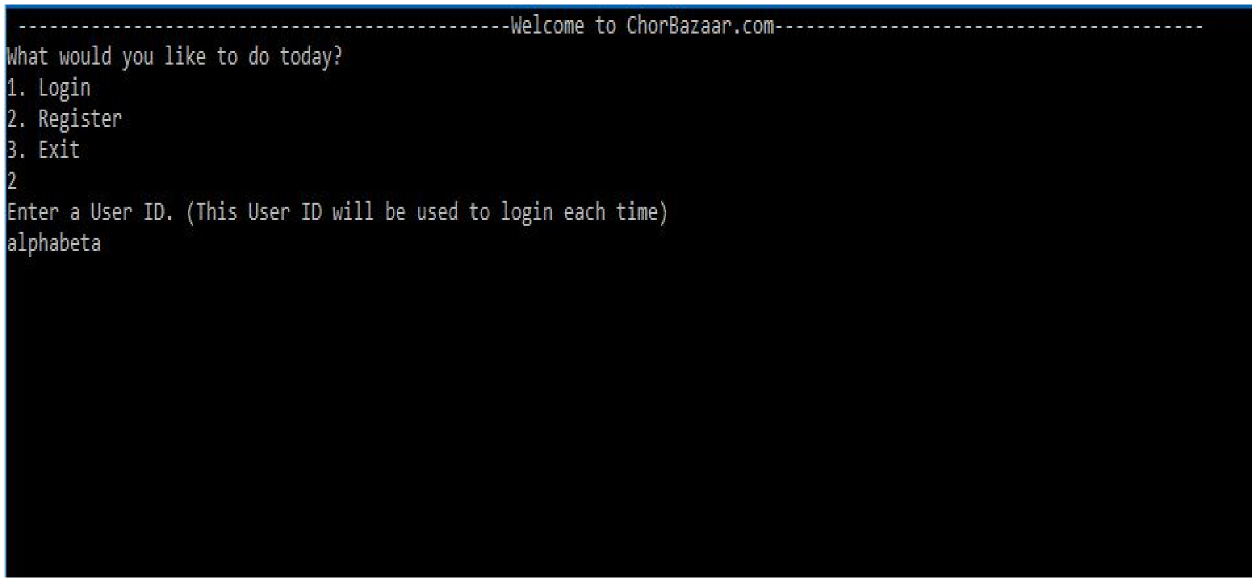
OUTPUTS

1. Initial output.



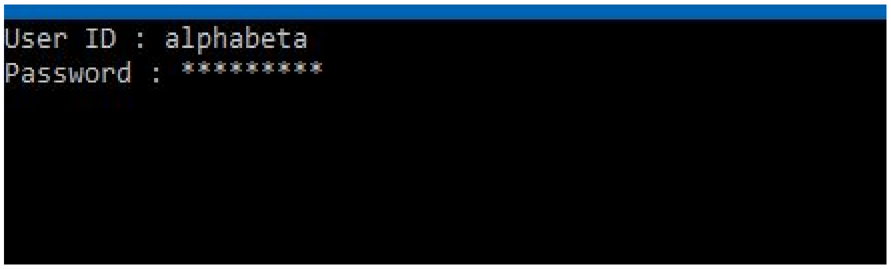
```
---------------------------------------------Welcome to ChorBazaar.com---------------------------------------
What would you like to do today?
1. Login
2. Register
3. Exit
```

2. Input: 2 (register)



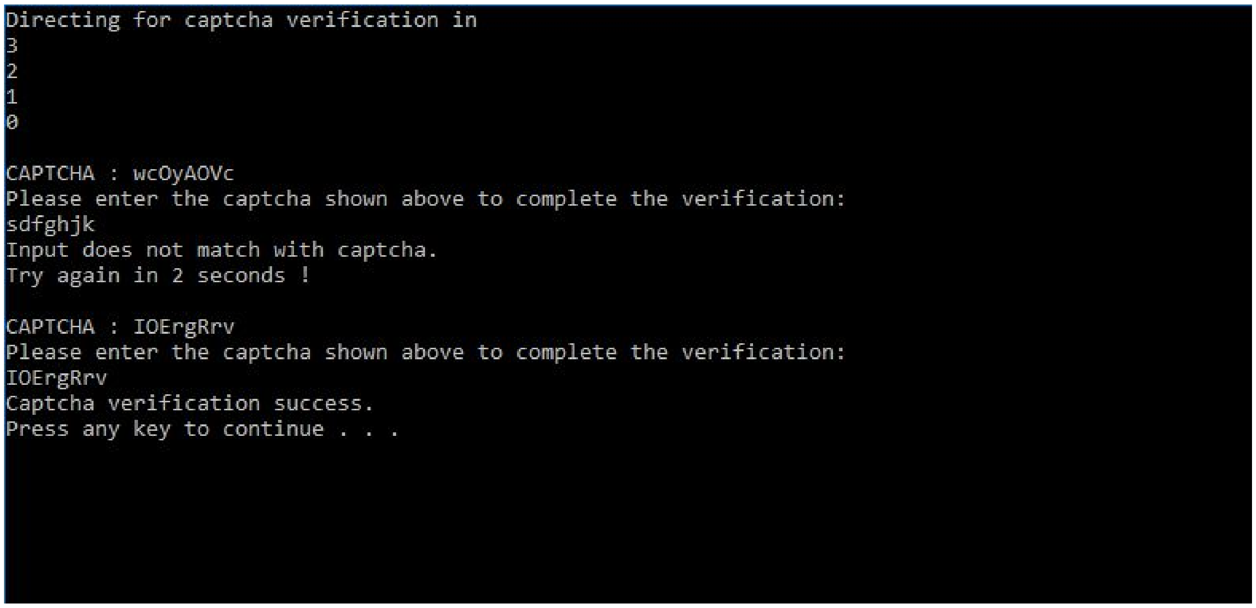
```
---------------------------------------------Welcome to ChorBazaar.com---------------------------------------
What would you like to do today?
1. Login
2. Register
3. Exit
2
Enter a User ID. (This User ID will be used to login each time)
alphabeta
```

3. User ID and password



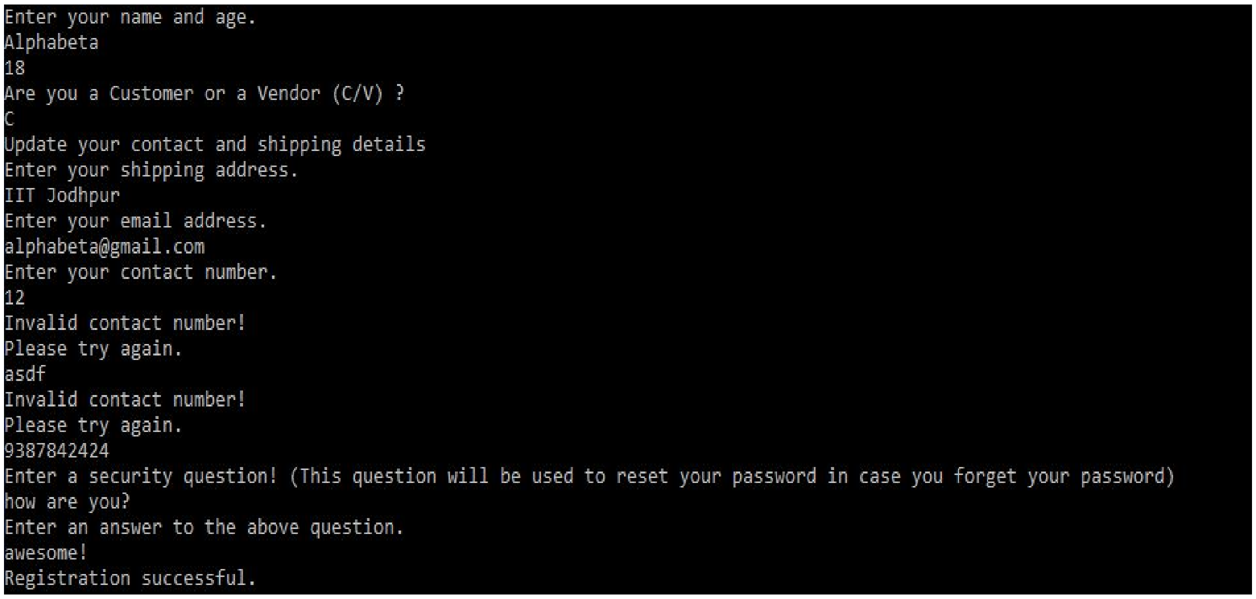
```
User ID : alphabeta
Password : *********
```

4. CAPTCHA verification



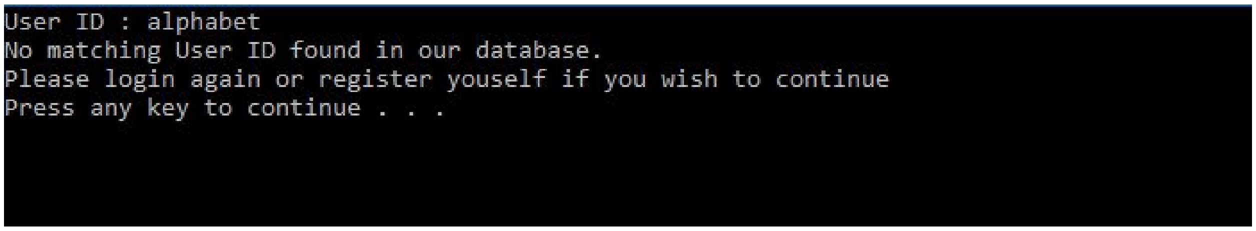
```
Directing for captcha verification in
3
2
1
0

CAPTCHA : wcOyAOVc
Please enter the captcha shown above to complete the verification:
sdfghjk
Input does not match with captcha.
Try again in 2 seconds !

CAPTCHA : IOErgRrv
Please enter the captcha shown above to complete the verification:
IOErgRrv
Captcha verification success.
Press any key to continue . . .
```

5. Updating profile details



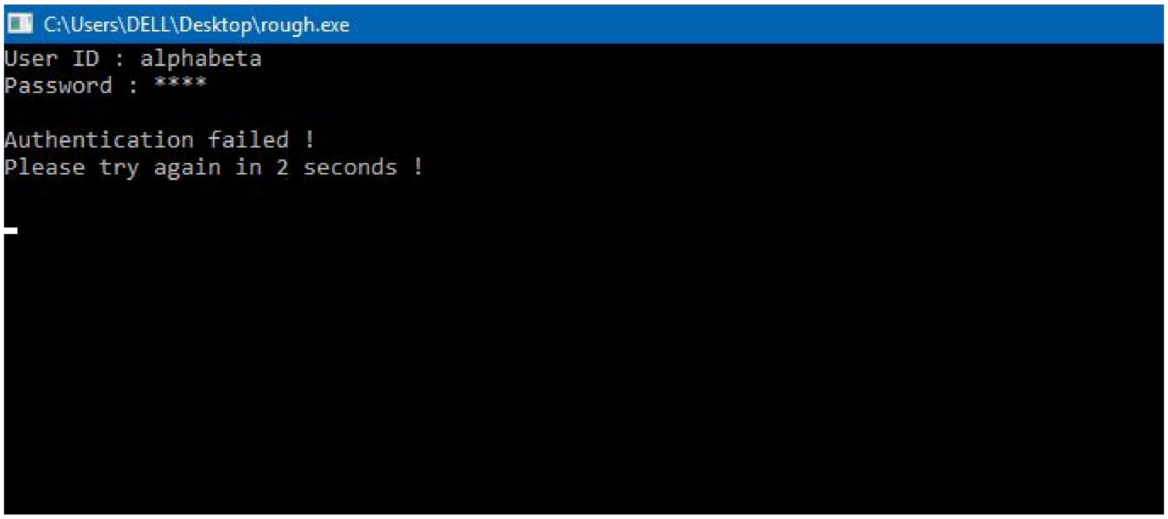
```
Enter your name and age.
Alphabeta
18
Are you a Customer or a Vendor (C/V) ?
C
Update your contact and shipping details
Enter your shipping address.
IIT Jodhpur
Enter your email address.
alphabeta@gmail.com
Enter your contact number.
12
Invalid contact number!
Please try again.
asdf
Invalid contact number!
Please try again.
9387842424
Enter a security question! (This question will be used to reset your password in case you forget your password)
how are you?
Enter an answer to the above question.
awesome!
Registration successful.
```

6. Login (with a wrong user id)



```
User ID : alphabet
No matching User ID found in our database.
Please login again or register youself if you wish to continue
Press any key to continue . . .
```

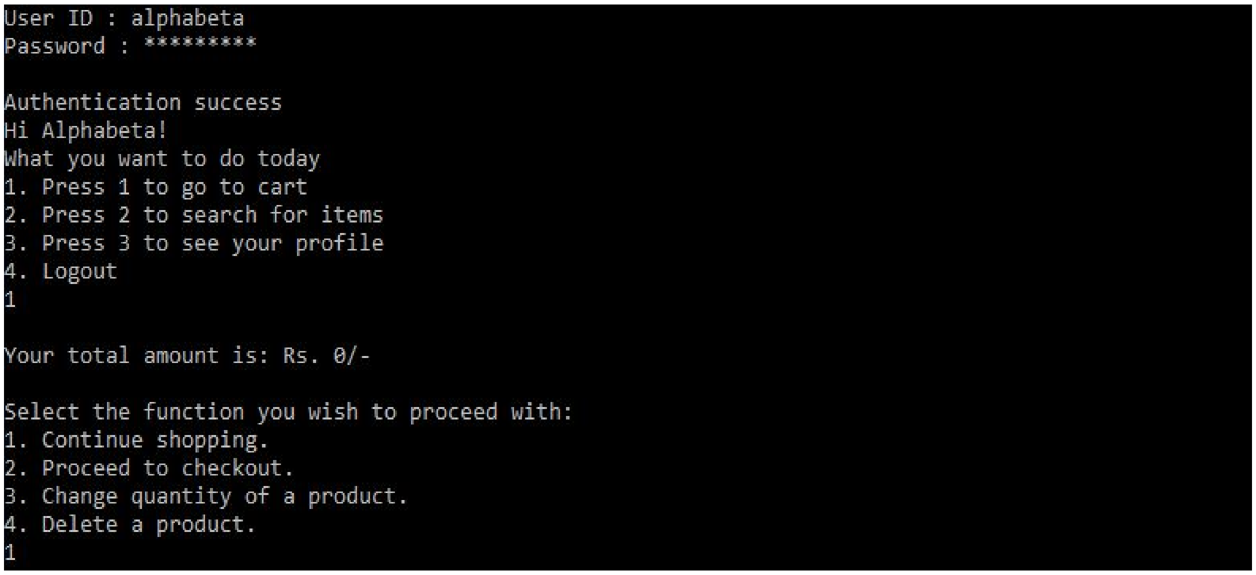
7. If password is incorrect,



C:\Users\DELL\Desktop\rough.exe

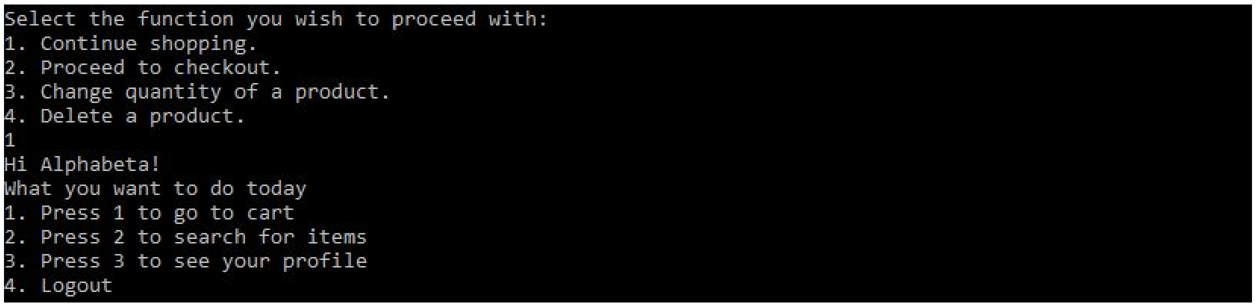
```
User ID : alphabeta
Password : ****

Authentication failed !
Please try again in 2 seconds !

_
```

8. After login,



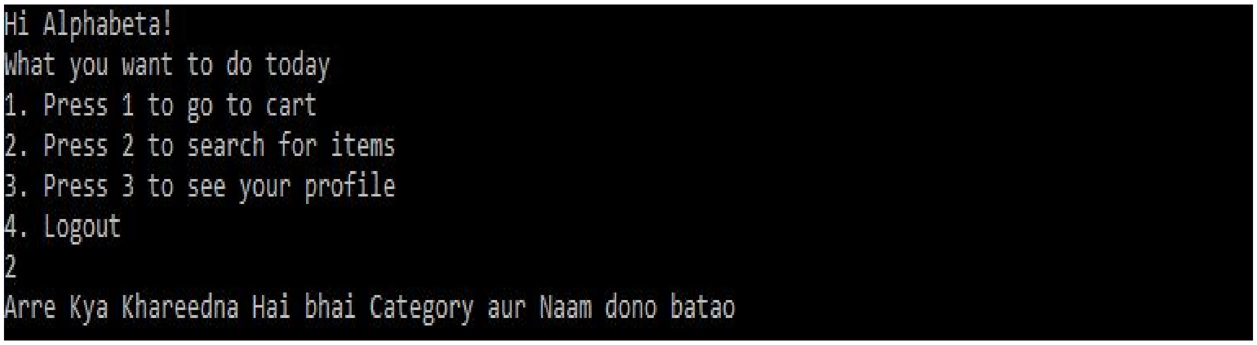
```
User ID : alphabeta
Password : *********

Authentication success
Hi Alphabeta!
What you want to do today
1. Press 1 to go to cart
2. Press 2 to search for items
3. Press 3 to see your profile
4. Logout
1

Your total amount is: Rs. 0/-

Select the function you wish to proceed with:
1. Continue shopping.
2. Proceed to checkout.
3. Change quantity of a product.
4. Delete a product.
1
```

9. Menu displayed after login



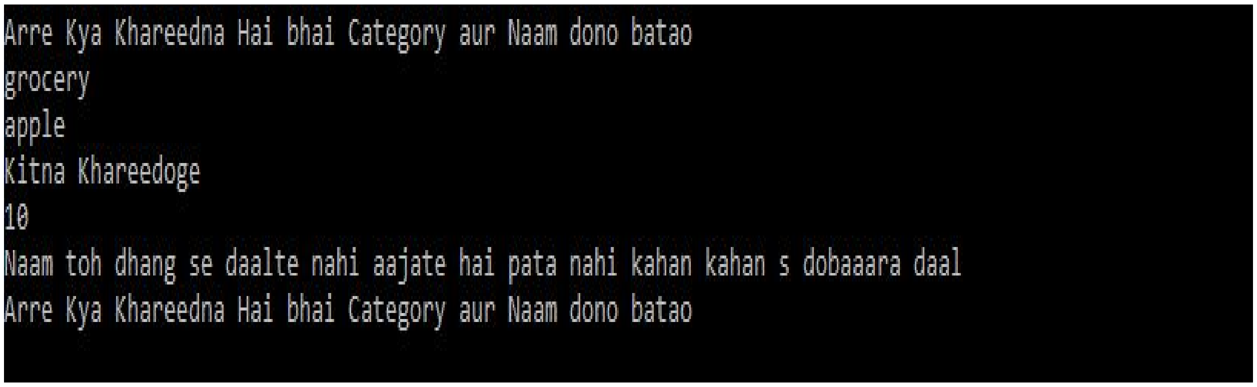
```
Select the function you wish to proceed with:
1. Continue shopping.
2. Proceed to checkout.
3. Change quantity of a product.
4. Delete a product.
1
Hi Alphabeta!
What you want to do today
1. Press 1 to go to cart
2. Press 2 to search for items
3. Press 3 to see your profile
4. Logout
```

10. Search for items



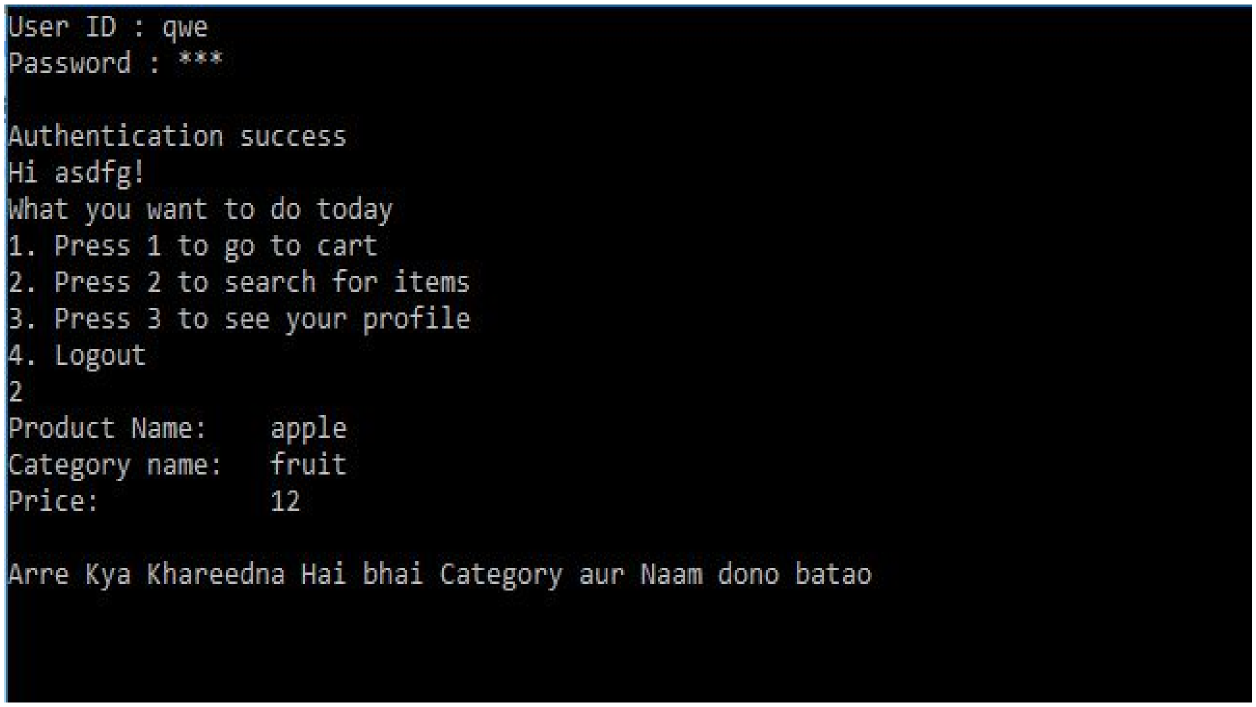
```
Hi Alphabeta!
What you want to do today
1. Press 1 to go to cart
2. Press 2 to search for items
3. Press 3 to see your profile
4. Logout
2
Arre Kya Khareedna Hai bhai Category aur Naam dono batao
```

11. Accepting product required and quantity



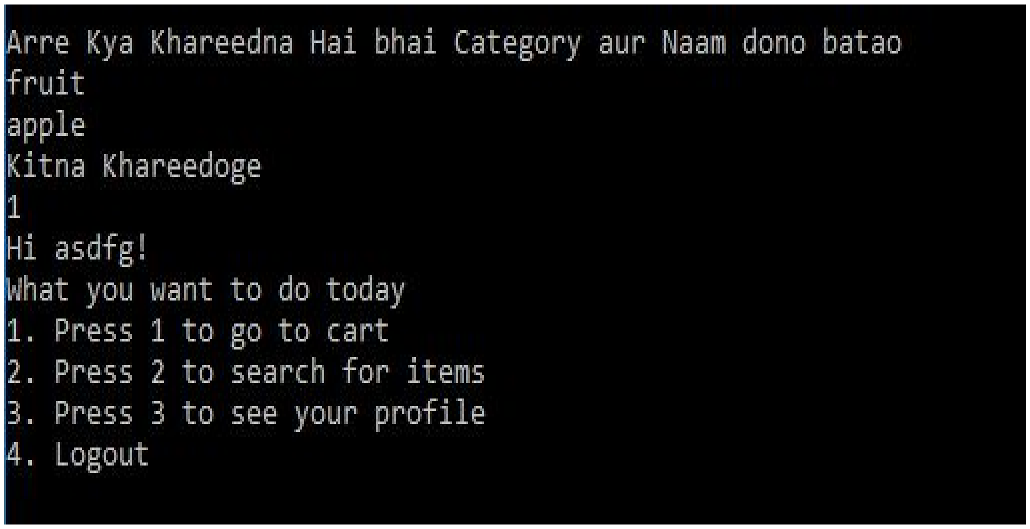
```
Arre Kya Khareedna Hai bhai Category aur Naam dono batao
grocery
apple
Kitna Khareedoge
10
Naam toh dhang se daalte nahi aajate hai pata nahi kahan kahan s dobaaara daal
Arre Kya Khareedna Hai bhai Category aur Naam dono batao
```

12. Vendor-add product



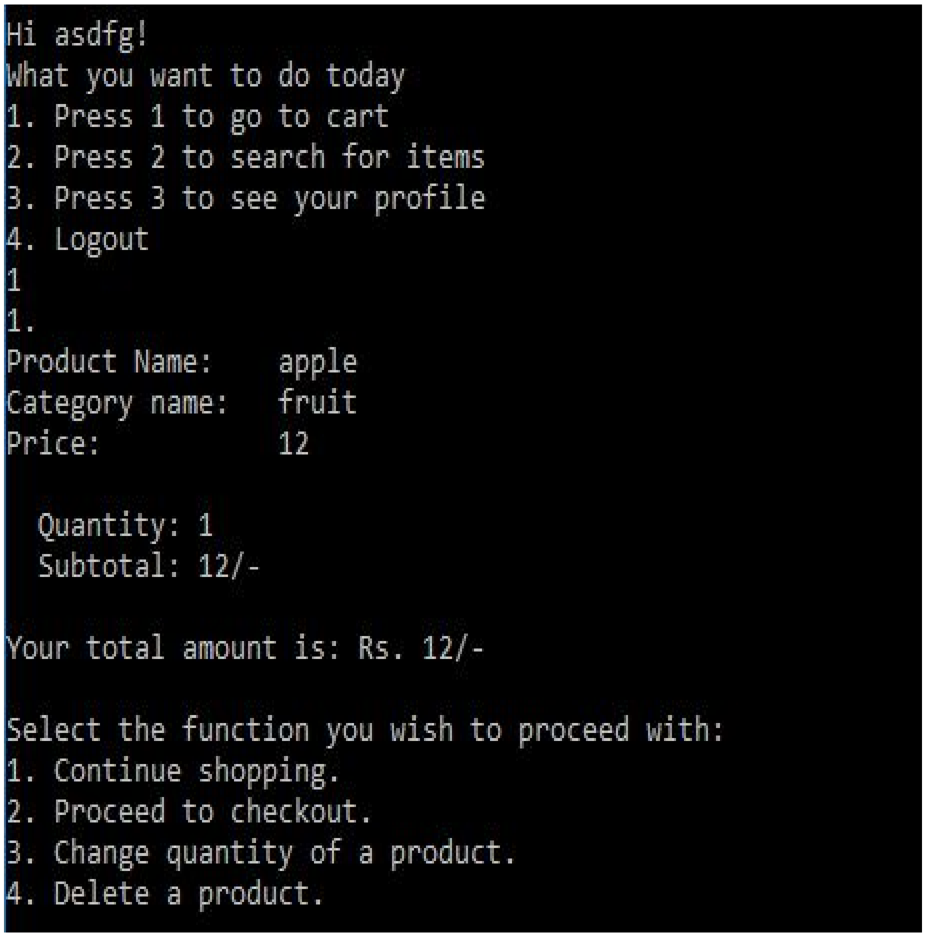
```
User ID : qwe
Password : ***

Authentication success
Hi asdfg!
What you want to do today
1. Press 1 to go to cart
2. Press 2 to search for items
3. Press 3 to see your profile
4. Logout
2
Product Name:     apple
Category name:    fruit
Price:            12

Arre Kya Khareedna Hai bhai Category aur Naam dono batao
```

13. Customer outputs



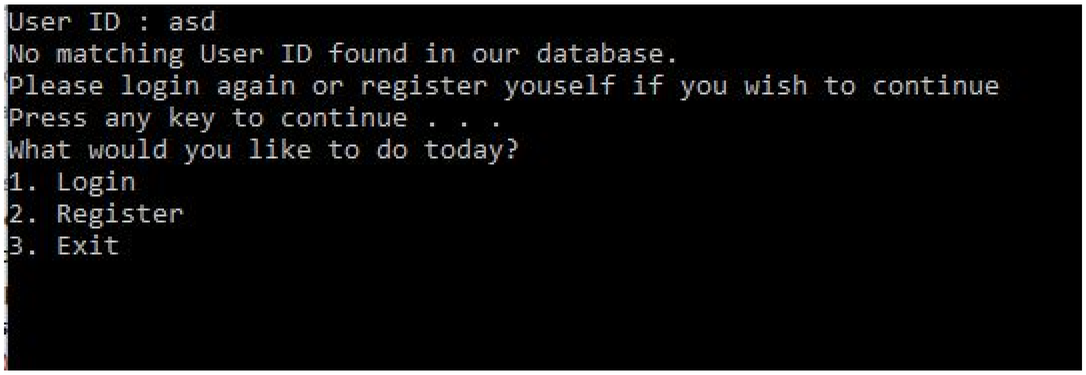
```
Arre Kya Khareedna Hai bhai Category aur Naam dono batao
fruit
apple
Kitna Khareedoge
1
Hi asdfg!
What you want to do today
1. Press 1 to go to cart
2. Press 2 to search for items
3. Press 3 to see your profile
4. Logout
```

14. Cart



```
Hi asdfg!
What you want to do today
1. Press 1 to go to cart
2. Press 2 to search for items
3. Press 3 to see your profile
4. Logout
1
1.
Product Name:     apple
Category name:    fruit
Price:            12

  Quantity: 1
  Subtotal: 12/-

Your total amount is: Rs. 12/-

Select the function you wish to proceed with:
1. Continue shopping.
2. Proceed to checkout.
3. Change quantity of a product.
4. Delete a product.
```

15. No matching user id



```
User ID : asd
No matching User ID found in our database.
Please login again or register youself if you wish to continue
Press any key to continue . . .
What would you like to do today?
1. Login
2. Register
3. Exit
```

Highlights

(Areas of maximum learning outcomes)

1. **Use of map<string, vector <Product> > for stocklist**

The use of such a data structure helps us store data in the form of category, list of product tuples which is easier and faster to traverse.

2. **Using vector <Product *> instead of vector<Product>**

   Currently used in Cart and Customer

3. **Use of Random Generator to generate Captcha()**

The use of a random generator to generate random numbers and produce a captcha of fixed length requires srand() which seeds the timestamps to generate the outputs.

4. **Virtual function profile**

The use of profile function as virtual helps in reduction of code as we can store the pointers to Customer class and Vendor class under a vector of User pointer objects. Helps in reducing code redundancy.

Up-scalabilities

(What can be done to make it closer to a real world entity)

1. **Search Optimization**

   There are several techniques for search optimization, but the one which hit us is the use of Longest Common Subsequence to find the closest match to an input string. For this a simple LCS code needs to be written and wherever the operator == was called for 2 string comparisons, it can be overloaded to call LCS which returns LCS between given string and the input set.

2. **Suggestions/Related Products**

   The solution to this problem can be converted to a graph based algorithm where all the products can be assumed to be nodes/vertices of a complete graph. The edge weights between 2 vertices represents the number of Customers who have purchased both the products.

3. **File Handling/Database**

   The problem of reinitialising the Users and Products as soon as the code is restarted can be frustrating. The use of simple file handling can however resolve this issue. The structures can be copied as it is into a binary or a text file and can be refreshed at the end of the program execution.

4. **Graphical User Interface(GUI)**

   Due to obvious reasons the aesthetics of the code are hindered due to the “Black and white “ terminal window which is plain and boring to look at. The use of the header graphics.h can however make the current code look more interesting.

5. **Trending Products**

   A max_heap containing all the products in terms of number of items can be stored. This heap can return the maximum element in constant time of operation.

Bibliography

1. https://www.amazon.in/ - Idea reference
2. http://www.geeksforgeeks.org/ - Coding reference
3. https://stackoverflow.com/ - Coding reference
4. http://en.cppreference.com/w/ - Coding reference
5. http://www.stroustrup.com/ - Coding reference