PROJECT REPORT

STORE MANAGEMENT SYSTEM

Name : AKASH

UID : 24BCA20008

Section : 24BCV-1A

Subject : OOPS

Subject Code : 24CAH-201

Name : KESHAV SHARMA

UID : 24BCA20010

Section : 24BCV-1A

Subject : OOPS

Subject Code : 24CAH-201

Submitted to : Mr. Jitendra

Designation : Assistant Professor

Signature :

INTRODUCTION

The **Store Management System with Customer Billing** is a C++ console-based application designed to manage a retail store's inventory and customer billing efficiently. The project demonstrates **Object-Oriented Programming (OOP)** principles including inheritance, encapsulation, and dynamic memory management.

This system allows store owners to maintain product inventory, handle discounted products, and generate customer bills with automatic tax calculation. A clean and interactive console interface provides ease of use.

PURPOSE / OBJECTIVES

- Automate inventory management and billing.
- Add, update, delete, and view products efficiently.
- Support discounted products and calculate totals correctly.
- Provide accurate billing including subtotal, GST, and grand total.
- Minimize manual errors and simplify store operations.

FEATURES

- **Inventory Management:**
  - Add, update, delete, view products.
  - Support for discounted products.
- **Customer Billing:**
  - Add products to bill and validate quantity.
  - Display itemized bills with subtotal, GST, and grand total.
  - Clear bill functionality after purchase.
- **User Interface:**
  - Simple, text-based menu-driven interface.
  - Handles invalid input and stock validation.
- **Scalable Design:**
  - OOP principles with classes and inheritance.
  - Dynamic memory management.

SCOPE OF THE PROJECT

- Useful for small retail stores to manage products and billing.
- Helps learn practical **C++ programming** and OOP concepts.
- Can be extended to include file/database storage and GUI-based interfaces.

Future Enhancements:

- Persistent storage using files or databases.
- Advanced billing with offers, multiple taxes, or loyalty points.
- GUI application for enhanced user experience.

DATA STRUCTURES USED

- **Vector (std::vector)**: Dynamic storage of products and bill items.
- **Classes & Inheritance:**
    - Product (base class)
    - DiscountedProduct (derived class)
    - Inventory for managing products
    - BillItem and Bill for billing
- **Primitive Data Types:** int for ID/quantity, double for price/discount/totals
- **Strings:** For product names.
- **Dynamic Memory Allocation:** Products are allocated on heap and deleted in destructor.

CODE IMPLEMENTATION

```
#include <iostream>
#include <vector>
#include <string>
#include <iomanip>
using namespace std;
// Indian Rupee Symbol
#define RUPEE "₹"
// Product Classes
class Product {
protected:
    int id;
    string name;
    double price;
    int quantity;
public:
    Product(int id = 0, string name = "", double price = 0.0, int quantity = 0)
        : id(id), name(name), price(price), quantity(quantity) {}
    virtual ~Product() {}
    int getId() const { return id; }
    string getName() const { return name; }
    double getPrice() const { return price; }
    int getQuantity() const { return quantity; }
    void setName(string newName) { name = newName; }
    void setPrice(double newPrice) { price = newPrice; }
    void setQuantity(int newQuantity) { quantity = newQuantity; }
    virtual void display() const {
        cout << "ID: " << id << " | Name: " << name
             << " | Price: " << RUPEE << price
             << " | Quantity: " << quantity;
    }
    virtual double calculateTotal(int qty) const { return price * qty; }
};
class DiscountedProduct : public Product {
    double discountPercent;
public:
    DiscountedProduct(int id, string name, double price, int quantity, double discount)
        : Product(id, name, price, quantity), discountPercent(discount) {}
    void display() const override {
        cout << "ID: " << id << " | Name: " << name
             << " | Price: " << RUPEE << price
             << " | Quantity: " << quantity
             << " | Discount: " << discountPercent << "%";
    }
    double calculateTotal(int qty) const override {
        double total = price * qty;
        return total - (total * discountPercent / 100);
    }
};
// Inventory
class Inventory {
    vector<Product*> products;
public:
    ~Inventory() {
        for (auto p : products) delete p;
    }
    void addProduct(Product* p) {
        products.push_back(p);
        cout << "Product added successfully!\n";
    }
    void displayAll() const {
        if (products.empty()) {
            cout << "No products in inventory.\n";
            return;
        }
        cout << "\n=== PRODUCTS ===\n";
        for (auto p : products) {
            p->display();
            cout << endl;
        }
    }
    Product* findProductById(int id) {
        for (auto p : products)
            if (p->getId() == id) return p;
        return nullptr;
    }
    void updateProduct(int id, string name, double price, int qty) {
        Product* p = findProductById(id);
        if (p) {
            p->setName(name);
            p->setPrice(price);
            p->setQuantity(qty);
            cout << "Product updated successfully!\n";
        } else {
            cout << "Product not found!\n";
        }
    }
    void deleteProduct(int id) {
        for (auto it = products.begin(); it != products.end(); ++it) {
            if ((*it)->getId() == id) {
                delete *it;
                products.erase(it);
                cout << "Product deleted successfully!\n";
                return;
            }
        }
        cout << "Product not found!\n";
    }
};
// Bill Item
class BillItem {
    Product* product;
    int quantity;
public:
    BillItem(Product* p, int qty) : product(p), quantity(qty) {}
    double getTotal() const { return product ? product->calculateTotal(quantity) : 0.0; }
    void display() const { cout << product->getName() << " x " << quantity
                             << " = " << RUPEE << fixed << setprecision(2) << getTotal() << endl; }
```

```
};

// Bill
class Bill {
   vector<BillItem> items;
   double taxRate;
public:
   Bill(double tax = 5.0) : taxRate(tax) {}
   void addProduct(Product* p, int qty = 1) {
      items.push_back(BillItem(p, qty));
   }
   double getSubtotal() const {
      double total = 0; for (auto i : items) total += i.getTotal(); return total;
   }
   double getTax() const { return getSubtotal() * taxRate / 100; }
   double getGrandTotal() const { return getSubtotal() + getTax(); }
   void display() const {
      if (items.empty()) { cout << "No items in the bill!\n"; return; }
      cout << "\n=== CUSTOMER BILL ===\n";
      for (auto i : items) i.display();
      cout << "Subtotal: " << RUPEE << getSubtotal() << "\n";
      cout << "GST (" << taxRate << "%): " << RUPEE << getTax() << "\n";
      cout << "Grand Total: " << RUPEE << getGrandTotal() << "\n";
   }
   void clear() { items.clear(); cout << "Bill cleared!\n"; }
};
// Main Menu
void mainMenu(Inventory& inventory) {
   int choice;
   do {
      cout << "\n=== MAIN MENU ===\n";
      cout << "1. Add Product\n";
      cout << "2. Update Product\n";
      cout << "3. View Products\n";
      cout << "4. Delete Product\n";
      cout << "5. Customer Billing\n";
      cout << "6. Exit\n";
      cout << "Choose option: ";
      cin >> choice;
      switch (choice) {
        case 1: {
          int id, qty, type;
          string name;
          double price, discount;
          cout << "Enter ID: "; cin >> id;
          cin.ignore();
          cout << "Enter Name: "; getline(cin, name);
          cout << "Enter Price: " << RUPEE; cin >> price;
          cout << "Enter Quantity: "; cin >> qty;
          cout << "Type (1-Regular, 2-Discounted): "; cin >> type;
          if (type == 2) {
            cout << "Enter Discount %: "; cin >> discount;
            inventory.addProduct(new DiscountedProduct(id, name, price, qty, discount));
          } else inventory.addProduct(new Product(id, name, price, qty));
          break;
        }
        case 2: {
          int id, qty; string name; double price;
          cout << "Enter Product ID to update: "; cin >> id;
          cin.ignore();
          cout << "Enter New Name: "; getline(cin, name);
          cout << "Enter New Price: " << RUPEE; cin >> price;
          cout << "Enter New Quantity: "; cin >> qty;
          inventory.updateProduct(id, name, price, qty);
          break;
        }
        case 3:
          inventory.displayAll();
          break;
        case 4: {
          int id; cout << "Enter Product ID to delete: "; cin >> id;
          inventory.deleteProduct(id);
          break;
        }
        case 5: {
          Bill bill;
          int subChoice;
          do {
            cout << "\n=== CUSTOMER BILLING MENU ===\n";
            cout << "1. View Products\n";
            cout << "2. Add Product to Bill\n";
            cout << "3. View Bill\n";
            cout << "4. Clear Bill\n";
            cout << "5. Back to Main Menu\n";
            cout << "Choose option: ";
            cin >> subChoice;
            switch(subChoice) {
                case 1: inventory.displayAll(); break;
                case 2: {
                    int id, qty;
                    cout << "Enter Product ID: "; cin >> id;
                    Product* p = inventory.findProductById(id);
                    if (!p) { cout << "Product not found!\n"; break; }
                    cout << "Enter Quantity: "; cin >> qty;
                    if (qty > p->getQuantity()) { cout << "Insufficient stock!\n"; break; }
                    bill.addProduct(p, qty);
```

```
                            cout << "Product added to bill!\n";
                            break;
                        }
                        case 3: bill.display(); break;
                        case 4: bill.clear(); break;
                        case 5: cout << "Returning to Main Menu...\n"; break;
                        default: cout << "Invalid choice!\n";
                    }
                } while(subChoice != 5);
                break;
            }
            case 6:
                cout << "Exiting... Goodbye!\n"; break;
            default: cout << "Invalid choice!\n";
        }
    } while(choice != 6);
}
int main() {
    Inventory inventory;
    mainMenu(inventory);
    return 0;
}
```

OUTPUT

```
=== MAIN MENU ===
1. Add Product
2. Update Product
3. View Products
4. Delete Product
5. Customer Billing
6. Exit
```

```
Choose option: 1
Enter ID: 101
Enter Name: Laptop
Enter Price: ₹45000
Enter Quantity: 5
Type (1-Regular, 2-Discounted): 2
Enter Discount %: 10
Product added successfully!
```

```
=== CUSTOMER MENU ===
1.View Products
2.Add to Bill
3.View Bill
4.Clear Bill
5.Back
Choose: 1

=== PRODUCTS ===
ID:101 | Name:Laptop | Price:₹45000 | Qty:5 | Discount:10%
ID:102 | Name:Mouse | Price:₹500 | Qty:20

=== CUSTOMER MENU ===
Choose: 2
ID: 101
Qty: 2
Added to bill!
```

CONCLUSION

The Store Management System with Customer Billing in C++ provides a complete solution for managing inventory and generating customer bills in a retail environment.

Key achievements:

- Implemented a modular OOP design using classes and inheritance.
- Enabled efficient inventory management and discount handling.
- Provided an intuitive menu system with emoji-based visual cues.
- Ensured accurate billing with automatic tax and discount calculation.

This project is a solid foundation for building more advanced retail management systems with database integration and graphical interfaces.