Fast Food Brand Management System

# 1. Project Overview

This C++ console-based application is designed to manage inventory (specifically food items) and employee records for a small business. It includes features to add, update, and view data, track sales, calculate profits, and persist data using file handling.

# 2. Objectives

* Implement a basic Inventory Management System using Object-Oriented Programming.
* Track food stock, sales, and profits.
* Maintain employee records.
* Utilize file I/O for data persistence.
* Demonstrate object-oriented programming (OOP) concepts, including inheritance, polymorphism, encapsulation, and abstraction.

# 3. Tools & Technologies Used

* Language: C++
* Software: Dev-C++
* Standard Libraries: <iostream>, <fstream>, <vector>, <string>, <iomanip>

# 4. OOP Concepts Applied

|  |  |
| --- | --- |
| **Concept** | **Applied Through** |
| Inheritance | Food and Employee classes inherit from Item |
| Polymorphism | Virtual methods input() and display() in base and derived classes |
| Encapsulation | Number of member functions and data members enclosed in single unit |
| File handling | Sequential file handling is use for storing data of food and employee |

# 5. Flowchart: C:\Users\ADMIN\AppData\Local\Packages\5319275A.WhatsAppDesktop_cv1g1gvanyjgm\TempState\E2BBB6C289A1F6FC299B4C365E04EA7C\WhatsApp Image 2025-06-03 at 16.35.54_adf41185.jpg

# 6. Functional Description / Module Breakdown

**1. Template function:**

1.template <typename T> void displayList(const vector<T>& list)

Purpose: Generic function to display a list of items that have a display() method.

Used For: Displaying collections like foods or employees.

---

**2. Base Class: Item**

void input()

Purpose: Inputs ID and name of an item.

Overridden By: Food and Employee

void display() const

Purpose: Displays ID and name.

Overridden By: Food and Employee.

Getters

int getId() const

string getName() const

---

**3. Derived Class: Food : public Item**

void input() override

Purpose: Inputs food-specific details (cost price, selling price, stock).

void display() const override

Purpose: Displays all attributes of a food item in tabular format.

bool recordSale(int qty)

Purpose: Deducts from stock, adds to sold count.

Returns: True if stock is sufficient.

bool cancelSale(int qty)

Purpose: Reverts a sale if already sold.

Returns: True if cancel is possible.

float getProfit() const

Purpose: Calculates total profit based on units sold.

void saveToFile(ofstream& out) const

Purpose: Saves food data to file foods.txt.

void loadFromFile(ifstream& in)

Purpose: Loads food data from file.

Additional Getters

float getCostPrice() const, getSellingPrice() const, getStock() const, getSold() const

---

**4. Derived Class: Employee : public Item**

void input() override

Purpose: Inputs employee-specific data (position, salary).

void display() const override

Purpose: Displays employee data in tabular format.

void saveToFile(ofstream& out) const

Purpose: Saves employee data to file.

void loadFromFile(ifstream& in)

Purpose: Loads employee data from file.

---

**5. Data Vectors**

vector<Food> foods;

vector<Employee> employees;

---

**6. File I/O Functions**

void saveFoods()

Saves all Food objects to foods.txt.

void saveEmployees()

Saves all Employee objects to employees.txt.

void loadFoods()

Loads Food objects from file.

If no data is found, prompts for new input.

void loadEmployees()

Loads Employee objects from file.

---

**7. Food Management Functions**

void addFood()

Adds a new food item to foods vector.

void updateFood()

Updates food info for a given ID.

void viewStock()

Displays current stock with full food details.

void recordSale()

Records a sale for a given food ID and quantity.

void cancelSale()

Cancels a recorded sale for a given ID and quantity.

void showProfit()

Displays total profit from all food sales.

---

**8. Employee Management Functions**

void addEmployee()

Adds a new employee to the list.

void viewEmployees()

Displays all employees in tabular form.

---

**9. Main Menu: main()**

Purpose: User interaction and choice handling.

Menu Options:

1. View stock

2. Add food

3. Update food

4. Record sale

5. Cancel sale

6. Show profit

7. Add employee

8. View employees

9. Save and exit

# 8. Code Snippets

## Inheritance & Polymorphism

class Item {  
Protected:  
 int id;  
 string name;  
"  
 "public:  
 virtual void input();  
 virtual void display() const;  
 virtual ~Item() {}  
};

## Template Function

template <typename T>  
void displayList(const vector<T>& list)  
{  
"  
 " for (const auto& item : list) {  
 item.display();  
 }  
}

## File Handling

void saveFoods() {  
 ofstream out("foods.txt");  
"  
 " for (const auto& f : foods) f.saveToFile(out);  
}

# 9. Challenges Faced

* Managing file I/O with mixed delimiter formats (space and '|') required careful parsing.
* Ensuring consistency in loadFromFile() methods for both Food and Employee.
* Maintaining user-friendly input/output while preserving OOP structure.

# 10. Conclusion

The project effectively demonstrates real-world application of OOP principles in C++. It provides an extendable and modular structure for managing inventory and employee records, with user interaction via a command-line interface and data persistence through files.

# 11. References

* https://www.scribd.com/document/312090200/Fast-Food-Management-System
* Classroom lectures and notes