SMARTVEND++

VENDING MACHINE

DEVELOPMENT DOCUMENT

Table of Contents

1. Specification

This project was developed as part of the *Code Lab 1* module at Bath Spa University, Elizabeth School of London. The task was to create an interactive console-based vending machine in C++ that accurately simulates the real-world process of product selection, payment handling, stock control, and customer interaction (W3Schools, 2024).

The resulting system, **SmartVend++**, is a refined and extended version of earlier vending machine programs. It offers improved performance, robust handling of user inputs, and additional business-oriented features. The program is built using **modular programming** to make the code maintainable and easy to extend (Geeks for Geeks, 2024). It also uses **real-time input validation** to prevent errors (Programiz, 2024) and an **intelligent upselling feature** to encourage additional purchases.

Key Features:

- Categorised menu with unique product IDs (Cplusplus.com, 2024)
- Cash and contactless card payment options (Microsoft Docs, 2024)
- Automatic 10% discount calculation and change handling (Programiz, 2024)
- Coffee purchase upsell for biscuits to boost sales potential
- Real-time stock control preventing overselling (Tutorials Point, 2024)
- Comprehensive error handling for a smooth user experience (Stack Overflow, 2023)
- Persistent storage via multiple external data files (Tutorials Point, 2024)

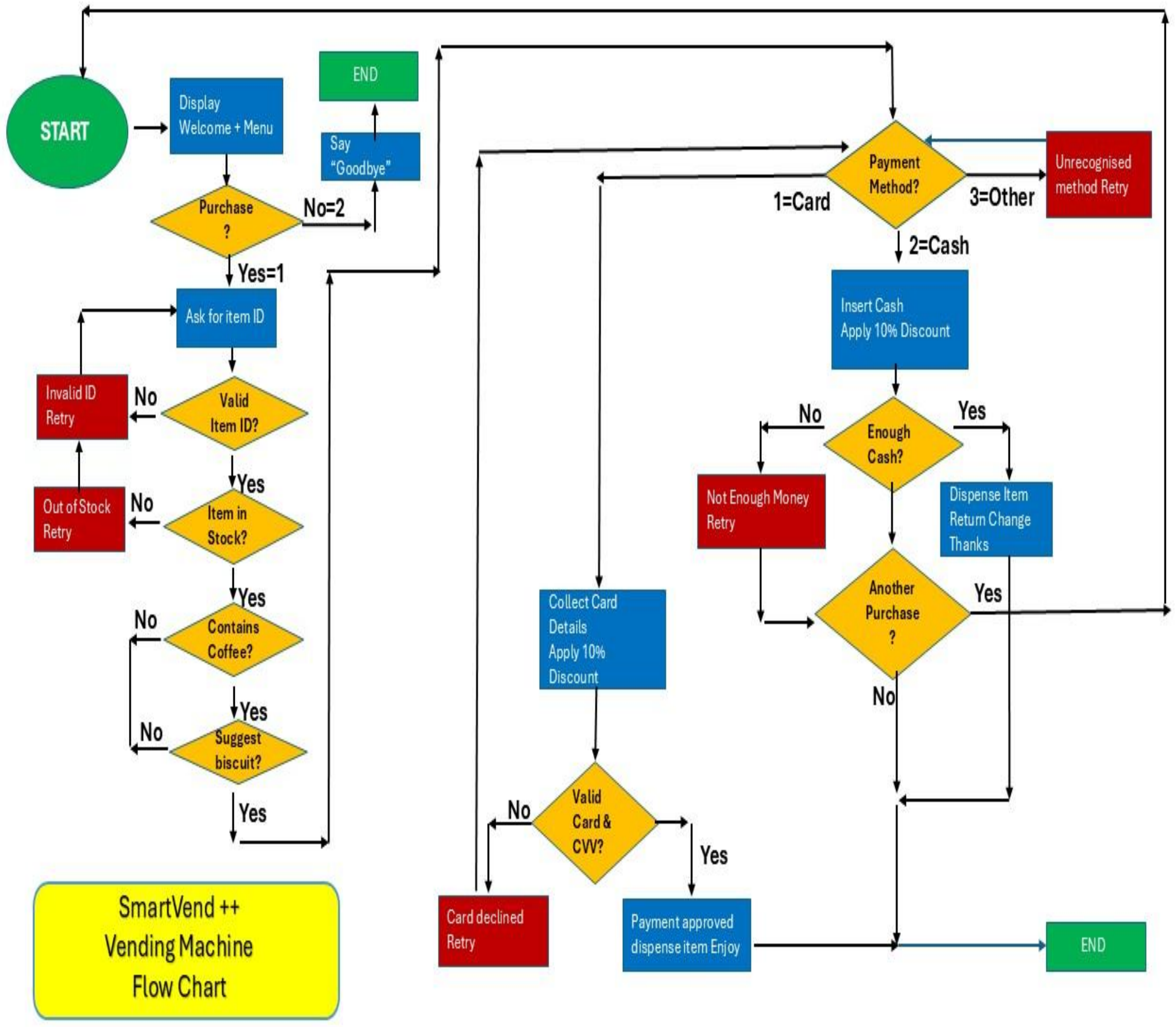
1.1 GitHub Link: www.github.com

2. System Flowchart

The SmartVend++ program follows a structured sequence of operations to ensure efficiency and prevent errors:

1. **Display Menu** – The system shows all available products in a clear, organised format.
2. **User Selection** – The user chooses a product using its ID number.
3. **Stock Check** – The system checks if the chosen item is available (Stack Overflow, 2023).
4. **Upsell Offer** – If coffee is selected, the system offers biscuits as an add-on.
5. **Select Payment Method** – The user chooses between card or cash (Microsoft Docs, 2024).
6. **Process Payment** – The system validates and processes the payment.
7. **Dispense Product & Change** – The product is given and change returned if necessary.
8. **Repeat or Exit** – The user can choose to make another purchase or exit.

Flow:
Display Menu → User Selection → Stock Check → Upsell Offer → Payment Method → Payment Processing → Product & Change Dispensed → Exit



3. Technical Description & Walkthrough

A full walkthrough video has been prepared demonstrating the development, logic, and functionality of the SmartVend++ vending machine system. This video includes a live demonstration of the program in use, covering everything from the menu interface to the payment and stock management features. It also contains a narrated technical explanation of how the key parts of the code operate and interact.

3.1 YouTube Walkthrough Video link: www.youtube.com

The walkthrough lasts approximately 7 minutes and provides insight into the implementation choices, structure, and logic used throughout the development of SmartVend++.

3.2 System Design

SmartVend++ uses object-oriented programming (OOP), which divides the system into logical units for better readability, maintainability, and scalability (Geeks for Geeks, 2024; W3Schools, 2024).

Product Class – Represents each vending machine item, storing attributes such as name, category, price, and stock (Tutorials Point, 2024).

- isAvailable(): Checks if stock is greater than zero.
- purchase(): Reduces stock by one.

Vending Machine Class – Manages all system functions:

- Displays the menu using <iomanip> for proper formatting (Cplusplus.com, 2024).
- Uses vector<Product> for flexible product storage (IBM Developer, 2023).
- Handles purchase logic, input validation, payment, and upselling (Programiz, 2024).

3.3 Menu Display

The menu is displayed as a formatted table with ID, name, category, price, and stock. The <iomanip> library aligns columns and ensures pricing is consistently shown to two decimal places (Cplusplus.com, 2024). This makes the interface clear and professional.

3.4 Input Validation and Error Handling

Two helper functions control user input:

- getIntInput() – Validates that input is a whole number (Programiz, 2024).
- getDoubleInput() – Validates decimal numbers for payments.

Invalid entries are handled with cin.clear() and cin.ignore() to reset the input stream and prompt again (Stack Overflow, 2023). This prevents crashes and ensures a smooth user experience.

3.5 Buying Process and Upselling

Once an item is selected, the system checks stock. If coffee is chosen, biscuits are offered as an add-on purchase, provided they are in stock (Programiz, 2024). This is a basic implementation of **suggestive selling** used in retail to increase revenue.

3.6 Discount Functionality

Every transaction automatically applies a 10% discount using:

cpp
Copy code
```
double price = selected.price * 0.90;
```

(Cplusplus.com, 2024)
This provides a consistent incentive for purchases and improves customer satisfaction.

3.7 Payment Processing

Two payment methods are available:

- **Card Payments** – Validates that the card number has 16 digits and the CVV has 3 digits (Microsoft Docs, 2024).
- **Cash Payments** – Ensures the inserted amount meets or exceeds the total price and calculates change if necessary (Programiz, 2024).

3.8 File Storage and Data Persistence

The program uses several external files for storage:

- items.txt – Product inventory
- sales.txt – Sales records
- restock.txt – Stock replenishment logs
- settings.txt – Runtime settings like discount rate
- error.txt – Logs error messages
- change_log.txt – Records of changes and updates

All file handling is performed using C++ file streams for efficiency and reliability (Tutorials Point, 2024).

3.9 Testing

Testing included:

- Invalid product ID entry
- Out-of-stock selection
- Insufficient payment
- Invalid card details
- Coffee purchase triggering biscuit upsell

These tests confirmed that the system responds correctly in all expected scenarios.

3.10 Future Development Opportunities

The system is modular and can be expanded with features such as:

- GUI development with Qt (Qt Documentation, 2024)
- Database integration for persistent inventory (IBM Developer, 2023)
- Multi-user accounts and purchase history tracking

4. Critical Reflection

Developing SmartVend++ strengthened my understanding of **C++ OOP principles** (Geeks for Geeks, 2024; W3Schools, 2024), **file handling** (Tutorials Point, 2024), and **STL containers** (IBM Developer, 2023). The robust **input validation** (Programiz, 2024) and **error handling** (Stack Overflow, 2023) improved program reliability, while the discount and upselling features demonstrated commercial thinking in design.

Areas for improvement include:

- Persistent inventory updates in items.txt (Tutorials Point, 2024)
- More advanced card validation logic
- GUI interface for improved usability (Qt Documentation, 2024)
- Database integration for real-time stock tracking (IBM Developer, 2023)
- Automated testing for long-term stability (OpenAI, 2023)
- SQLite or JSON databases (persistent storage)
- Modular design patterns in C++
- Unit testing and mock frameworks
- Higher level C++ syntax and usage of STL

Overall, SmartVend++ meets the project requirements and provides a professional, user-friendly UI/UX, actual business logic, and scalable foundation for future development (Geeks for Geeks, 2024).

5. References

Cplusplus.com (2024) *std::vector - C++ Reference*. Available at: https://cplusplus.com/reference/vector/vector/ (Accessed: 6 August 2025).

Geeks for Geeks (2024) *Object-Oriented Programming in C++*. Available at: https://www.geeksforgeeks.org/object-oriented-programming-in-cpp/ (Accessed: 6 August 2025).

Microsoft Docs (2024) *Input and Output with Streams in C++*. Available at: https://learn.microsoft.com/en-us/cpp/standard-library/input-output-streams (Accessed: 6 August 2025).

W3Schools (2024) *C++ Classes and Objects*. Available at: https://www.w3schools.com/cpp/cpp_classes.asp (Accessed: 6 August 2025).

Tutorials Point (2024) *C++ Files and Streams*. Available at: https://www.tutorialspoint.com/cplusplus/cpp_files_streams.htm (Accessed: 6 August 2025).

Stack Overflow (2023) *How to flush cin in C++*. Available at: https://stackoverflow.com/questions/257091/how-do-i-flush-the-cin-buffer (Accessed: 6 August 2025).

Programiz (2024) *C++ if, if...else and Nested if...else*. Available at: https://www.programiz.com/cpp-programming/if-else (Accessed: 6 August 2025).

IBM Developer (2023) *C++ Standard Template Library (STL)*. Available at: https://developer.ibm.com/articles/standard-template-library/ (Accessed: 6 August 2025).

Qt Documentation (2024) *Getting Started with Qt*. Available at: https://doc.qt.io/qt-6/gettingstarted.html (Accessed: 6 August 2025).

OpenAI (2023) *ChatGPT for Coding Assistance*. Available at: https://openai.com/chatgpt (Accessed: 6 August 2025).

6.Appendices

Appendix A: Vending Machine main.cpp

```
// SmartVend ++ Vending Machine System

// Created by Ozgur Serin | Elizabeth School of London - Bath Spa University

// This project simulates a vending machine that sells snacks and drinks

// It includes payment handling, stock management, and suggestion features


#include <iostream>   // Needed for input and output like cout and cin

#include <iomanip>    // Lets us format decimal values (like setting 2 decimal places)

// 'string' lets us work with words and text instead of just numbers

#include <string>     // So we can use string types for names and categories

// 'vector' is like a flexible list that can grow or shrink to hold items

#include <vector>     // Gives us the vector container to store product lists

#include <limits>     // Helps us clear invalid inputs from the user buffer


using namespace std;  // executes this step



// --- Product Class: Handles individual item properties and behaviour ---

// This class defines a product inside the vending machine.

// Each product has a name, category (like Drink or Snack), price, and stock level.

// It includes behaviour like checking availability and reducing stock after a purchase.

// This class sets up each item (product) in the vending machine
// 'class' lets us group data (like name/price) and actions (methods) into one object
class Product {
// 'public' means other parts of the program can use these members (like name or purchase())
public:
  string name;       // Product name (e.g. Cola, Crisps)
  string category;    // What type of product – Drink, Snack, etc
  double price;      // Cost of the item in pounds
  int stock;         // How many of this item are available

  // This sets up the values for a product when it's created
  Product(string name, string category, double price, int stock)
    : name(name), category(category), price(price), stock(stock) {}

  // This just checks if the item is still in stock
  bool isAvailable() const { return stock > 0; }  // executes this step

  // After purchase, reduce the stock by 1
// 'void' means this function doesn't return any value
  void purchase() { if (stock > 0) stock--; }  // executes this step
};  // executes this step


// --- VendingMachine Class: Manages item list, purchasing, and display ---
// The main vending machine class encapsulates all operations such as:
// - Displaying items
// - Handling user input
// - Managing purchases and stock
// - Interfacing with payment logic

// This is the main machine class which manages everything

// This class holds the vending machine's items and handles all customer actions

class VendingMachine {

// 'private' means these members are only used inside the class, not accessible from
outside

private:

// This is a list that holds all the products in the vending machine

  vector<Product> items;  // List of all the products in the machine


// 'public' means other parts of the program can use these members (like name or
purchase())

public:


// --- Constructor: Initialise vending machine items ---

// This constructor adds a predefined list of drinks and snacks to the machine.

// These items will appear on the menu when the program runs.

  // This is where we add all the items to the machine

  VendingMachine() {

    items.emplace_back("Water", "Drink", 1.00, 10);  // executes this step

    items.emplace_back("Fanta", "Drink", 1.50, 8);  // executes this step

    items.emplace_back("Cola", "Drink", 1.50, 7);  // executes this step

    items.emplace_back("Black Coffee", "Hot Drink", 1.80, 5);  // executes this step

    items.emplace_back("White Coffee", "Hot Drink", 1.80, 5);  // executes this step

    items.emplace_back("Crisps", "Snack", 1.20, 6);  // executes this step

    items.emplace_back("Chocolate", "Snack", 1.80, 4);  // executes this step

    items.emplace_back("Biscuits", "Snack", 1.50, 3);  // executes this step

  }
```

```
// 'void' means this function doesn’t return any value
  void start() {
    cout << "\nWelcome to SmartVend++ by Ozgur Serin \n";  // executes this step
    cout << "Providing tasty snacks and drinks on demand!\n\n";  // executes this step

    bool running = true;  // executes this step
// 'while' runs the loop repeatedly while the condition is true
    while (running) {
      showMenu();  // executes this step
      cout << "\nWould you like to make a purchase? (1 = Yes, 2 = Exit): ";  // executes
this step
      int action = getIntInput();  // executes this step

// 'if' checks a condition. If it’s true, the next block runs
      if (action == 1) {
        handlePurchase();  // executes this step
// 'if' checks a condition. If it’s true, the next block runs
      } else if (action == 2) {
        cout << "\nCheers for using SmartVend++! Have a lovely day.\n";  // executes this
step
        running = false;  // executes this step
// 'else' runs when no other 'if' or 'else if' conditions are true
      } else {
        cout << "\nSorry, that's not a valid option. Try again.\n";  // executes this step
      }
    }
  }
```

```
// 'private' means these members are only used inside the class, not accessible from
outside
private:
// 'void' means this function doesn’t return any value
  void showMenu() {
    cout << "================ VENDING MENU ================\n";  // executes
this step
// 'cout' is used to display output to the user
    cout << left << setw(5) << "ID" << setw(20) << "Item"
         << setw(12) << "Category" << setw(7) << "Price" << "Stock" << endl;  // executes
this step
    cout << "--------------------------------------------\n";  // executes this step
// 'for' loop goes over a range or collection
    for (size_t i = 0; i < items.size(); ++i) {  // executes this step
// 'cout' is used to display output to the user
      cout << setw(5) << i + 1
           << setw(20) << items[i].name
           << setw(12) << items[i].category
// 'setprecision' helps format decimal places when printing prices
           << "£" << fixed << setprecision(2) << setw(6) << items[i].price
           << items[i].stock << endl;  // executes this step
    }
  }

// 'void' means this function doesn’t return any value
  void handlePurchase() {
// This is the heart of the vending logic where item selection and payment happen.
    cout << "\nEnter the item ID you'd like to buy: ";  // executes this step
    int id = getIntInput();  // executes this step
```

```
// Get the item number the user wants to buy


// 'if' checks a condition. If it's true, the next block runs
    if (id < 1 || id > (int)items.size()) {
// Make sure the user selected a valid item ID
      cout << "\nThat ID doesn't match anything. Try again.\n";  // executes this step
      return;  // executes this step
    }

    Product &selected = items[id - 1];  // executes this step

// 'if' checks a condition. If it's true, the next block runs
    if (!selected.isAvailable()) {
      cout << "\nSorry, we're out of " << selected.name << ".\n";  // executes this step
      return;  // executes this step
    }

// 'if' checks a condition. If it's true, the next block runs
    if (selected.name.find("Coffee") != string::npos) {
      cout << "\nFancy a biscuit with your coffee? (y/n): ";  // executes this step
      char suggestion;  // executes this step
      cin >> suggestion;  // executes this step
      cin.ignore(numeric_limits<streamsize>::max(), '\n');  // executes this step

// 'if' checks a condition. If it's true, the next block runs
      if (tolower(suggestion) == 'y') {
// 'for' loop goes over a range or collection
        for (Product &p : items) {
```

```
// 'if' checks a condition. If it’s true, the next block runs
          if (p.name == "Biscuits" && p.isAvailable()) {
            cout << "Adding biscuits to your order. Cheers!\n";  // executes this step
            p.purchase();  // executes this step
            break;  // executes this step
          }
        }
      }
    }

    cout << "\nWould you like to pay by contactless card or cash? (1 = Card, 2 = Cash): ";
// executes this step
    int method = getIntInput();  // executes this step


    double price = selected.price * 0.90;  // executes this step
    cout << "\nLucky you! You get a 10% discount. Final price: £" << fixed <<
setprecision(2) << price << endl;  // executes this step


// 'if' checks a condition. If it’s true, the next block runs
    if (method == 1) {
      string cardNumber, expiry, cvv;  // executes this step
      cout << "\nTap your card (enter 16-digit number): ";  // executes this step
      cin >> cardNumber;  // executes this step
      cout << "Enter expiry date (MM/YY): ";  // executes this step
      cin >> expiry;  // executes this step
      cout << "Enter 3-digit CVV: ";  // executes this step
      cin >> cvv;  // executes this step
      cin.ignore(numeric_limits<streamsize>::max(), '\n');  // executes this step
```

```
// 'if' checks a condition. If it’s true, the next block runs
      if (cardNumber.length() == 16 && cvv.length() == 3) {
        cout << "\nProcessing card... Please wait...\n";  // executes this step
        cout << "Payment approved.\n";  // executes this step
        selected.purchase();  // executes this step
        cout << "Enjoy your " << selected.name << "!\n";  // executes this step
// 'else' runs when no other 'if' or 'else if' conditions are true
      } else {
        cout << "\nCard declined – details look wrong.\n";  // executes this step
        return;  // executes this step
      }

// 'if' checks a condition. If it’s true, the next block runs
    } else if (method == 2) {
      cout << "\nPlease insert cash: £";  // executes this step
      double inserted = getDoubleInput();  // executes this step

// 'if' checks a condition. If it’s true, the next block runs
      if (inserted < price) {
        cout << "\nNot enough money. Cancelling transaction.\n";  // executes this step
        return;  // executes this step
      }

      selected.purchase();  // executes this step
      double change = inserted - price;  // executes this step
      cout << "\nDispensing: " << selected.name << endl;  // executes this step
// 'if' checks a condition. If it’s true, the next block runs
      if (change > 0) {
        cout << "Returning change: £" << fixed << setprecision(2) << change << endl;  //
executes this step
      }
      cout << "Thanks a lot – enjoy your purchase!\n";  // executes this step
// 'else' runs when no other 'if' or 'else if' conditions are true
    } else {
      cout << "\nThat's not a payment method I recognise.\n";  // executes this step
    }
  }

  int getIntInput() {
    int val;  // executes this step
// 'while' runs the loop repeatedly while the condition is true
    while (!(cin >> val)) {
      cout << "Invalid input. Try again: ";  // executes this step
      cin.clear();  // executes this step
      cin.ignore(numeric_limits<streamsize>::max(), '\n');  // executes this step
    }
    cin.ignore(numeric_limits<streamsize>::max(), '\n');  // executes this step
    return val;  // executes this step
  }

  double getDoubleInput() {
    double val;  // executes this step
// 'while' runs the loop repeatedly while the condition is true
    while (!(cin >> val)) {
      cout << "Invalid amount. Try again: ";  // executes this step
      cin.clear();  // executes this step
```

```
            cin.ignore(numeric_limits<streamsize>::max(), '\n');  // executes this step
        }
        cin.ignore(numeric_limits<streamsize>::max(), '\n');  // executes this step
        return val;  // executes this step
    }
};  // executes this step

// 'int' main function returns 0 if everything went fine
int main() {
    VendingMachine machine;  // executes this step
    machine.start();  // executes this step
    return 0;  // executes this step
}
```