**UNIVERSITY OF BOHOL**

**Tagbilaran City, Bohol, Philippines**

**COLLEGE OF ENGINEERING, TECHNOLOGIES,**

**ARCHITECTURE AND FINE ARTS**

**COMPUTER ENGINEERING**

**Programming Logic and Design**

**CPEP112**

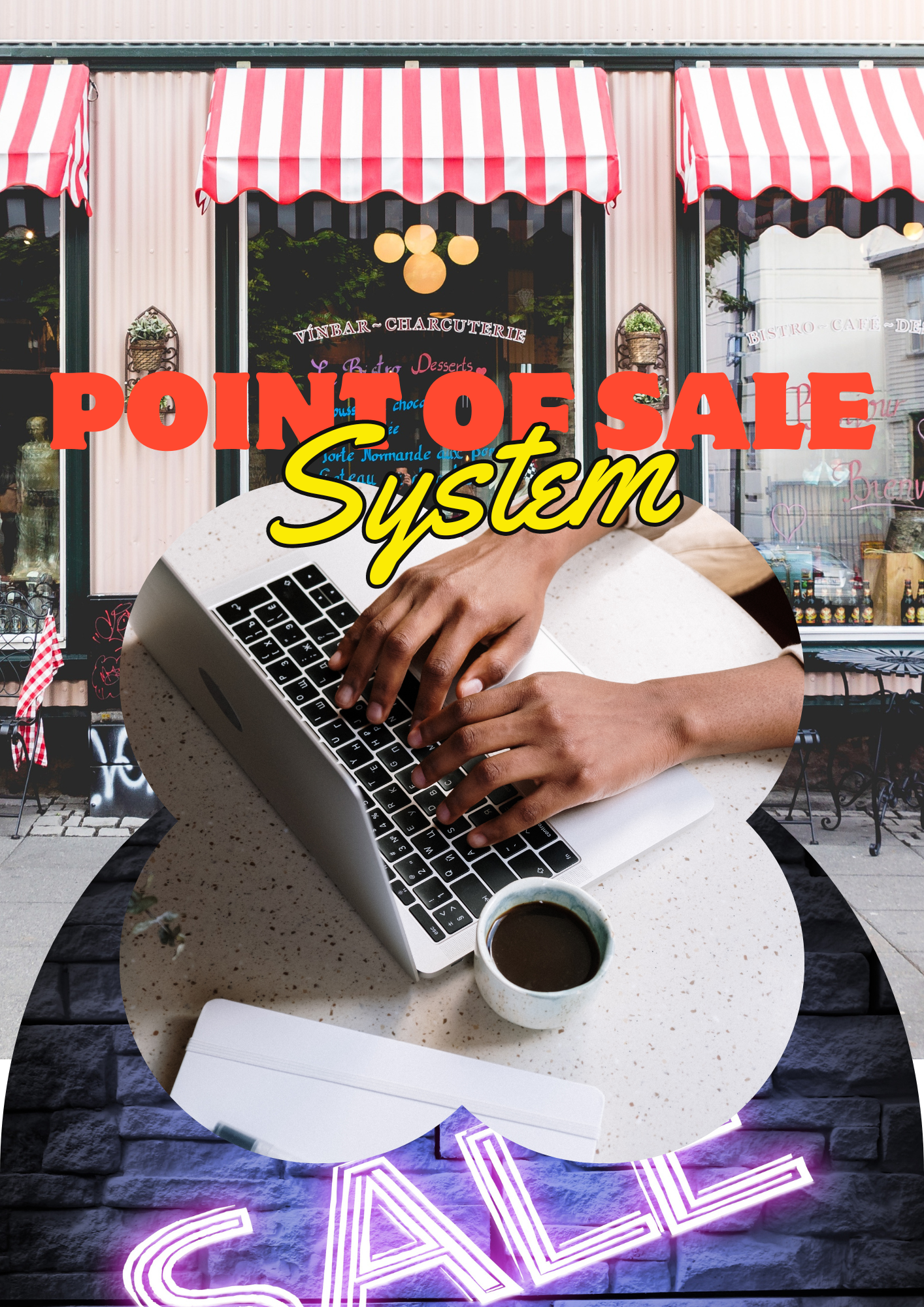
**Submitted by Group 2 & 19**

**CURAYAG, JAY**

**LOMONTAD, ELLA MARIE**

**OCIONES, RUDNIE NIEL**

**FERNANDEZ, RAFAEL**

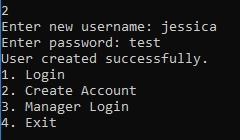
****

**INTRODUCTION**

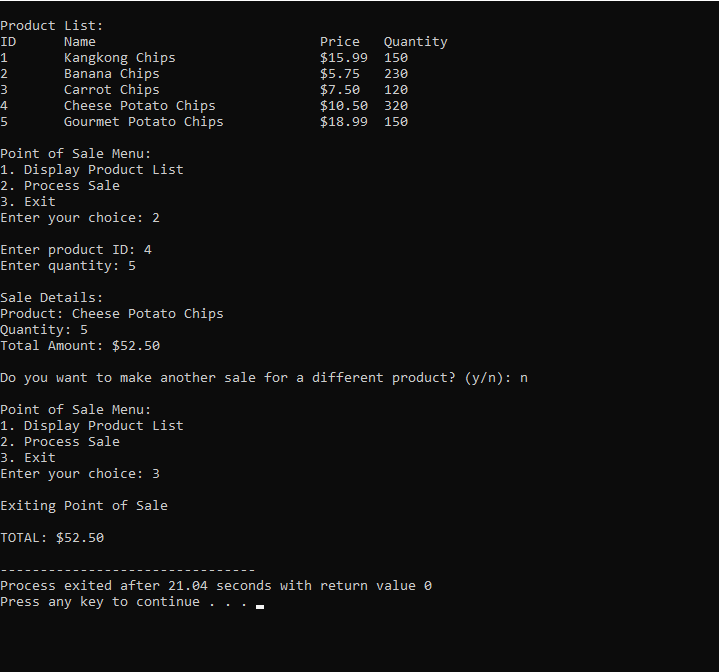
Points of sale (POSs) are an important focus for marketers because consumers tend to make purchasing decisions on high-margin products or services at these strategic locations. Traditionally, businesses set up POSs near store exits to increase the rate of impulse purchases as customers leave. However, varying POS locations can give retailers more opportunities to micro-market specific product categories and influence consumers at earlier points in the sales funnel. A POS or point of sale is a device that is used to process transactions by retail customers. A cash register is a type of POS. The cash register has largely been replaced by electronic POS terminals that can be used to process credit cards and debit cards as well as cash. A POS may be a physical device in a brick-and-mortar store or a checkout point in a web-based store. Electronic POS software systems streamline retail operations by automating the transaction process and tracking important sales data. Basic systems include an electronic cash register and software to coordinate data collected from daily purchases. Retailers can increase functionality by installing a network of data-capture devices, including card readers and barcode scanners. Depending on the software features, retailers can track pricing accuracy, inventory changes, gross revenue, and sales patterns. Using integrated technology to track data helps retailers catch discrepancies in pricing or cash flow that could lead to profit loss or interrupt sales. POS systems that monitor inventory and buying trends can help retailers avoid customer service issues, such as out-of-stock sales, and tailor purchasing and marketing to consumer behavior.

**SCOPE AND LIMITATIONS**

Our code establishes a basic Point of Sale (POS) system for "Crispy Delights Chips." It defines a product structure and includes functionalities such as displaying the product list, processing sales, and generating receipts. However, the program is limited to a predefined maximum of five products (MAX\_PRODUCTS) and has fixed constraints on the maximum length of product names, receipt content, and the overall quantity of products. Additionally, it lacks dynamic memory allocation for product scalability, error handling for invalid inputs during user interactions, and a more comprehensive sales reporting mechanism. While suitable for educational purposes or small-scale scenarios, this POS system may require enhancements for broader practical applications



**SYSTEM REFERENCE:**



**ANALYSIS**

**Input Requirements:**

The program involves user interaction through a console interface for a Point of Sale (POS) system. Users input numeric choices to navigate the menu options, with choices ranging from displaying the product list (option 1) to processing a sale (option 2) or exiting the program (option 3). When processing a sale, users are prompted to enter the product ID and quantity for the desired product. Additionally, the code opens a file named "sales.txt" for sales record-keeping, ensuring users provide valid inputs to avoid errors during file operations. Throughout the program, input is acquired using the `scanf` function, and the overall success of file operations and closures is checked to ensure proper functionality.

**Output Requirements:**

The program includes both console output and file output for sales record-keeping. Console outputs comprise the display of the product list, sale details, and a summary of the total amount. Specifically, the product list is formatted to include the product ID, name, price, and quantity available. In the case of processing a sale, sale details such as the product name, quantity sold, and total amount are printed to the console. Additionally, a receipt summary is generated and appended to the "sales.txt" file, containing information about the product, quantity sold, and total amount. The total sales amount is displayed on the console and also written to the sales file for comprehensive record-keeping. Error messages are printed if there are issues with file opening or if there is insufficient stock for a particular product during a sale transaction.

**DESIGN**

**Files and Their Descriptions:**

Source Code File: contains the main program written in C, incorporating standard libraries (, , ), defining constants (MAX\_PRODUCTS, MAX\_NAME\_LENGTH, MAX\_RECEIPT\_LENGTH), declaring a Product structure for inventory items, and implementing functions for displaying, writing, printing sales information, while the main function initializes an inventory array, facilitates user interaction, processes sales, and records information in a sales file. Sales File (sales.txt): opened in the program with an append mode ("a"), serves as a record-keeping repository for sales information, including a header with the business name, separators, details of each sale (product name, quantity, total amount), and concludes with a summary line containing the overall total sales amount; it is created if it does not exist, and an error message is displayed on the console if there is an issue opening the file.

**User Interface Design:**

* The program starts with a header displaying the business name ("Crispy Delights

Chips")

* It presents a menu-driven interface in the console with three options: Display Product List (Option 1), Process Sale (Option 2), and Exit (Option 3)
* Users input their choice by entering a numeric value corresponding to the menu options
* If Option 1 is selected, the program displays a product list with IDs, names, prices, and quantities.
* If Option 2 is chosen, the user is prompted to enter a product ID and quantity for processing a sale. It provides sale details, updates inventory, and appends sales information to a file named "sales.txt."
* The program handles errors, such as insufficient stock or an invalid product ID, and prints corresponding error messages.
* The loop continues until the user selects ‘n’ to stop buying and press the number 3 to Exit the sale process.
* After exiting, the program prints the overall total sales amount to the console and appends it to the sales file.
* If there is an error opening the sales file, an error message is displayed.

**Features of the Project:**

1. **Product Listing**: Displays a list of available products with IDs, names, prices, and quantities.

2. **Sales Processing**: Allows users to input product ID and quantity to process a sale, updating inventory and generating a sales receipt.

3. **Sales Record Keeping**: Appends sales information, including product details, quantity, and total amount, to a sales file ("sales.txt") for record-keeping.

4. **Error Handling**: Detects and handles errors such as insufficient stock or an invalid product ID, providing corresponding error messages.

5. **User Interface**: Presents a user-friendly menu-driven interface with options to display the product list, process a sale, or exit the program.

6. **Overall Total Calculation**: Calculates and displays the overall total sales amount at the end of the program execution.

7. **Sales File Header**: Appends a header with the business name ("Crispy Delights Chips") and separators to the sales file

8. **File Opening Error Handling**: Checks for errors when opening the sales file and prints an error message if encountered.

**Security and Audit Consideration:**

* **File Input Validation:**

Ensure proper validation and sanitization of user inputs, especially when reading from files or taking user choices to prevent potential security vulnerabilities like buffer overflows or injection attacks.

* **Secure File Operations:**

Implement secure file handling practices, including proper permissions and file access controls. Validate the success of file operations to handle potential errors or exceptions, preventing unauthorized access.

* **Data Encryption:**

Consider implementing data encryption techniques, especially if sensitive information is being stored in files. This can add an extra layer of security, protecting data from unauthorized access.

* **Code Injection Prevention:**

Validate and sanitize any external inputs (e.g., user choices, file content) to prevent code injection attacks. This is crucial for maintaining the integrity of the program and avoiding potential security loopholes.

* **Logging and Auditing:**

Implement logging mechanisms to record critical events, errors, or securityrelated incidents. Regularly review logs to detect unusual activities or potential security breaches.

* **Authentication and Authorization:**

If applicable, consider implementing user authentication and authorization mechanisms to control access to certain functionalities or sensitive data within the program.

* **Error Handling and Reporting**

Provide meaningful error messages to users without revealing sensitive information. Properly handle exceptions and errors to avoid potential security risks and improve the overall robustness of the application.

* **Secure Memory Handling:**

Ensure secure memory handling practices to prevent buffer overflows or memory leaks. Validate and sanitize inputs before processing to avoid potential vulnerabilities.

* **Regular Code Reviews and Audits:**

Conduct regular code reviews and security audits to identify and address potential vulnerabilities. Follow best practices for secure coding to minimize the risk of security issues.

* **Compliance with Privacy Regulations:**

Ensure compliance with relevant privacy regulations and standards, especially if the application handles sensitive user information. This includes considerations for data storage, processing, and access controls.

* **Secure Communication:**

If the application communicates with external services or networks, implement secure communication protocols (e.g., HTTPS) to protect data during transit

* **Update and Patch Management:**

Stay informed about security updates and patches related to the programming language, libraries, and dependencies used in the application. Regularly update the software to address known vulnerabilities.

* **Input Validation and Sanitization:**

Validate and sanitize user inputs thoroughly to prevent input-based attacks. This is essential for maintaining the integrity and security of the application.

* **Database Security:**

If the application uses a database, ensure that it follows secure database practices, such as parameterized queries to prevent SQL injection attacks.

* **Code Obfuscation:**

Consider code obfuscation techniques to make the source code less readable and understandable for potential attackers. This adds an additional layer of protection against reverse engineering

* **Security Training:**

Ensure that developers are trained in secure coding practices and are aware of common security threats and vulnerabilities. This knowledge can significantly contribute to building a more secure application.

**IMPLEMENTATION**

**URL OF THE SAVED SOURCE CODE**

[**https://drive.google.com/file/d/1CMeeJlabN0NR8ArEGu2\_g8G2kjWN7Gcu/view?usp=sharing**](https://drive.google.com/file/d/1CMeeJlabN0NR8ArEGu2_g8G2kjWN7Gcu/view?usp=sharing)

**Software Package Files and Their Descriptive Information:**

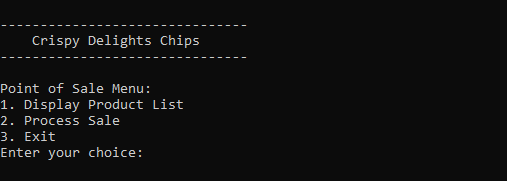
* **main.c**
* Description: This file serves as the main program written in C, acting as the entry point for the application.
* Includes :
* **<stdio.h>**
* **<stdlib.h>**
* **<string.h>**
* Constants
* MAX\_PRODUCTS: Maximum number of products in the inventory (set to 5)
* MAX\_NAME\_LENGTH: Maximum length for a product name (set to 50 characters).
* MAX\_RECEIPT\_LENGTH: Maximum length for a sales receipt (set to 1000 characters).
* Structures:
* struct Product: Represents each item in the inventory with attributes such as name, price, and quantity.
* Functions:
* displayProducts: Displays the list of products with their details..
* writeReceipt: Writes sales receipt details to a string.
* printReceipt: Prints the sales receipt details to the console.
* processSale: Processes a sale, updates inventory, and records sales information.
* main: The main function where the program execution begins. Initializes products, interacts with users through a menu-driven interface, and writes sales information to a file.
* **sales.txt**
* Description: This file is used for recording sales information. It is named "sales.txt" and is opened in the program for appending ("a" mode). If it doesn't exist, it is created.
* Contents:
* Bullet 1: Header with the business name ("Crispy Delights Chips") and separators.
* Bullet 2: For each sale, it appends a section with details such as the product name, quantity sold, and total amount.
* Bullet 3: The file ends with a summary line containing the overall total sales amount.

**Function Declarations and their Descriptive Purposes:**

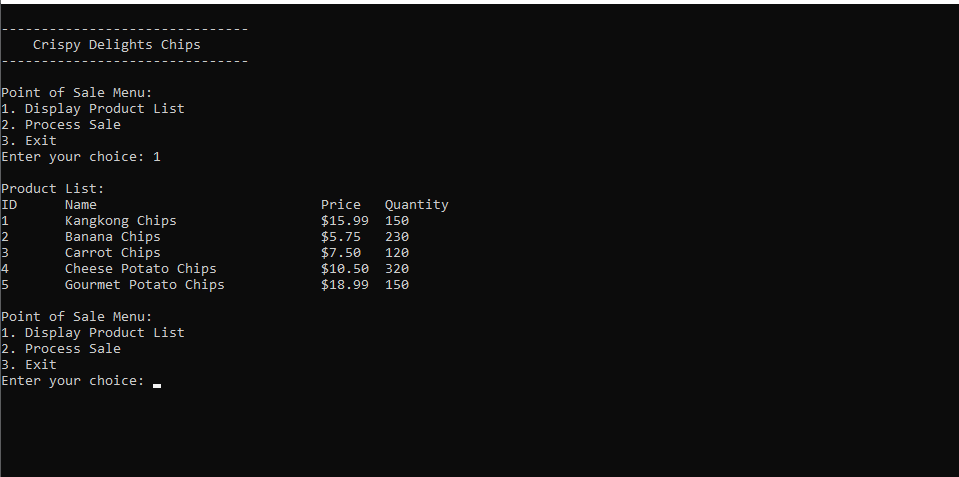
1. **void displayProducts(struct Product products[], int numProducts):** Displays the list of products with their details.
2. **void writeReceipt(char receipt[MAX\_RECEIPT\_LENGTH], struct Product product, int quantity, float total):** Writes sales receipt details to a string.
3. **void printReceipt(struct Product product, int quantity, float total):** Prints the sales receipt details to the console.
4. **void processSale(struct Product products[], int productId, int quantity, FILE \*salesFile, float \*overallTotal):** Processes a sale, updates inventory, and records sales information.
5. **int main():** The main function where the program execution begins. Initializes products, interacts with users through a menu-driven interface, and writes sales information to a file.

**TESTING AND DEBUGGING**

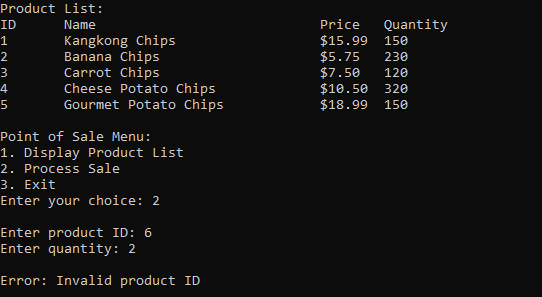
In running the source code, You can see The Point of Sale Menu and you can enter your own choice.



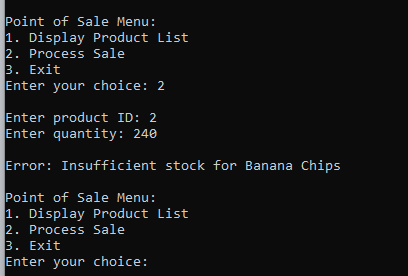
Let’s try choosing number 1. It display the Product List of our Item, Where you can see the name of the product , price and quantity.



To Buy a Product you need to choose number 2. As you can see in the product list you can see the ID number of the Product you can only enter 1-5 if u enter number above 5 the system will say error. “Invalid Product ID”.



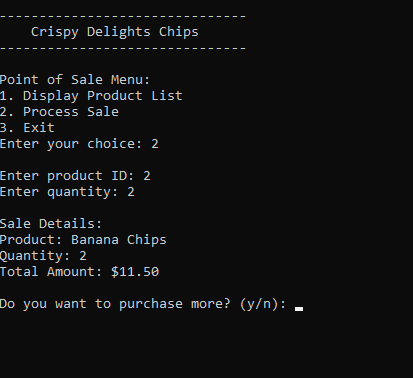
Also when the product quantity is out of stock the system will be an error. “Insufficient stock for Product”

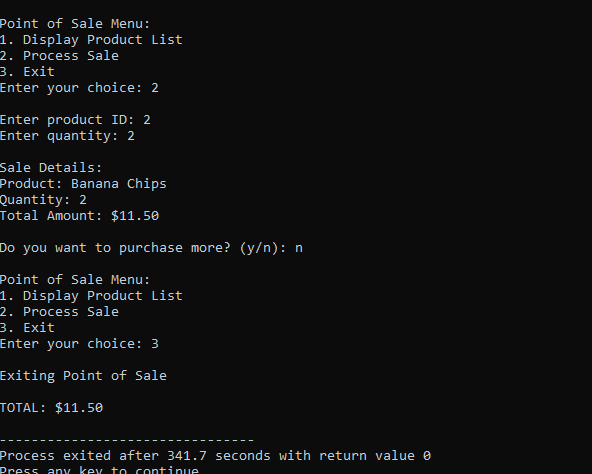


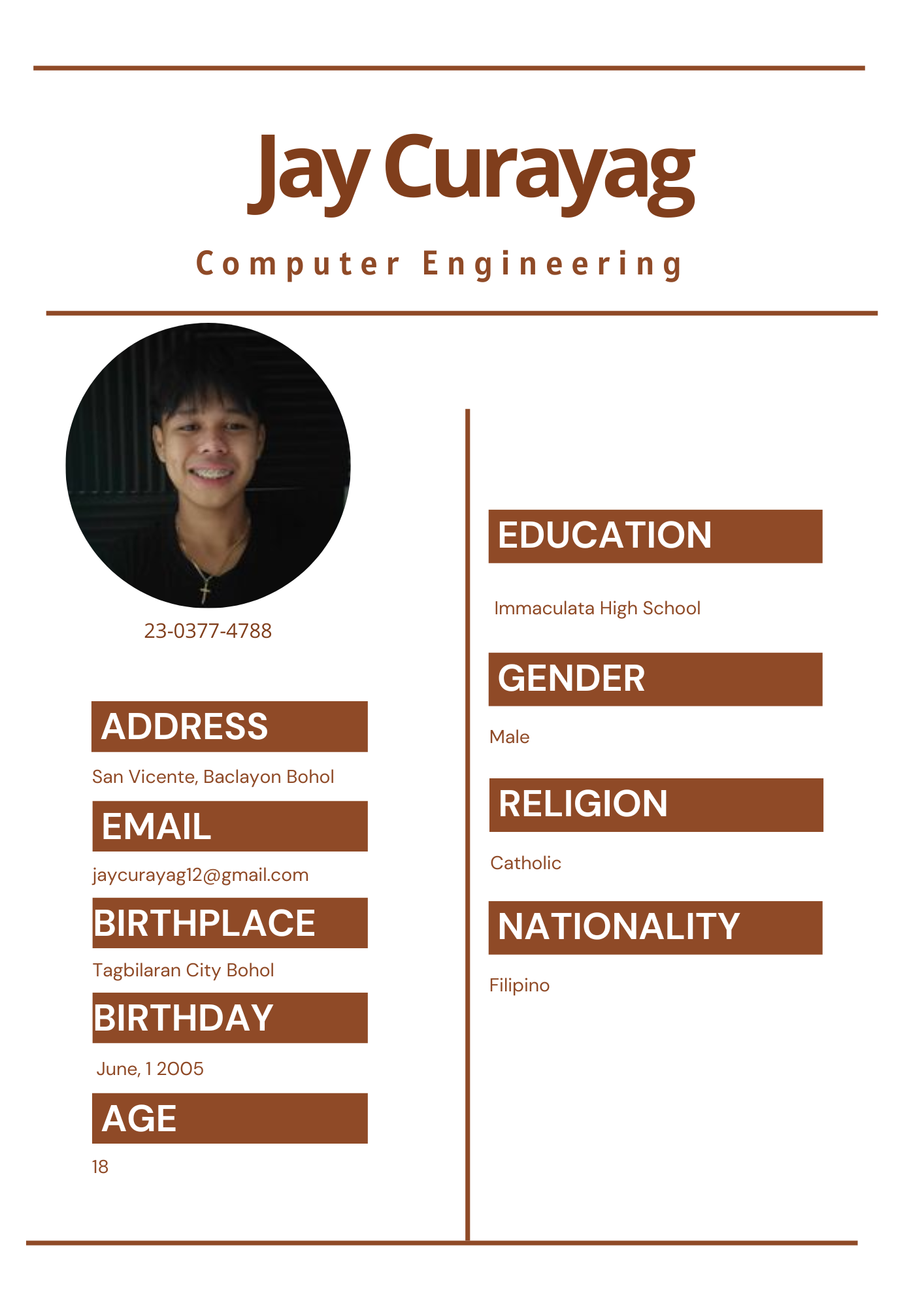
.

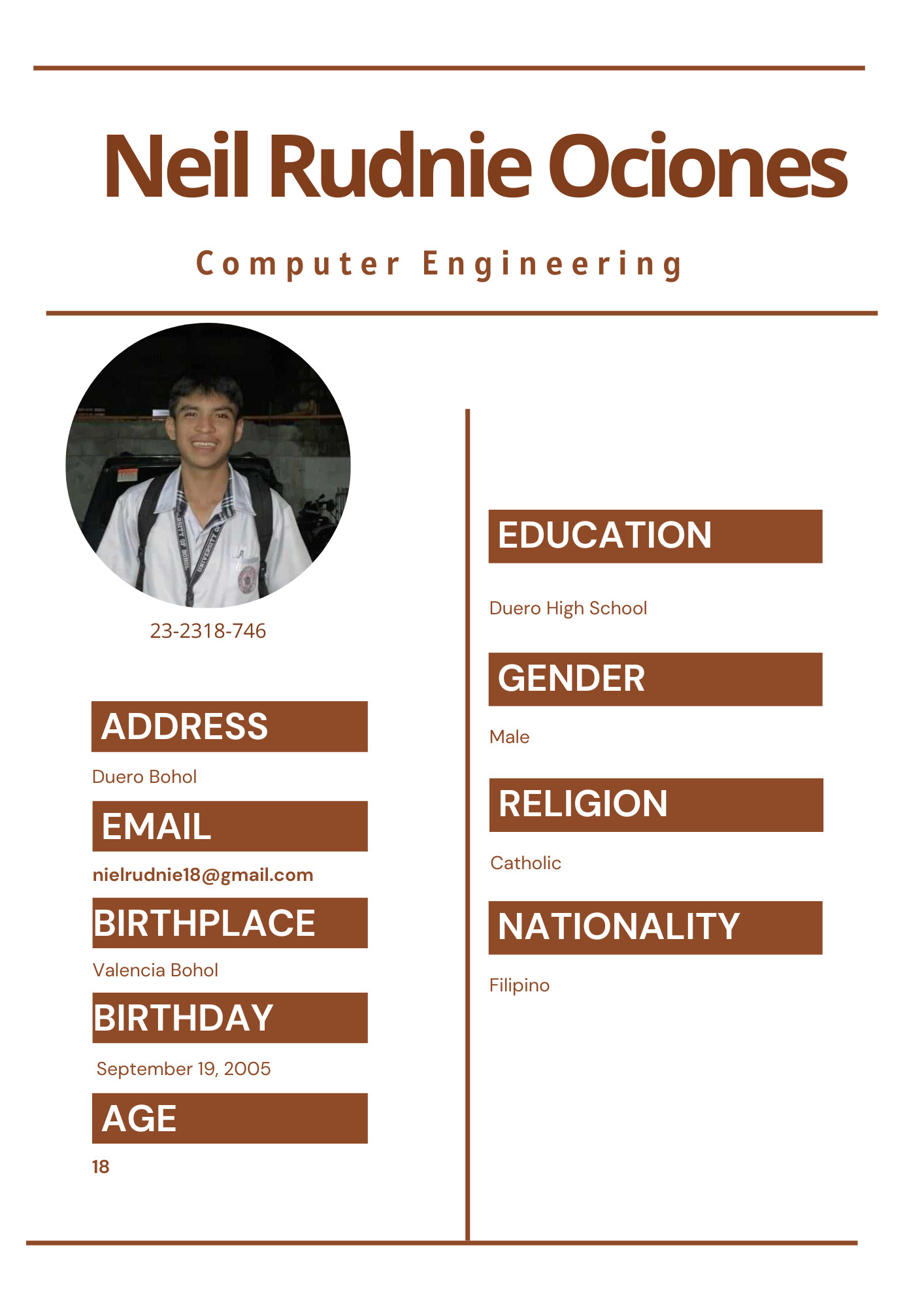
So if a customer is buying product you can just enter the number 2 and enter the product ID and also the quantity of the customer’s product to buy. And so after Entering the product ID and quantity you can see the Sale Details The product name , quantity and the total amount of the product that he/she bought. Also if you want to make another purchase you can just press ‘y’

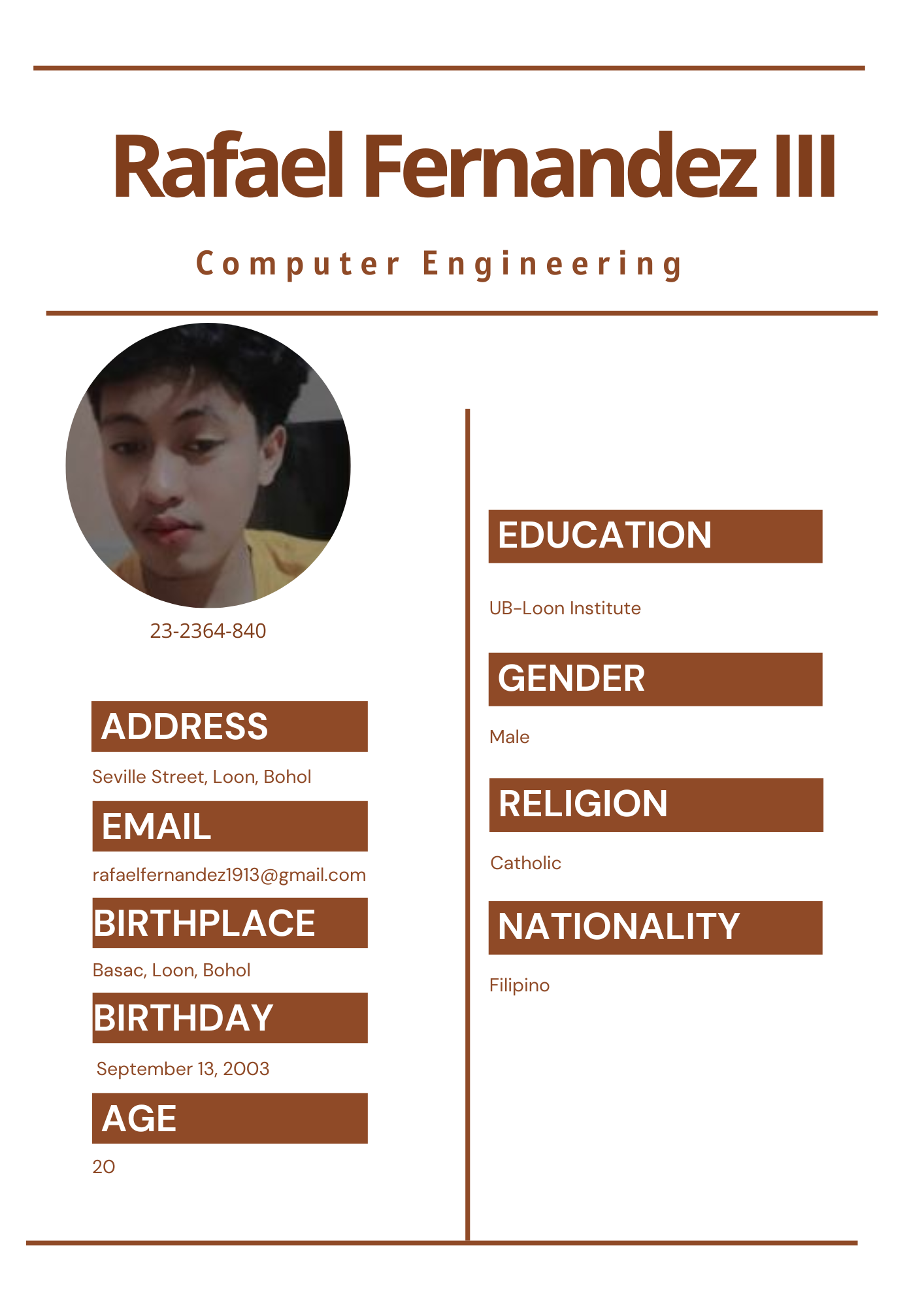
Or ‘Y’ and to stop purchasing you can just press ‘n’.



And if you stop buying you will go to the Sale Menu and to Exit you just need to enter 3.









**Documentation**

****

**FUTURE DEVELOPMENT**

1. **User Authentication and Authorization:** Implement user accounts with different roles (cashier, manager) and password protection to ensure secure access to the POS system.
2. **Transaction History:** Maintain a comprehensive transaction history that includes details such as timestamps, products sold, quantities, and prices. This history can be useful for auditing and analysis.
3. **Inventory Alerts:** Implement alerts for low stock levels or products nearing expiration. This can help in restocking decisions and preventing the sale of expired items.

**Glossary**

**A Point of Sale (POS):** system is a combination of hardware and software that allows businesses to process transactions with customers. It is commonly used in retail stores, restaurants, and various other types of businesses where goods or services are sold to customers.

**<stdio.h>:** Standard Input/Output library.

**<stdlib.h>:** Standard Library for general-purpose functions.

**<string.h>:** String Library for string manipulation.

**MAX\_PRODUCTS:** Maximum number of handled products.

**MAX\_NAME\_LENGTH:** Maximum length of a product name.

**MAX\_RECEIPT\_LENGTH:** Maximum length of a sales receipt.

**struct Product:** Structure defining product attributes.

**displayProducts:** Displays the product list.

**writeReceipt:** Writes a sales receipt.

**processSale:** Handles the sales process.

**FILE \*salesFile:** File pointer for sales file.