**ASSIGNMENT 8 SUBMISSION**

**Student: Dat Hoang Vien**

**Course: COMSC-165: Advanced Programming With C & C++**

**Part 1:**

* Source Code:

/\*

Student: Dat Hoang Vien

Subject: COMSC-165: Advanced Programming With C & C++

Assignment: 8

Part: 1

Date: 07/05/2025

Description: a program that reads .txt files containing normal text content and

            allows user to decrypt or encrypt the files. Here's a detailed

            description of the operations that the user can perform:

            1. File encryption

            - This option will ask the user to enter the name of the input and

            output file. If the user enter blank names or the files cannot be

            opened, an error message will be displayed. Otherwise, the content of

            the file will be encryted.

            - Encryption format: each character is shifted by a repeating

            sequence of values: +10, +20, +30, +10, +20, +30, ... and the pattern

            restarts at the beginning of each new line.

            2. File decryption

            - This option will also ask the user to enter the name of the input

            and output file. If the user enter blank names or the files cannot be

            opened, an error message will be displayed. Otherwise, the content of

            the file will be decryted.

            - Decryption format: each character is shifted by the reverse order

            of the encryption sequence: -10, -20, -30, -10, -20, -30, ... It also

            restarts at the beginning of each new line.

            3. Display file content

            - This option will ask the user to enter the name of the file whose

            content they want to display. If the user enter a blank name or the

            file cannot be opened, an error message will be displayed. Otherwise

            the content of the file will be displayed.

            4. Display the encrypted content

            - This option will display the content of the file that is most

            recently encrypted. If none of the file has been encrypted (meaning

            option 1 has never been selected), it will not display anything. If

            the file cannot be opened, an error message will be displayed.

            Otherwise, the content of most recently encrypted file will be

            displayed.

            5. Display the decrypted content

            - This option will display the content of the file that is most

            recently decrypted. If none of the file has been decrypted (meaning

            option 2 has never been selected), it will not display anything. If

            the file cannot be opened, an error message will be displayed.

            Otherwise, the content of most recently decrypted file will be

            displayed.

            6. Exit

            - This option ends the program.

\*/

#include <iostream>

#include <string>

#include <fstream>

using namespace std;

// Function Prototype

bool fileEncryption(const string fileInput, const string fileOutput, string& errorMessage);

void stringEncryption(string&);

bool fileDecryption(const string fileInput, const string fileOutput, string& errorMessage);

void stringDecryption(string&);

bool displayContent(const string fileName, string& errorMessage);

int main()

{

    // Declare variables

    int choice = 0;                    // User's choice

    bool loop = true;                  // Loop the program

    string fileToEncrypt = "";         // Name of the encryption file

    string fileToDecrypt = "";         // Name of the decryption file

    string currentEncryptedFile = "";  // Name of the encrypted file

    string currentDecryptedFile = "";  // Name of the decrypted file

    string errorMessage = "";          // Error message

    // Ask user to enter choice

    while (loop)

    {

        // Ask the user to pick an option

        cout << "Choose from these options: " << endl;

        cout << "1. File encryption" << endl;

        cout << "2. File decryption" << endl;

        cout << "3. Display file content" << endl;

        cout << "4. Display the encrypted content" << endl;

        cout << "5. Display the decrypted content" << endl;

        cout << "6. Exit!" << endl;

        cout << "Your option: ";

        cin >> choice;

        cin.ignore(1000, 10);

        // Check whether the option is valid

        while (choice < 1 || choice > 6)

        {

            // Ask the user to enter a valid option

            cout << "Invalid! Please enter an option from 1 - 5: ";

            cin >> choice;

            cin.ignore(1000, 10);

        }

        // Switch-cases

        switch (choice)

        {

            // 1. File encryption

            case 1:

            {

                // Let user enter the name of the file to encrypt

                cout << " - Enter the name of the file you want to encrypt: ";

                getline(cin, fileToEncrypt);

                // Let user enter the name of the output file,

                // whose content needs decryption

                cout << " - Enter the name of the output file: ";

                getline(cin, fileToDecrypt);

                // Print a blank line

                cout << endl;

                // Encrypt the file and check whether it was successful

                if (fileEncryption(fileToEncrypt, fileToDecrypt, errorMessage))

                {

                    cout << " --- File encryption successful! --- \n" << endl;

                    currentEncryptedFile = fileToDecrypt;

                }

                else

                {

                    cout << "Error: " << errorMessage << "\n" << endl;

                }

                // Reset the variables

                errorMessage = "";

                fileToEncrypt = "";

                fileToDecrypt = "";

                // Break out of the switch-case

                break;

            }

            // 2. File decryption

            case 2:

            {

                // Let user enter the name of the file to decrypt

                cout << " - Enter the name of the file you want to decrypt: ";

                getline(cin, fileToDecrypt);

                // Let user enter the name of the output file,

                // whose content needs encryption

                cout << " - Enter the name of the output file: ";

                getline(cin, fileToEncrypt);

                // Print a blank line

                cout << endl;

                // Decrypt the file and check whether it was successful

                if (fileDecryption(fileToDecrypt, fileToEncrypt, errorMessage))

                {

                    cout << " --- File decryption successful! --- \n" << endl;

                    currentDecryptedFile = fileToEncrypt;

                }

                else

                {

                    cout << "Error: " << errorMessage << "\n" << endl;

                }

                // Reset the variables

                errorMessage = "";

                fileToEncrypt = "";

                fileToDecrypt = "";

                // Break out of the switch-case

                break;

            }

            // Display the file's content

            case 3:

            {

                // Store file's name

                string fileName = "";

                // Let user enter the name of the file to display

                cout << " - Enter the name of the file you want to display: ";

                getline(cin, fileName);

                // Print a header to console

                cout << " --- File content --- " << endl;

                cout << endl;

                // Display the encrypted content

                if(!displayContent(fileName, errorMessage))

                {

                    cout << "Error: " << errorMessage << endl;

                }

                // Print a footer to console

                cout << endl;

                cout << " --- File content --- " << endl;

                cout << endl;

                // Reset the variables

                errorMessage = "";

                // Break out of the switch-case

                break;

            }

            // 4. Display the encrypted content

            case 4:

            {

                // Print a header to console

                cout << " --- Encrypted content --- " << endl;

                cout << endl;

                // Display the encrypted content

                if(!displayContent(currentEncryptedFile, errorMessage))

                {

                    cout << "Error: " << errorMessage << endl;

                }

                // Print a footer to console

                cout << endl;

                cout << " --- Encrypted content --- " << endl;

                cout << endl;

                // Reset the variables

                errorMessage = "";

                // Break out of the switch-case

                break;

            }

            // 5. Display the decrypted content

            case 5:

            {

                // Print a header to console

                cout << " --- Decrypted content --- " << endl;

                cout << endl;

                // Display the decrypted content

                if(!displayContent(currentDecryptedFile, errorMessage))

                {

                    cout << "Error: " << errorMessage << endl;

                }

                // Print a footer to console

                cout << endl;

                cout << " --- Decrypted content --- " << endl;

                cout << endl;

                // Reset the variables

                errorMessage = "";

                // Break out of the switch-case

                break;

            }

            // 6. Exit

            case 6:

            {

                // Print an exit message and set the looping boolean to false

                cout << "Thank you for using this program!" << endl;

                loop = false;

                break;

            }

        }

    }

    return 0;

}

// Define function

void stringEncryption(string& str)

{

    // Variables

    int shifts[] = {10, 20, 30};  // The format to encrypt the characters

    // Loop and encrypt the string

    for(int i = 0; i < str.length(); i++)

    {

        str.at(i) = str.at(i) + shifts[i % 3];

    }

}

bool fileEncryption(const string fileInput, const string fileOutput, string& errorMessage)

{

    // Check whether the input file name is empty

    if (fileInput.empty())

    {

        errorMessage = "Input file name is empty!";

        return false;

    }

    // Check whether the output file name is empty

    if (fileOutput.empty())

    {

        errorMessage = "Output file name is empty!";

    }

    // Create 2 file stream objects and open the files

    ifstream fin;

    fin.open(fileInput, ios::in);

    ofstream fout;

    fout.open(fileOutput, ios::out);

    // Check whether we can open the input file

    if (!fin)

    {

        errorMessage = "Could not open " + fileInput + "!";

        return false;

    }

    // Check whether we can open the output file

    if (!fout)

    {

        errorMessage = "Could not open " + fileOutput + "!";

        return false;

    }

    // Variables

    string line = "";  // Store the lines of the file

    // Read the content of the file

    while (getline(fin, line))

    {

        // Encrypt the string and paste it into the output file

        stringEncryption(line);

        fout << line << endl;

    }

    // Close and clear the fie stream objects

    fin.close();

    fin.clear();

    fout.close();

    fout.clear();

    return true;

}

void stringDecryption(string& str)

{

    // Variables

    int shifts[] = {-10, -20, -30};  // The format to decrypt the characters

    // Loop and encrypt the string

    for(int i = 0; i < str.length(); i++)

    {

        str.at(i) = str.at(i) + shifts[i % 3];

    }

}

bool fileDecryption(const string fileInput, const string fileOutput, string& errorMessage)

{

    // Check whether the input file name is empty

    if (fileInput.empty())

    {

        errorMessage = "Input file name is empty!";

        return false;

    }

    // Check whether the output file name is empty

    if (fileOutput.empty())

    {

        errorMessage = "Output file name is empty!";

        return false;

    }

    // Create 2 file stream objects and open the files

    ifstream fin;

    fin.open(fileInput, ios::in);

    ofstream fout;

    fout.open(fileOutput, ios::out);

    // Check whether we can open the input file

    if (!fin)

    {

        errorMessage = "Could not open " + fileInput + "!";

        return false;

    }

    // Check whether we can open the output file

    if (!fout)

    {

        errorMessage = "Could not open " + fileOutput + "!";

        return false;

    }

    // Variables

    string line = "";  // Store the lines of the file

    // Read the content of the file

    while (getline(fin, line))

    {

        // Encrypt the string and paste it into the output file

        stringDecryption(line);

        fout << line << endl;

    }

    // Close and clear the fie stream objects

    fin.close();

    fin.clear();

    fout.close();

    fout.clear();

    return true;

}

bool displayContent(const string fileName, string& errorMessage)

{

    // Check whether file name is empty

    if (fileName.empty())

    {

        errorMessage = "No content to display!";

        return false;

    }

    // Create a file stream object to read the files

    ifstream fin;

    fin.open(fileName, ios::in);

    // Check whether we can open the file

    if (!fin)

    {

        errorMessage = "Could not open " + fileName + "!";

        return false;

    }

    // Variables

    string line = "";  // Store the lines of the file

    // Print all the lines to the console

    while (getline(fin, line))

    {

        // Paste the data to the console

        cout << line << endl;

    }

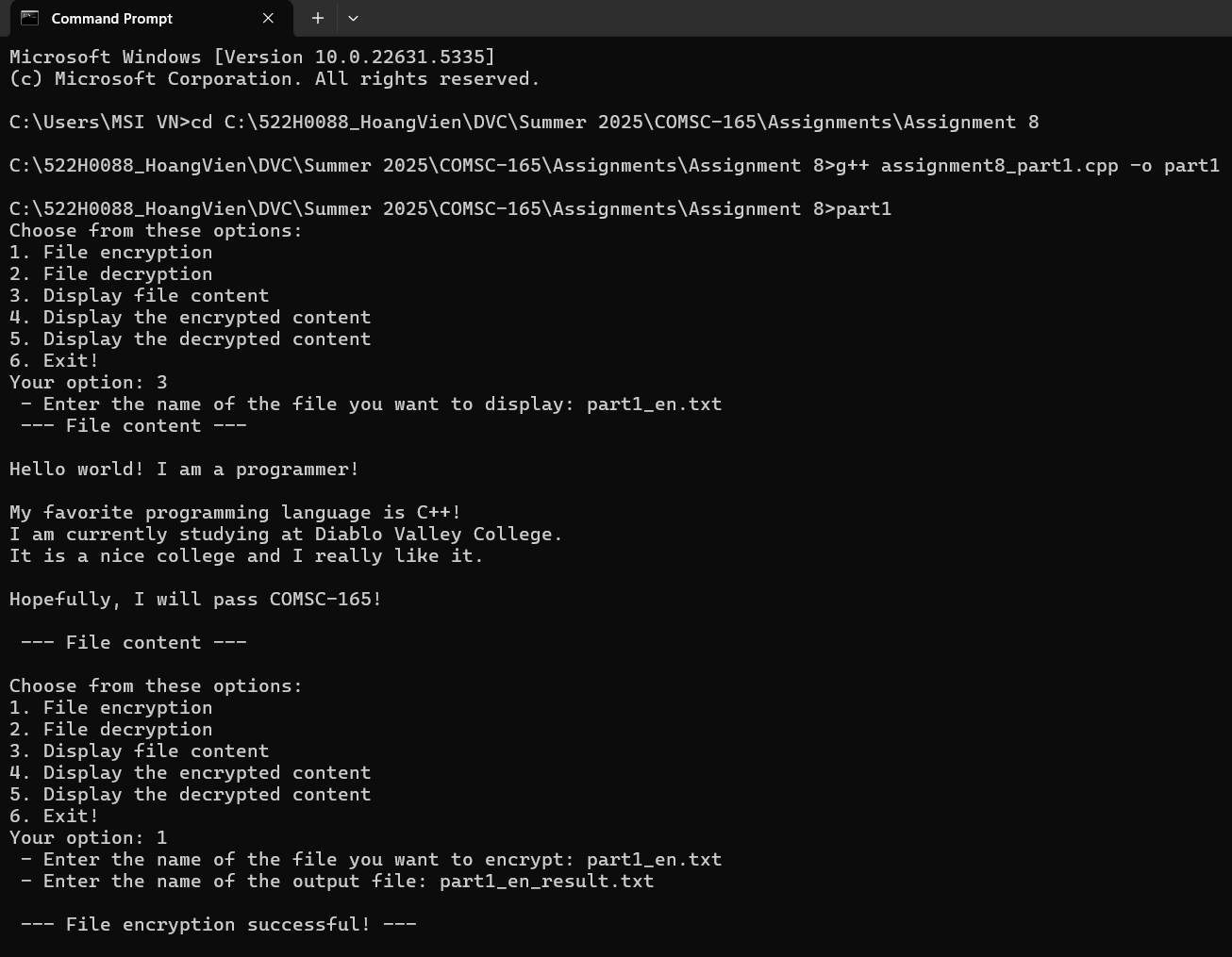
    // Close and clear the fie stream object

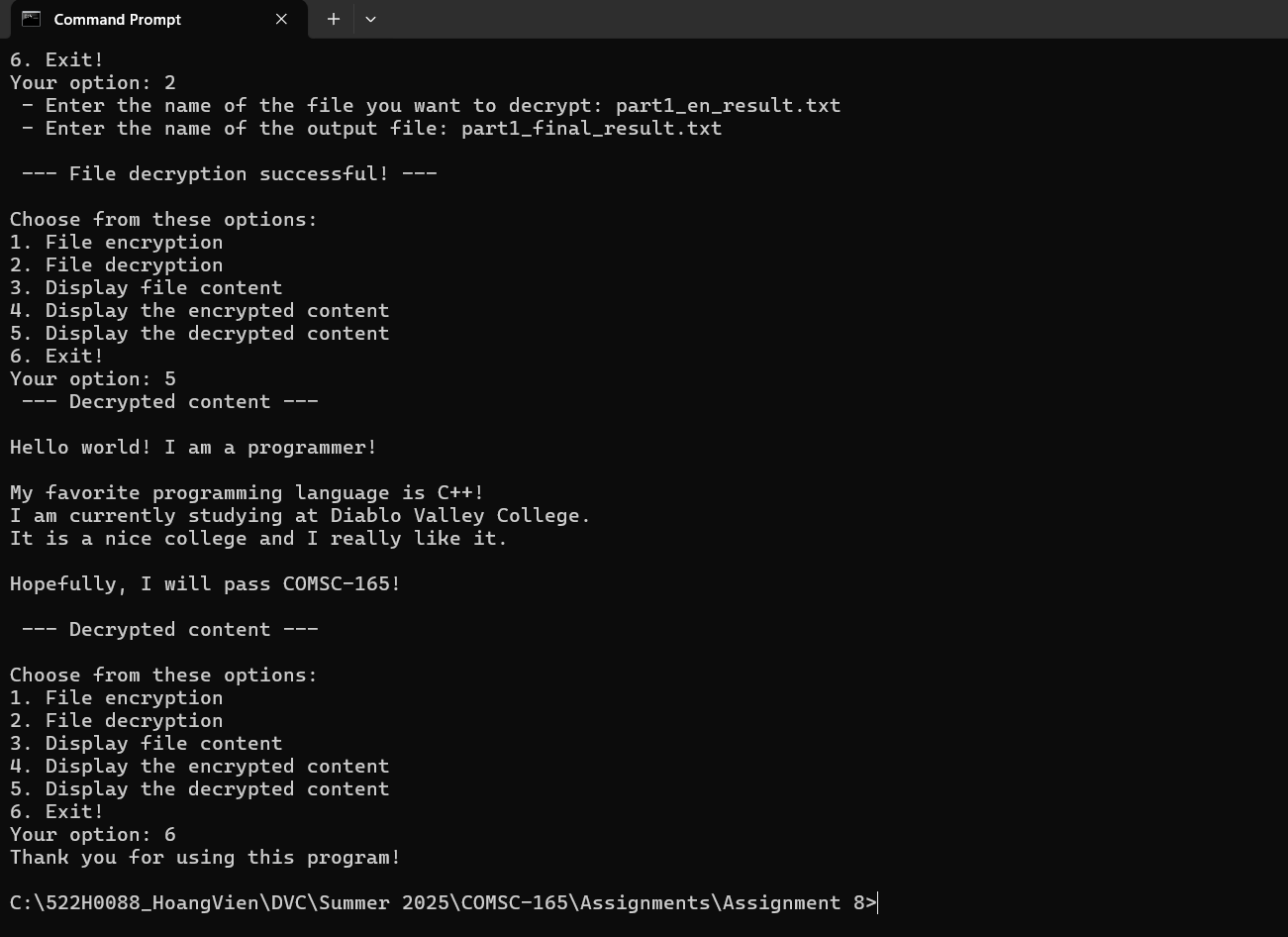
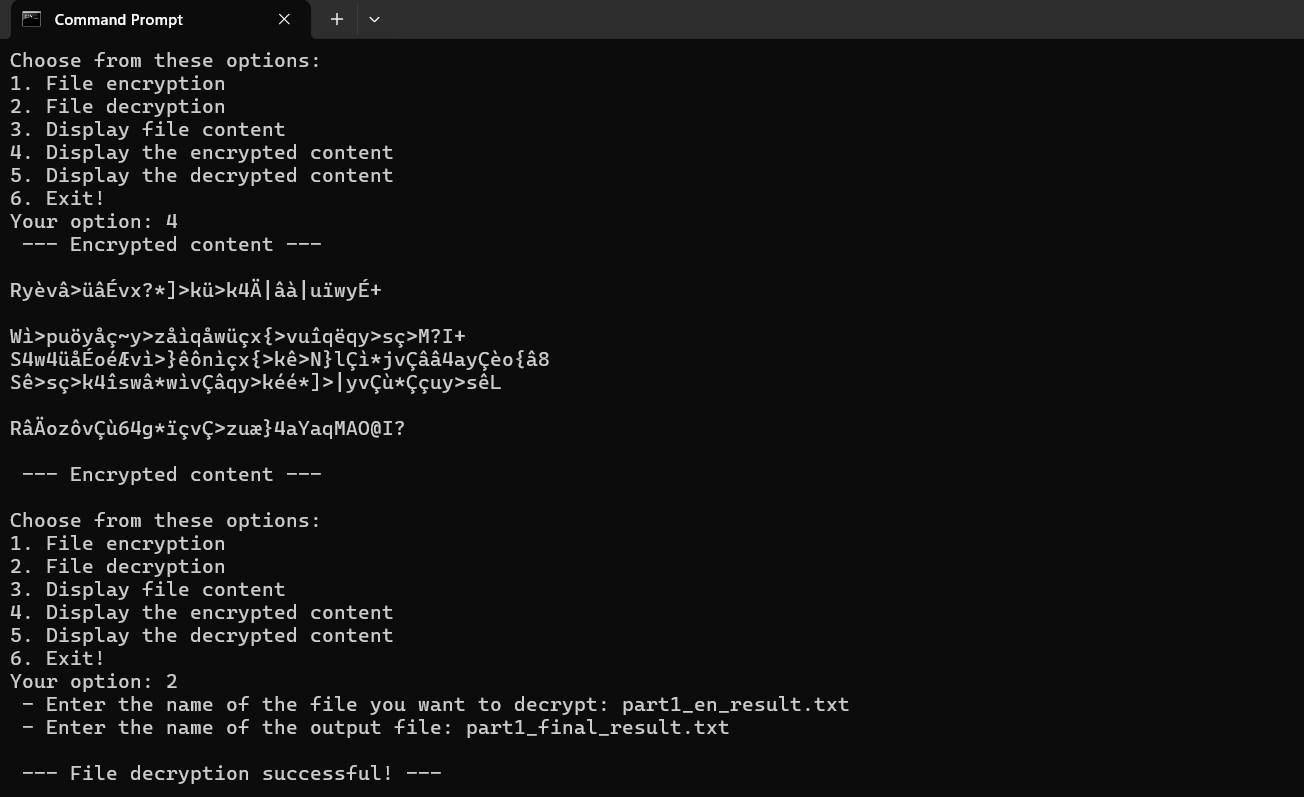
    fin.close();

    fin.clear();

    return true;

}

* Output:



**Part 2:**

* Source Code:

/\*

Student: Dat Hoang Vien

Subject: COMSC-165: Advanced Programming With C & C++

Assignment: 8

Part: 2

Date: 07/05/2025

Description: a program that reads .txt files containing information on items

            such as their name, quantity on hand, wholesale cost, and retail

            cost. The program will only store new items and check whether the

            item has existed based on the item name. Here's a detailed

            description of the operations:

            1. Input

            - The program allows user to input via console or file. If the user

            choose to input the item's information via the console input option,

            they will be required to enter the item name, quantity on hand,

            wholesale cost, and retail cost one by one. If the user wishes to use

a file to add the data, they will be required to enter the file's name.

            2. Search and display the items

            - The user will be asked to enter the name of the item to search for

            the matching item name(s) inside the inventory. The function will

            display all the matching item names starting from the first letter.

            3. Change record

            - This option will prompt the user to enter the item name whose

            information the user wishes to change. The user will be required to

            enter the new quantity on hand, wholesale cost, and the retail cost.

            4. Display all records

            - The user can select this option to display all the current items in

            the inventory.

            5. Display inventory report

            - This option will display a report on the total number of items, the

            total wholesale cost, and the total retail cost of the inventory.

            6. Reset inventory

            - If selected, this option will allow the user to reset the whole

            inventory. Upon selecting, the user will be asked to confirm the

            choice once more time before the inventory is reset. This option

            cannot be undone once completed.

            7. Exit

            - This option ends the program.

\*/

#include <iostream>

#include <fstream>

#include <iomanip>

#include <cstring>

using namespace std;

// Constants

const int SIZE = 101;

const string inventory = "inventory.dat";

// Structures

struct Item  // Item structure

{

    char itemName[SIZE];  // Item name

    int quantity;         // Quantity on hand

    double wholeSale;     // Wholesale cost

    double retail;        // Retail

};

// Function Prototype

bool consoleInput();

bool fileInput();

bool isValidInput(const char[], const int, const double, const double);

bool isNewItem(Item);

bool changeRecord(char[]);

bool searchItems(char[]);

void displayAllRecords();

void displayReport();

void resetInventory();

int main()

{

    // Declare variables

    bool loop = true;  // Determine when the loop ends

    int choice = -1;   // The user's choice

    // Ask user to enter choice

    while (loop)

    {

        // Display option for user

        cout << "Choose an option: " << endl;

        cout << "(1 - Add new records, 2 - Search and display records, 3 - Change record, 4 - Display all records, 5 - Display inventory report, 6 - Reset the inventory, 7 - Exit the program)" << endl;

        cout << "Your option: ";

        cin >> choice;

        cin.ignore(1000, 10);

        while (choice < 1 || choice > 7)

        {

            cout << "Invalid option! Please choose from the menu above: ";

            cin >> choice;

            cin.ignore(1000, 10);

        }

        // Switch-cases

        switch(choice)

        {

            // Input data

            case 1:

            {

                int inputChoice = 0;  // User's input choice

                bool validInput = false;  // Check whether the operation(s) failed

                // Allow user to input data through 2 options: file and console

                cout << " - Choose an input option: " << endl;

                cout << " - (1 - Console input, 2 - File input)" << endl;

                cout << " - Your option: ";

                cin >> inputChoice;

                cin.ignore(1000, 10);

                // Check whether the option is valid or not

                while (inputChoice != 1 && inputChoice != 2)

                {

                    cout << " - Invalid! Please choose a valid option: ";

                    cin >> inputChoice;

                    cin.ignore(1000, 10);

                }

                if (inputChoice == 1)  // If user choose console input

                {

                    validInput = consoleInput();

                    // Check whether the operation was successful

                    if (validInput)

                    {

                        cout << " ---- Item(s) data input successful! ---- " << endl;

                    }

                    else

                    {

                        cout << " ---- Item(s) data input unsuccessful! ---- " << endl;

                    }

                }

                else                   // If user choose file inputinput

                {

                    validInput = fileInput();

                    // Check whether the operation was successful

                    if (validInput)

                    {

                        cout << " ---- Item(s) data input successful! ---- " << endl;

                    }

                    else

                    {

                        cout << " ---- Item(s) data input unsuccessful! ---- " << endl;

                    }

                }

                // Break from the switch case

                break;

            }

            // Search and display records

            case 2:

            {

                // Declare a cstring variable

                char pattern[SIZE];

                // Ask user to enter the name of the item

                cout << "Enter the name of the item: ";

                cin.getline(pattern, SIZE);

                // Call function to display the record based on the pattern

                searchItems(pattern);

                // Break from the switch case

                break;

            }

            // Change record

            case 3:

            {

                // Declare a cstring variable

                char pattern[SIZE];

                // Ask user to enter the name of the item

                cout << "Enter the name of the item: ";

                cin.getline(pattern, SIZE);

                // Call function to change the record based on the pattern

                changeRecord(pattern);

                // Break from the switch case

                break;

            }

            // Display all records

            case 4:

            {

                // Display all the records

                displayAllRecords();

                // Break from the switch case

                break;

            }

            // Display inventory report

            case 5:

            {

                // Display the inventory report

                displayReport();

                // Break out of the switch case

                break;

            }

            // Reset inventory

            case 6:

            {

                // Calls function to reset inventory

                resetInventory();

                // Break out of the switch case

                break;

            }

            // Exit the program

            case 7:

            {

                // Display thank you message

                cout << "Thank you for using this program!" << endl;

                // End the loop

                loop = false;

                // Break out of the switch case

                break;

            }

        }

    }

    return 0;

}

// Define function

bool consoleInput()

{

    // Declare a temporary structure

    Item it;

    // Ask user to enter fields

    cout << "    + Enter item's name: ";

    cin.getline(it.itemName, SIZE);

    cout << "    + Enter item's quantity on hand: ";

    cin >> it.quantity;

    cin.ignore(1000, 10);

    cout << "    + Enter item's wholesale value: ";

    cin >> it.wholeSale;

    cin.ignore(1000, 10);

    cout << "    + Enter item's retail value: ";

    cin >> it.retail;

    cin.ignore(1000, 10);

    // Declare an stream output object

    ofstream fout;

    fout.open(inventory, ios::out | ios::app | ios::binary);

    // Check if the fields are valid and whether this is a new item

    if (isValidInput(it.itemName, it.quantity, it.wholeSale, it.retail)

     && isNewItem(it))

    {

        // If the data is valid, we add the item data to the output file

        fout.write((char\*)&it, sizeof(Item));

        // Increment the number of items

        cout << " ---- Item Data Successfully Saved! ---- " << endl;

        return true;

    }

    else

    {

        // If the data is not valid

        cout << " ---- Error! Invalid Data Or Item Has Existed! ---- " << endl;

        return false;

    }

}

bool fileInput()

{

    // Declare variables

    Item it;               // A temporary Item object

    int itemsAdded = 0;    // An item counter

    string fileName = "";  // Input file name

    // Ask user to enter file name

    cout << " - Please enter file name: ";

    getline(cin, fileName);

    // Declare file stream objects

    ifstream fin;

    fin.open(fileName, ios::in);

    ofstream fout;

    fout.open(inventory, ios::out | ios::app | ios::binary);

    // Check whether the stream object fails

    if (fin.fail())

    {

        cout << " - Error! Failed to open file!" << endl;

        return false;

    }

    else

    {

        // Read the data on the line

        while (fin >> it.itemName >> it.quantity >> it.wholeSale >> it.retail)

        {

            // Check if the values are valid and whether the item is new

            if (isValidInput(it.itemName, it.quantity, it.wholeSale, it.retail) && isNewItem(it))

            {

                // Add the item to the inventory file

                fout.write((char\*)&it, sizeof(Item));

                // Increment the number of items added

                itemsAdded++;

            }

        }

    }

    // Check whether at least 1 item was added

    if (itemsAdded > 0)

    {

        return true;

    }

    return false;

}

bool isValidInput(const char itemName[SIZE], const int quantity, const double wholeSale, const double retail)

{

    return strlen(itemName) > 0 && quantity >= 0 && wholeSale >= 0 && retail >= 0;

}

bool isNewItem(Item it)

{

    // Declare file stream object

    ifstream fin;

    fin.open(inventory, ios::in | ios::binary);

    // Declare variables

    Item temp;  // A temporary item

    // Read through the file and save the item

    while (fin.read((char\*) &temp, sizeof(Item)))

    {

        // Compare the names of the items

        if (!strcmp(temp.itemName, it.itemName))

        {

            // Return false, the item is not new

            return false;

        }

    }

    return true;  // This is a new item

}

bool searchItems(char pattern[])

{

    // Declare file stream object

    ifstream fin;

    fin.open(inventory, ios::in | ios::binary);

    // Declare variables

    int lenPattern = strlen(pattern);  // Length of the pattern

    Item it;    // A temporary structure object

    int n = 0;  // Count the number of matching results

    // Display a header

    cout << " --- Search results --- " << endl;

    // Display the header

    cout << setw(25) << "Item Name" << " | ";

    cout << setw(10) << "Quantity" << " | ";

    cout << setw(10) << "Wholesale" << " | ";

    cout << setw(10) << "Retail" << endl;

    cout << setw(55) << "-----------------------------------------------------------------";

    cout << endl;

    // Read the file and compare the names

    while(fin.read((char\*) &it, sizeof(Item)))

    {

        if (!strncmp(it.itemName, pattern, lenPattern))

        {

            // Display the record

            cout << setw(25) << it.itemName << " | ";

            cout << setw(10) << it.quantity << " | ";

            cout << setw(10) << it.wholeSale << " | ";

            cout << setw(10) << it.retail << endl;

            // Increment the count

            n++;

        }

    }

    // If there is no result, notify the user

    if (n == 0)

    {

        cout << " --- (No Result Found!) --- " << endl;

        return false;

    }

    return true;

}

bool changeRecord(char pattern[])

{

    // Declare file stream object

    fstream finFout;

    finFout.open(inventory, ios::in | ios::out | ios::binary);

    // Declare variables

    Item it;        // Temporary structure

    long pos = -1;  // Position of the matching pattern

    // Read through the file

    while (finFout.read((char\*) &it, sizeof(Item)))

    {

        // Compare and see whether the names matches

        if(!strcmp(it.itemName, pattern))

        {

            // Save the matching position

            pos = finFout.tellg() - (long long) sizeof(Item);

            // Break from the while loop and stop reading

            break;

        }

    }

    // Check whether there is a match

    if (pos == -1)

    {

        return false;

    }

    // If there is a match, we allow the user to modify the whole structure

    cout << "Please enter the new quantity on hand: ";

    cin >> it.quantity;

    cout << "Please enter the new wholesale cost: ";

    cin >> it.wholeSale;

    cout << "Please enter the new retail cost: ";

    cin >> it.retail;

    // Go to the saved position and rewrite the data

    finFout.seekp(pos, ios::beg);

    finFout.write((char\*) &it, sizeof(Item));

    return true;

}

void displayAllRecords()

{

    // Declare file stream object

    ifstream fin;

    fin.open(inventory, ios::in | ios::binary);

    // Declare a temporary structure

    Item it;

    // Display the header

    cout << setw(25) << "Item Name" << " | ";

    cout << setw(10) << "Quantity" << " | ";

    cout << setw(10) << "Wholesale" << " | ";

    cout << setw(10) << "Retail" << endl;

    cout << setw(55) << "-----------------------------------------------------------------";

    cout << endl;

    // Read the data from the file

    while (fin.read((char\*) &it, sizeof(Item)))

    {

        // Check if the values are valid

        if (isValidInput(it.itemName, it.quantity, it.wholeSale, it.retail))

        {

            // Display the record

            cout << setw(25) << it.itemName << " | ";

            cout << setw(10) << it.quantity << " | ";

            cout << setw(10) << it.wholeSale << " | ";

            cout << setw(10) << it.retail << endl;

        }

    }

    // Clear and close file stream object

    fin.clear();

    fin.close();

}

void displayReport()

{

    // Declare file stream object

    ifstream fin;

    fin.open(inventory, ios::in | ios::binary);

    // Declare variables

    double totalWholeSale = 0;  // Total wholesale value of the inventory

    double totalRetail = 0;     // Total retail value of the inventory

    int totalQuantity = 0;      // Total quantity of all values in the inventory

    Item it;                    // Temporary structure

    // Read the data of the line

    while(fin.read((char\*) &it, sizeof(Item)))

    {

        // Check if the values are valid

        if (isValidInput(it.itemName, it.quantity, it.wholeSale, it.retail))

        {

            // Update the data

            totalWholeSale += it.wholeSale \* (double) it.quantity;

            totalRetail += it.retail \* (double) it.quantity;

            totalQuantity += it.quantity;

        }

    }

    // Display the report

    cout << "The total wholesale value of the inventory is: ";

    cout << totalWholeSale << endl;

    cout << "The total retail value of the inventory is: ";

    cout << totalRetail << endl;

    cout << "The total number of items of the inventory is: ";

    cout << totalQuantity << endl;

    // Clear and close file stream object

    fin.clear();

    fin.close();

}

void resetInventory()

{

    // Declare variable

    char c;  // Store user's choice

    // Confirm inventory reset

    cout << "Are you sure you want to reset the inventory? (Y/N): ";

    cin >> c;

    cin.ignore(1000, 10);

    if (toupper(c) == 'Y')

    {

        // Declare file stream object

        ofstream fout;

        fout.open(inventory, ios::out | ios::trunc | ios::binary);

        // Check whether the file is truncated (reset)

        if (fout)

        {

            cout << " --- Inventory file reset successfully! --- " << endl;

        }

        else

        {

            cout << " --- Failed to reset inventory file! --- " << endl;

        }

    }

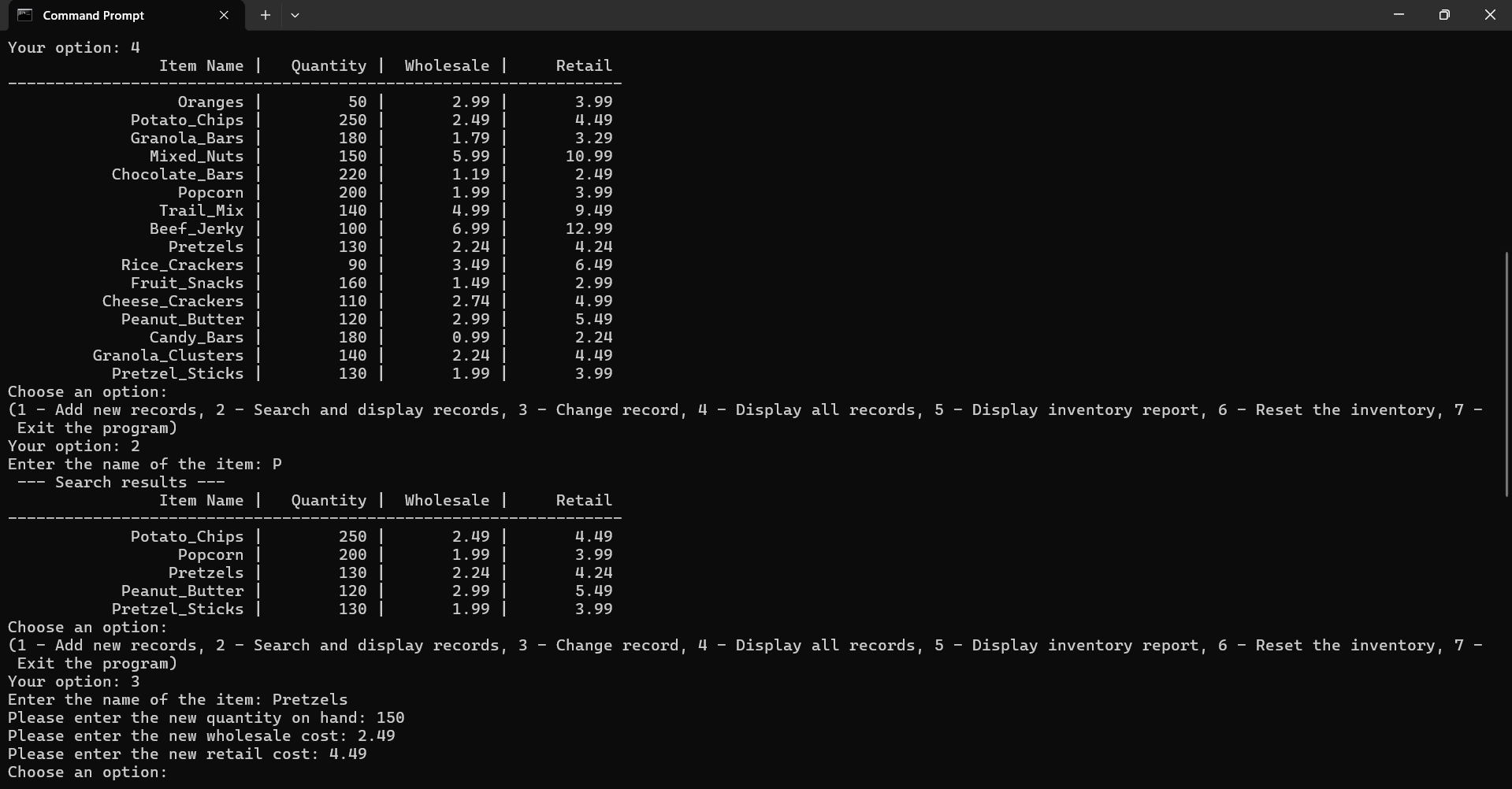
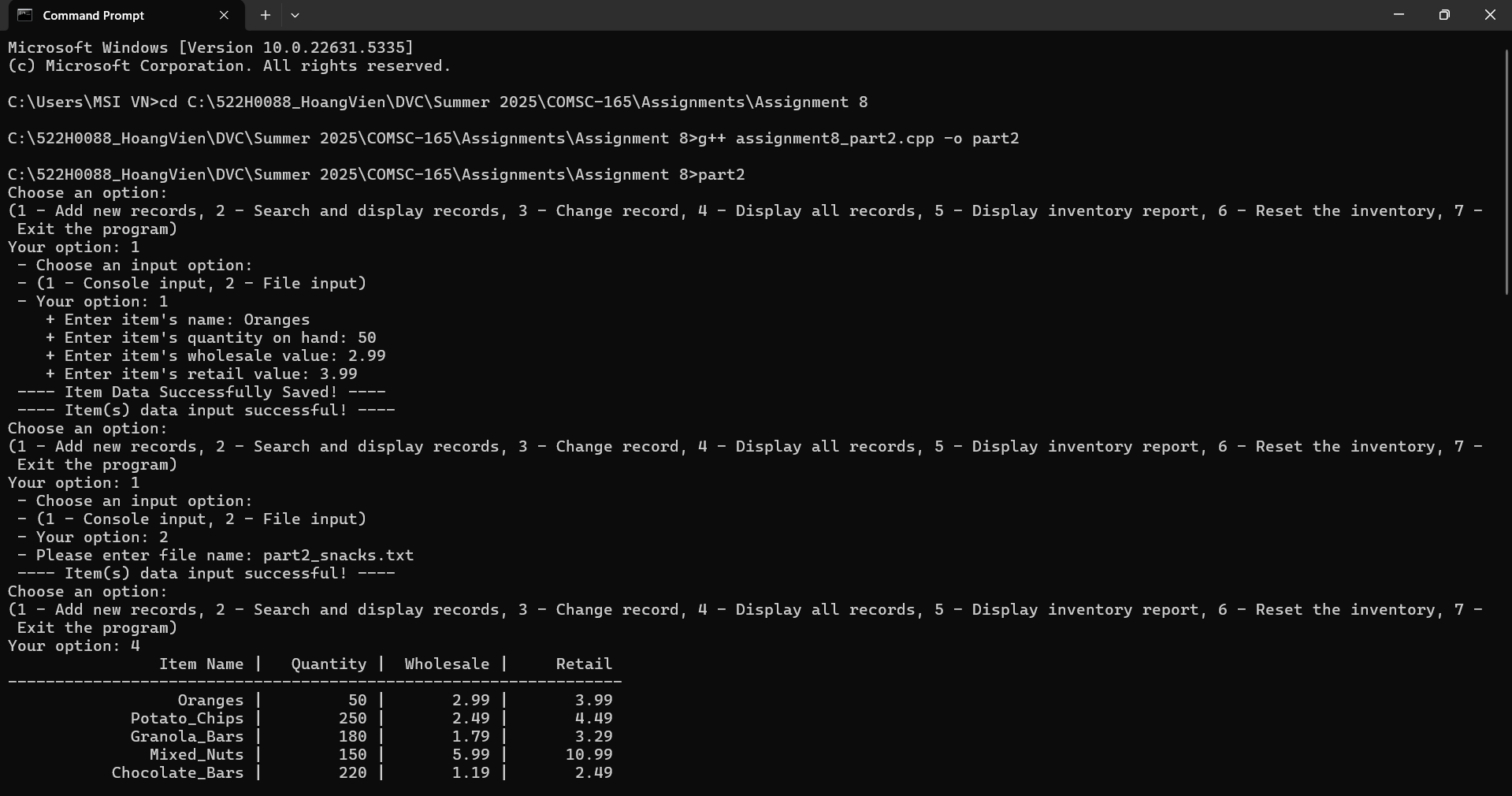
    else

    {

        cout << " --- Inventory not reset! --- " << endl;

    }

}

* Output:

