BRITISH UNIVERSITY COLLEGE

COS 212 – Object Oriented Programming with C++ Assignment

Name	:	Chaw Thiri Win

Table of Contents

System Overview

This file program, Member is designed to efficiently manage a list of members within an organization. It provides features for adding new members, displaying member information, and modifying the added members. The member class represents a single member of the organization which encapsulates member attributes such as id, name, member type and credit points. By encapsulating important properties in a Member class, the Member Management System effectively manages member data, enabling additions, updates, and display of member details.

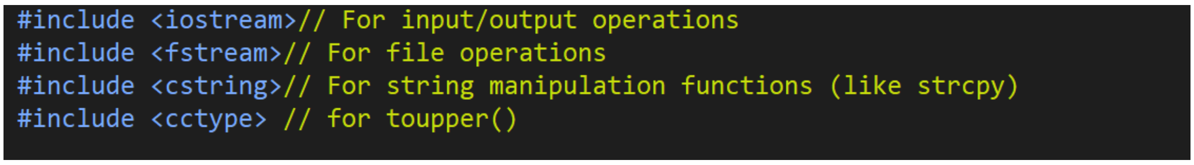
I/O File Manipulation

An open file is represented within a program by a stream. Each of the open member functions of the classes ofstream, ifstream, and fstream has default mode. These files are designed to store text and all values that are input or output. File streams in C++ are essential for interacting with files whether we are writing data to them or reading data from it. They provide a convenient and efficient way to handle file operations, allowing for both input and output capabilities through classes like ifstream, ofstream, and fstream. These classes offer various member functions and operators for reading, writing, and manipulating file data, making file handling in C++ straightforward and versatile.

FileI/O:

File I/O is the essential process of transferring data between a program and a file. It involves opening, reading, writing, and closing files using streams. Text files store human-readable characters, while binary files store data in a more efficient, raw format.

The Header Files



```
#include <iostream>// For input/output operations
#include <fstream>// For file operations
#include <cstring>// For string manipulation functions (like strcpy)
#include <cctype> // for toupper()
```

The code contains the header files required to run a C++ application.

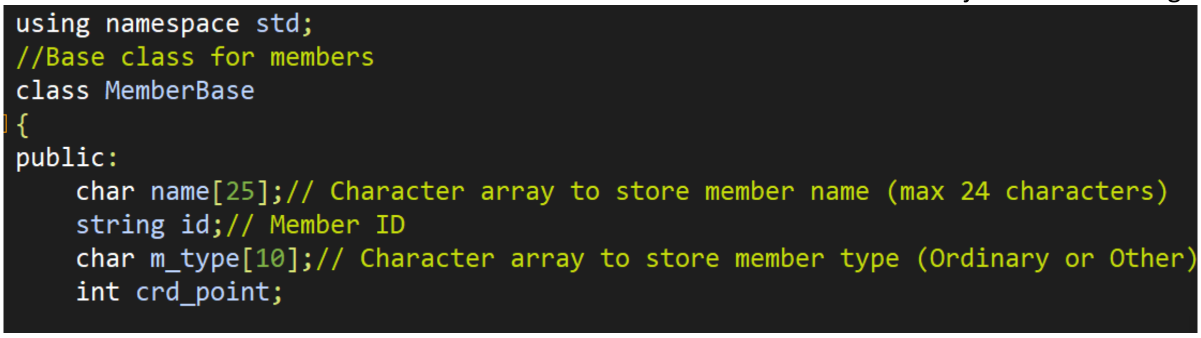
- Input/output operations (such as cin and cout) are provided by iostream.
- File management (reading and writing to files) is made possible via fstream.
- Cstring provides methods for manipulating strings, such as strcpy.
- Character categorization functions (such as toupper) are included in cctype. The application may carry out a number of functions linked to input, output, file management, string manipulation, and character operations thanks to these headers.

Classes:

Classes are user-defined data types that may be utilized by constructing an instance of the class. Classes keep track of their own members' functions and data. Functions, behavior information, and attributes of members are declared and specified within a class.

Member Base:

A basic template for building objects that represent members in a system is the MemberBase class, which gives derived member classes a common set of properties and methods.



```
using namespace std;
//Base class for members
class MemberBase
{
public:
    char name[25];// Character array to store member name (max 24 characters)
    string id;// Member ID
    char m_type[10];// Character array to store member type (Ordinary or Other)
    int crd_point;
```

Figure: Memberbase Class and it's Data Types

Function in Class Member

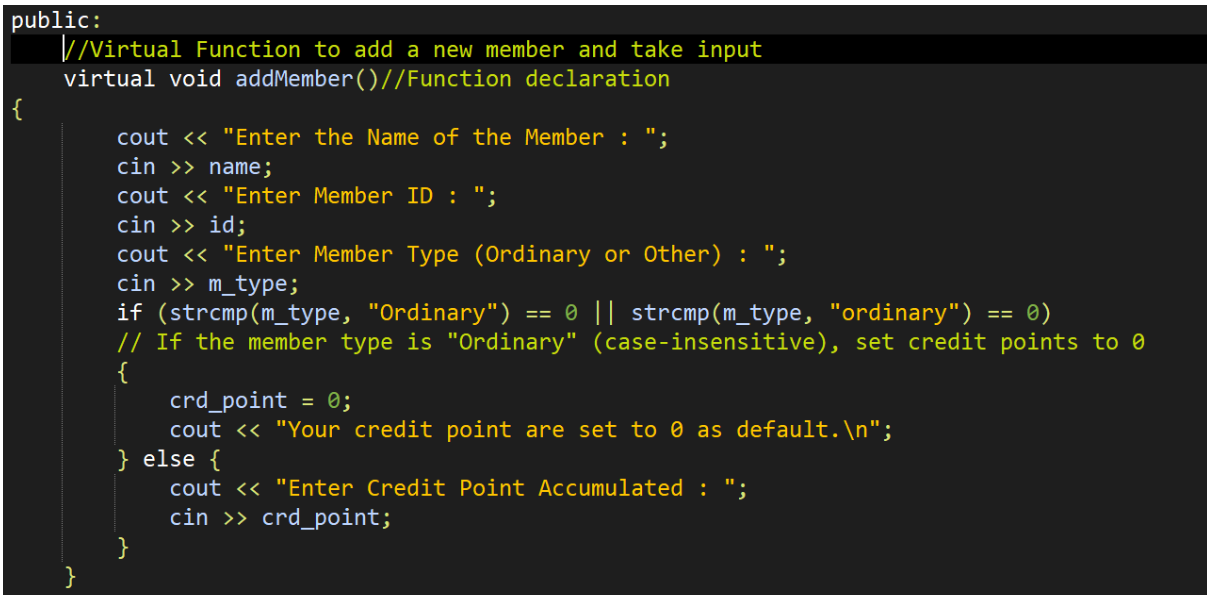
A member function is a function defined within a class, providing operations on the class's data. Functions act as the actions or behaviors an object can perform.

In this member file, there are four main functions in base class. They are:

1. addmember()
2. displaymember()
3. modifymember()

I. virtual void addmember()

The mention virtual function within a class is designed to add a new member to the system by taking input from user for member name, ID, and type. The credit point is automatically set to 0 if the member type is "Ordinary" (case-insensitive). If not, the user is asked to input the amount of the credit point. Polymorphic behavior is possible since this method can be overridden in derived classes, as indicated by the use of the virtual keyword.

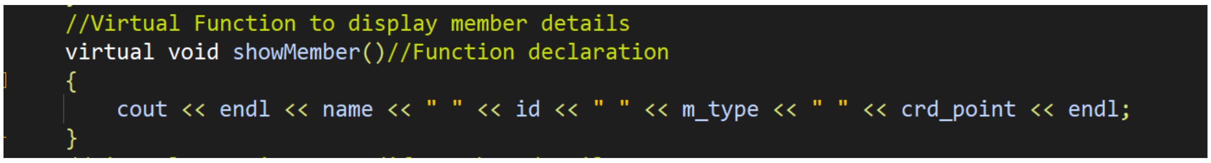


```
public:
    //Virtual Function to add a new member and take input
    virtual void addMember()//Function declaration
{
        cout << "Enter the Name of the Member : ";
        cin >> name;
        cout << "Enter Member ID : ";
        cin >> id;
        cout << "Enter Member Type (Ordinary or Other) : ";
        cin >> m_type;
        if (strcmp(m_type, "Ordinary") == 0 || strcmp(m_type, "ordinary") == 0)
        // If the member type is "Ordinary" (case-insensitive), set credit points to 0
        {
            crd_point = 0;
            cout << "Your credit point are set to 0 as default.\n";
        } else {
            cout << "Enter Credit Point Accumulated : ";
            cin >> crd_point;
        }
    }
```

Figure: Virtual void addMember()

II. virtual void showMember()

Within the class, the supplied code provides a different virtual method called showMember(). This feature is intended to show a member's ID, type, name, and credit points, among other data. Here, virtual is used to guarantee that derived classes may be able to override this function in order to show polymorphism and offer more precise or in-depth member information.

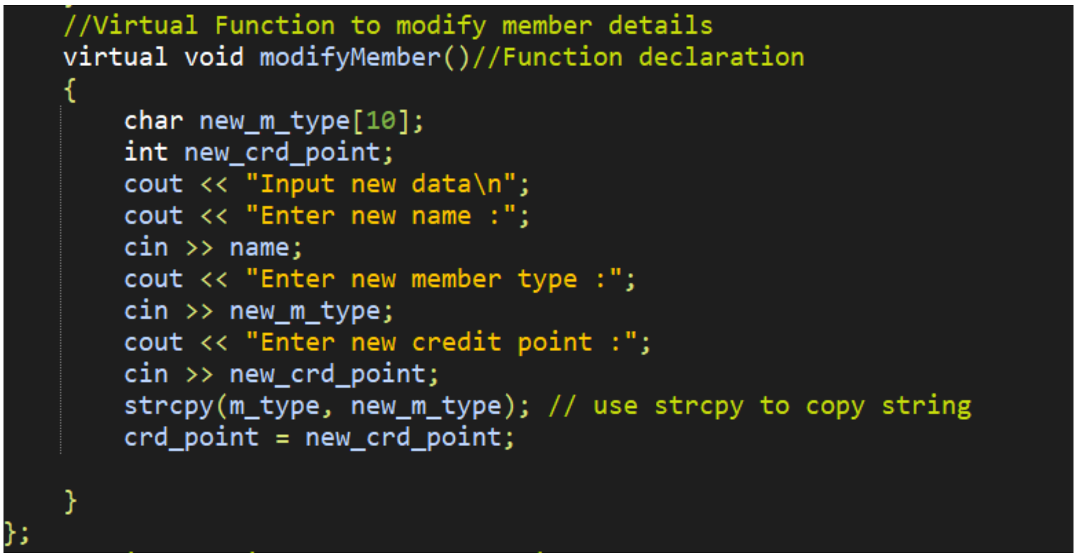


```
    //Virtual Function to display member details
    virtual void showMember()//Function declaration
    {
        cout << endl << name << " " << id << " " << m_type << " " << crd_point << endl;
    }
```

Figure: Virtual void showMember()

III. virtual void modifyMember()

The code provides a virtual method called modifyMember() that lets you change the information about a member. It requires input for credit points, member type, and new ID. The member's data is subsequently updated when these new values are applied to the appropriate member variables. The usage of virtual suggests that this function may be overridden in derived classes to give particular modification behaviors.

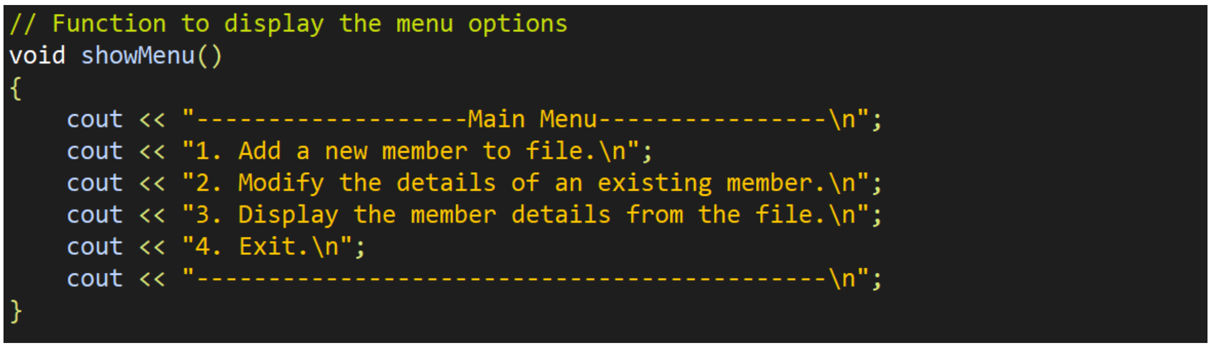


```
    //Virtual Function to modify member details
    virtual void modifyMember()//Function declaration
    {
        char new_m_type[10];
        int new_crd_point;
        cout << "Input new data\n";
        cout << "Enter new name :";
        cin >> name;
        cout << "Enter new member type :";
        cin >> new_m_type;
        cout << "Enter new credit point :";
        cin >> new_crd_point;
        strcpy(m_type, new_m_type); // use strcpy to copy string
        crd_point = new_crd_point;

    }
};
```

Figure: Virtual void modifyMember()

void showMenu()

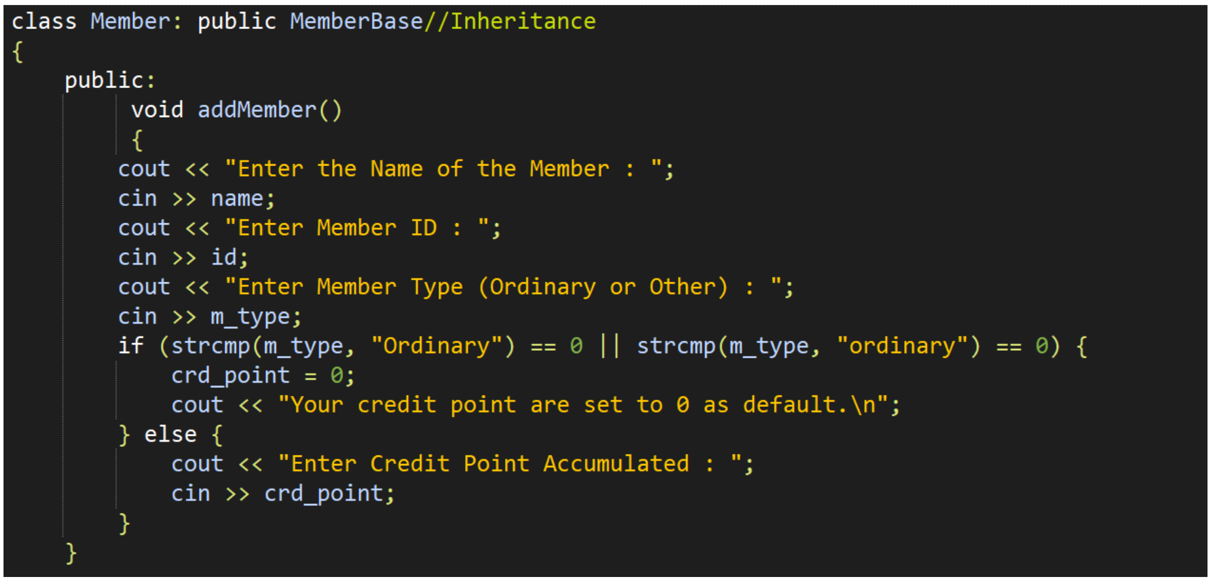
An options menu is shown to the user using the showMenu() method. The possible options are outlined, making the interface simple and easy to use: adding a new member, changing existing member data, showing member information, and quitting the software. This feature is essential for directing how the user interacts with the system.
The MAIN MENU has the following four functions:
1. To update the file with a new member
2. To alter the current member
3. To show every newly recruited and current member
4. To conclude the procedure



```
// Function to display the menu options
void showMenu()
{
    cout << "------------------Main Menu---------------\n";
    cout << "1. Add a new member to file.\n";
    cout << "2. Modify the details of an existing member.\n";
    cout << "3. Display the member details from the file.\n";
    cout << "4. Exit.\n";
    cout << "------------------------------------------\n";
}
```

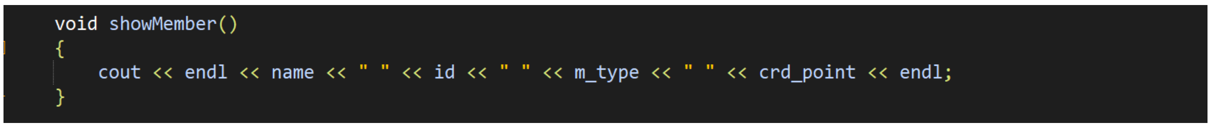
Figure: void showMenu()

A class Member that inherits from a MemberBase class which is defined by the code given below. This behavior void functions and data members are included in this class. It has void member functions that update current member data, show member details, and add new members. Because of the above function will work for only the declaration function. This at member function method at members ID carded points and member type to this file. Member data can be updated by using the modified member function and it can be displayed with the show member function.



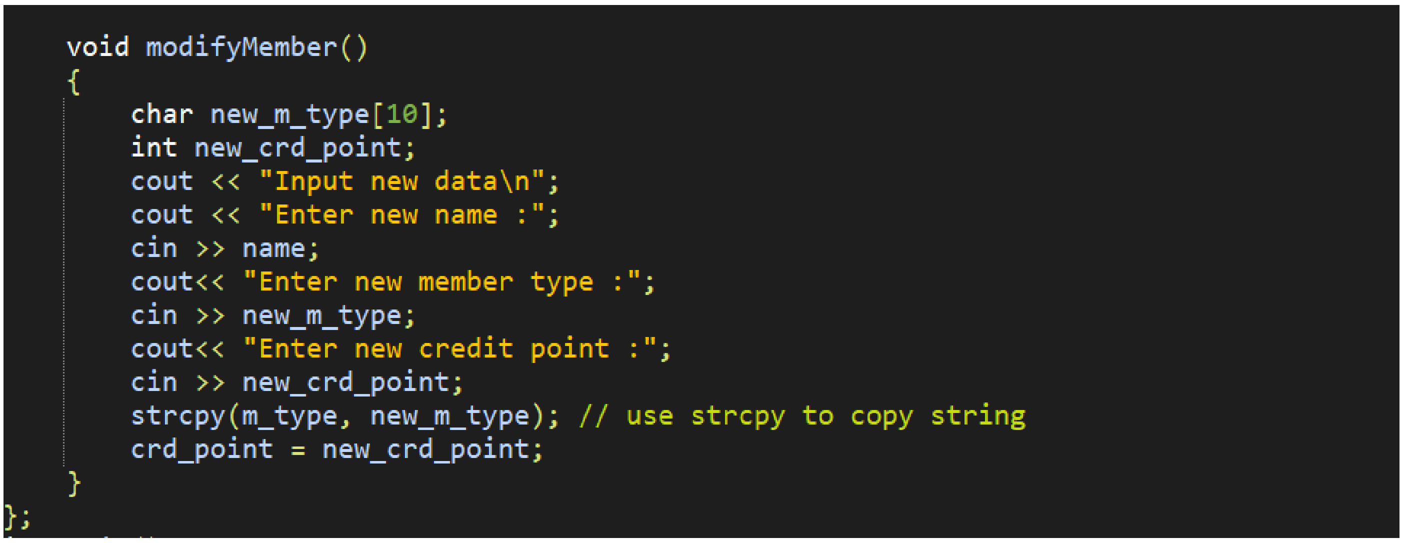
```
class Member: public MemberBase//Inheritance
{
    public:
         void addMember()
         {
        cout << "Enter the Name of the Member : ";
        cin >> name;
        cout << "Enter Member ID : ";
        cin >> id;
        cout << "Enter Member Type (Ordinary or Other) : ";
        cin >> m_type;
        if (strcmp(m_type, "Ordinary") == 0 || strcmp(m_type, "ordinary") == 0) {
            crd_point = 0;
            cout << "Your credit point are set to 0 as default.\n";
        } else {
            cout << "Enter Credit Point Accumulated : ";
            cin >> crd_point;
        }
    }
```

Figure: Void addmember()



```
    void showMember()
    {
        cout << endl << name << " " << id << " " << m_type << " " << crd_point << endl;
    }
```

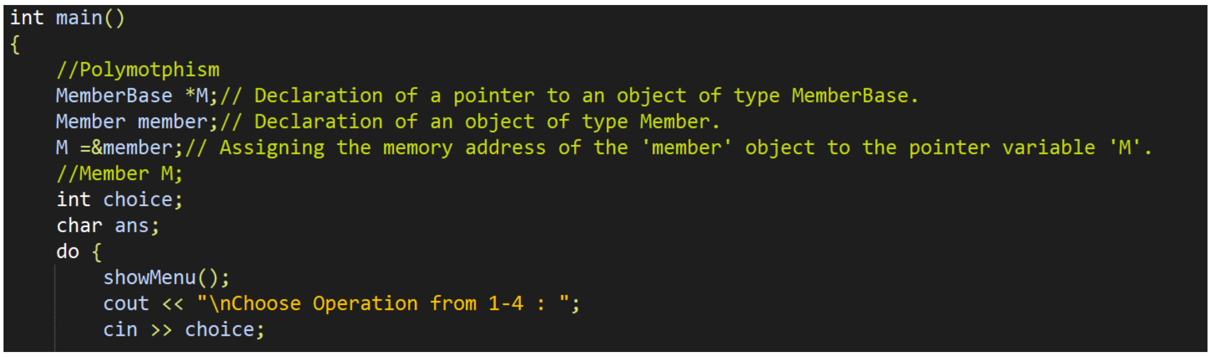
Figure: Void showmember()



```
    void modifyMember()
    {
        char new_m_type[10];
        int new_crd_point;
        cout << "Input new data\n";
        cout << "Enter new name :";
        cin >> name;
        cout<< "Enter new member type :";
        cin >> new_m_type;
        cout<< "Enter new credit point :";
        cin >> new_crd_point;
        strcpy(m_type, new_m_type); // use strcpy to copy string
        crd_point = new_crd_point;
    }
};
```

Figure: Void modifymember()

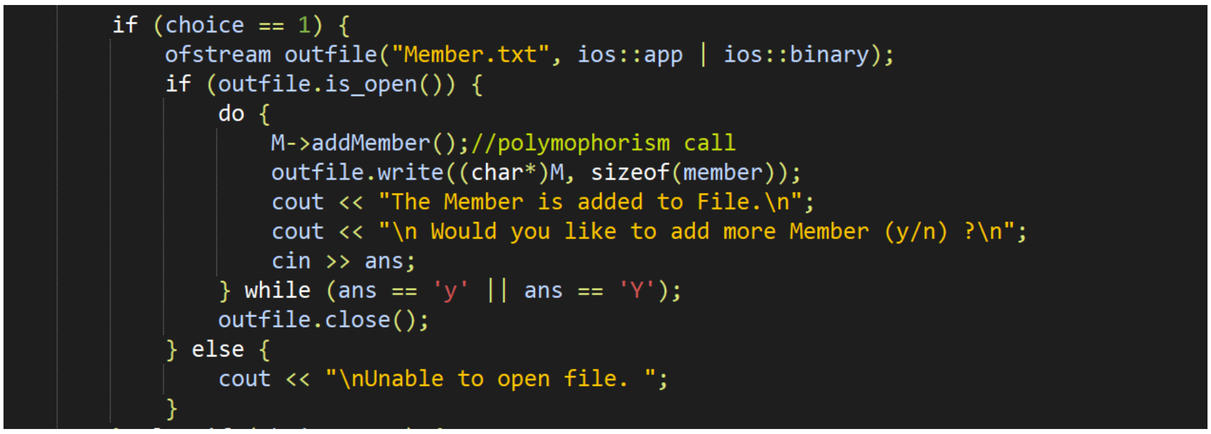
A member of the class is created in the main function. This object and as a container for the member data. The user has to pick Any kind of function they want from show menu to store in the defined integer variable choice. This program continues to decide upon using the character variable ans.
This continues until the user chooses to quit, the menu will be shown. And for the input will be sent to them via a do-while loop.
This structure offers the fundamental building block for a menu-driven application. The user has to interact with the system by choosing the option from the display. The loop's subsequent cold would provide the necessary functions by handling the various scenarios based on the user's decision value.



```
int main()
{
    //Polymotphism
    MemberBase *M;// Declaration of a pointer to an object of type MemberBase.
    Member member;// Declaration of an object of type Member.
    M =&member;// Assigning the memory address of the 'member' object to the pointer variable 'M'.
    //Member M;
    int choice;
    char ans;
    do {
        showMenu();
        cout << "\nChoose Operation from 1-4 : ";
        cin >> choice;
```

Figure: Main Function()

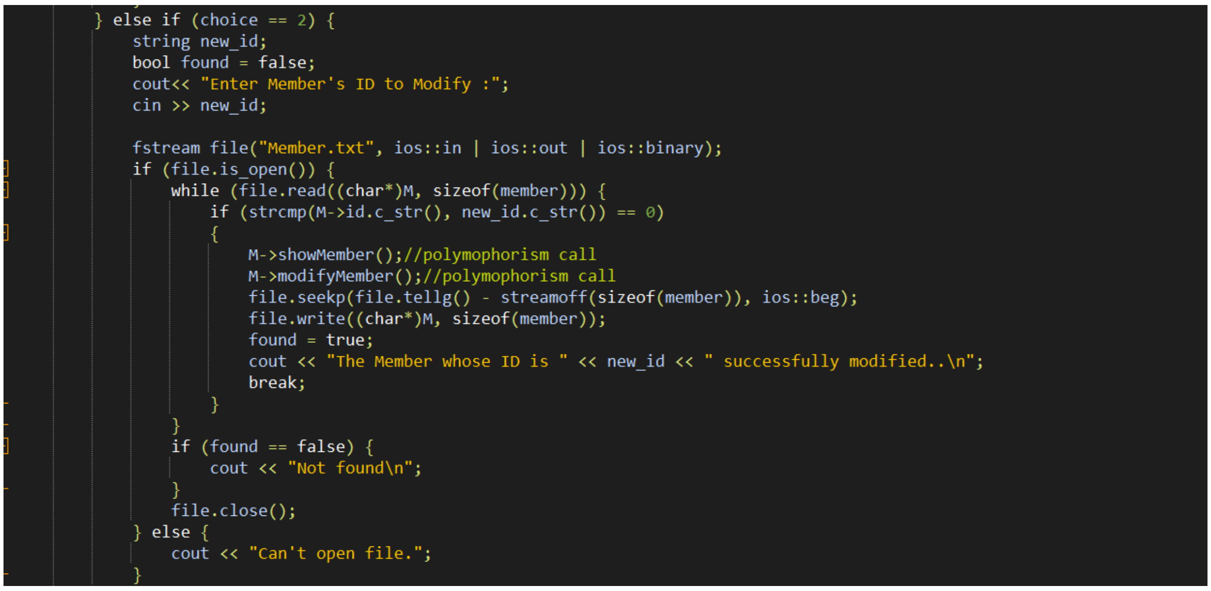
According to the program display below it starts by trying to open "member.txt" which is in binary mode. This is the best for writing data that is not text to add members. By using the addmember() in the do-while loop, it allows for the continuous adding of members. The polymorphic function calls at member points to an object-oriented methodology that would make it possible to introduce several member kinds. By including the member object in the file in binary, you can make sure that the data is kept in a small format and that the user is prompted to participate interactively, which makes the procedure easy to use. The last step in fine handling is to close the file and provide error handling. Where both of which are essential for resource management.



```
        if (choice == 1) {
            ofstream outfile("Member.txt", ios::app | ios::binary);
            if (outfile.is_open()) {
                do {
                    M->addMember();//polymophorism call
                    outfile.write((char*)M, sizeof(member));
                    cout << "The Member is added to File.\n";
                    cout << "\n Would you like to add more Member (y/n) ?\n";
                    cin >> ans;
                } while (ans == 'y' || ans == 'Y');
                outfile.close();
            } else {
                cout << "\nUnable to open file. ";
            }
```

Figure: Choice 1 to Adding Members

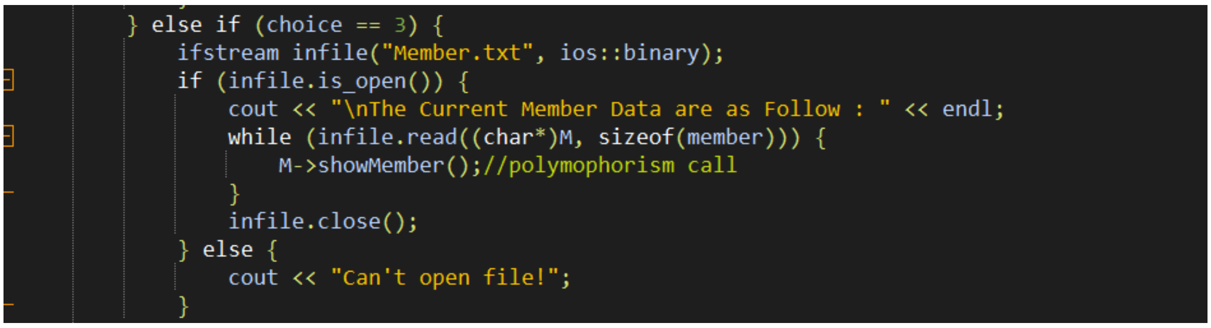
When the user selects option 2, The program will ask memberID to be modified by the user. It then opens the "Member.txt" file In both read and write binary mode. This code iterates through the file, reading member data into a member object. If the program found the merchant idea it will display the current member detail using polymorphism. Which allowed the user to modify data in another polymorphic function call. After the user at the new member type, credit points, and name, it overrides the Member information in the file. On the other hand, if the matching ID is not found and message will be displayed. Error handling is included to address cases where the file cannot be opened.



```
} else if (choice == 2) {
    string new_id;
    bool found = false;
    cout<< "Enter Member's ID to Modify :";
    cin >> new_id;

    fstream file("Member.txt", ios::in | ios::out | ios::binary);
    if (file.is_open()) {
        while (file.read((char*)M, sizeof(member))) {
            if (strcmp(M->id.c_str(), new_id.c_str()) == 0)
            {
                M->showMember();//polymophorism call
                M->modifyMember();//polymophorism call
                file.seekp(file.tellg() - streamoff(sizeof(member)), ios::beg);
                file.write((char*)M, sizeof(member));
                found = true;
                cout << "The Member whose ID is " << new_id << " successfully modified..\n";
                break;
            }
        }
        if (found == false) {
            cout << "Not found\n";
        }
        file.close();
    } else {
        cout << "Can't open file.";
    }
```

Figure: Choice 2 to Modify Existing Members

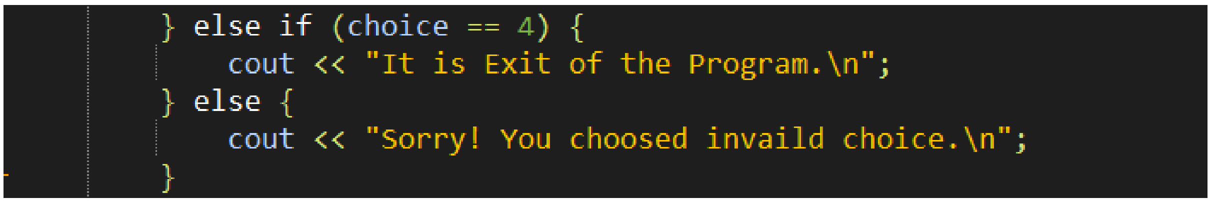
Option 3 is the display member function. The show member data is handled by the code given below. In read-binary mode, this option opens the "Member.txt" file. The file shows a message indicating the beginning added of the member data if it is successful. The added member data are read first before displaying across the file. After reading the added members are displayed using the polymorphism-demonstrating showMember() method. Lastly, if an error appears, it'll display that the file is unable to open.



```
} else if (choice == 3) {
    ifstream infile("Member.txt", ios::binary);
    if (infile.is_open()) {
        cout << "\nThe Current Member Data are as Follow : " << endl;
        while (infile.read((char*)M, sizeof(member))) {
            M->showMember();//polymophorism call
        }
        infile.close();
    } else {
        cout << "Can't open file!";
    }
```

Figure: Choice 3 to Display Existing Members

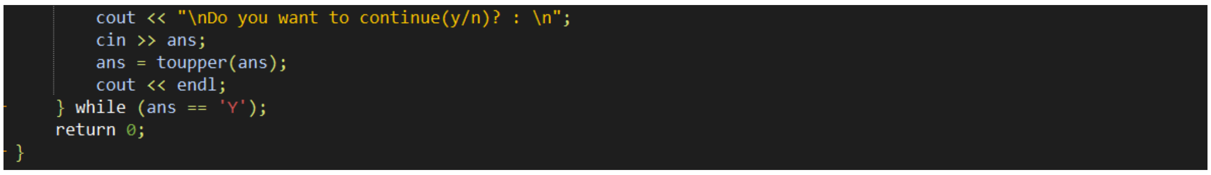
The user's selection is handled by this code segment. The software will end if the user chooses option 4, which will result in an exit message being shown. If the user selects any other incorrect option, an error notice alerting them to the incorrect input is shown.



```
} else if (choice == 4) {
    cout << "It is Exit of the Program.\n";
} else {
    cout << "Sorry! You choosed invaild choice.\n";
}
```

Figure: Choice 4 to Exit

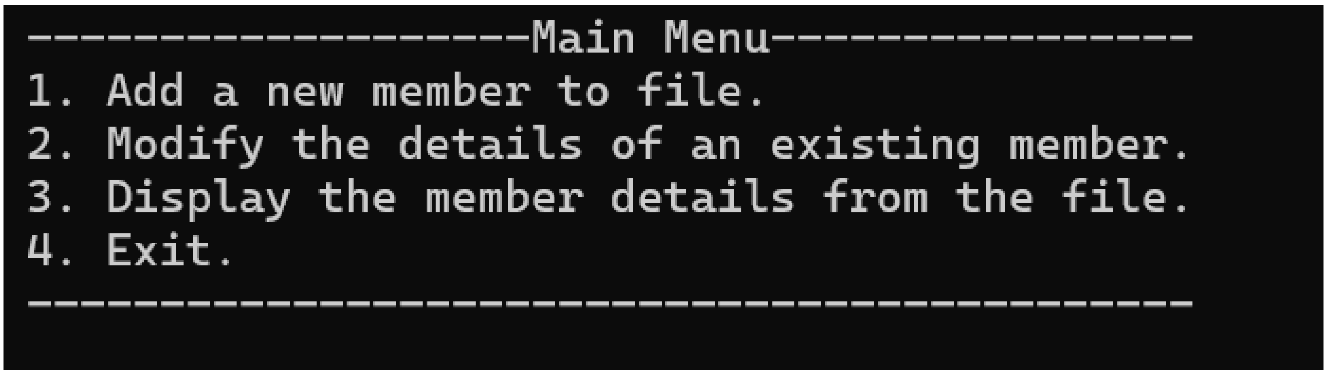
According to the program below, a loop continuation is implemented. When the user inputs y(yes) or n(no), y(yes) will function to proceed to the program again and n(no) will function to end the program. Not to be affected by cases, the input is changed to uppercase as well. The while loop guarantees that the application will keep running as long as the user inputs 'y'. The loop finishes, and the program exits with a return value of zero once the user inputs any other characters.



```
        cout << "\nDo you want to continue(y/n)? : \n";
        cin >> ans;
        ans = toupper(ans);
        cout << endl;
    } while (ans == 'Y');
    return 0;
}
```

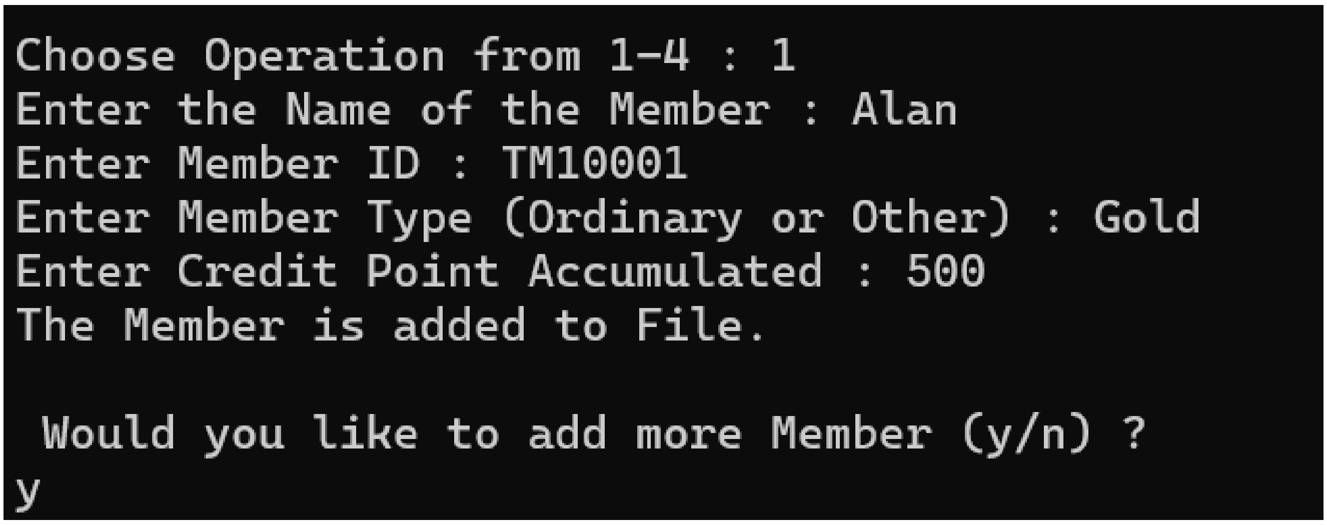
Figure: End of Program

OUTPUT:



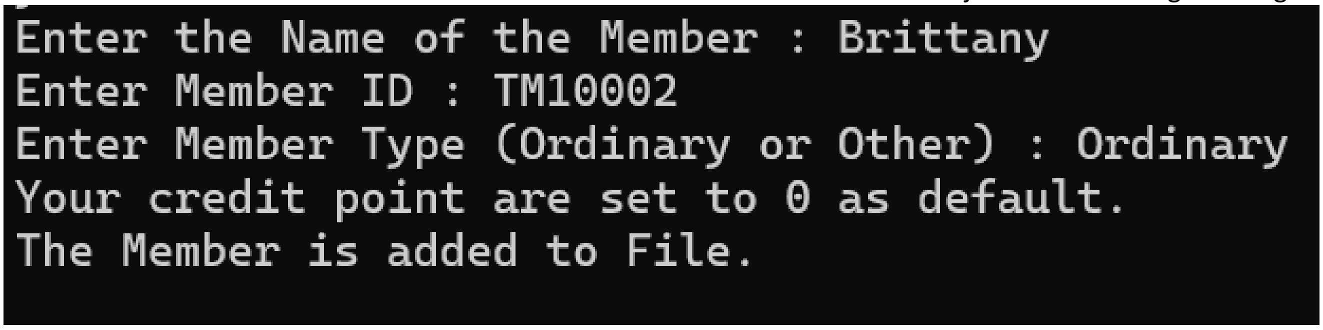
```
--------------------Main Menu-----------------
1. Add a new member to file.
2. Modify the details of an existing member.
3. Display the member details from the file.
4. Exit.
----------------------------------------------
```

This menu has four ways to manage member data: add a new member, edit current member information, show member information, and quit the program. Users can input a number between 1 and 4 to select an option.



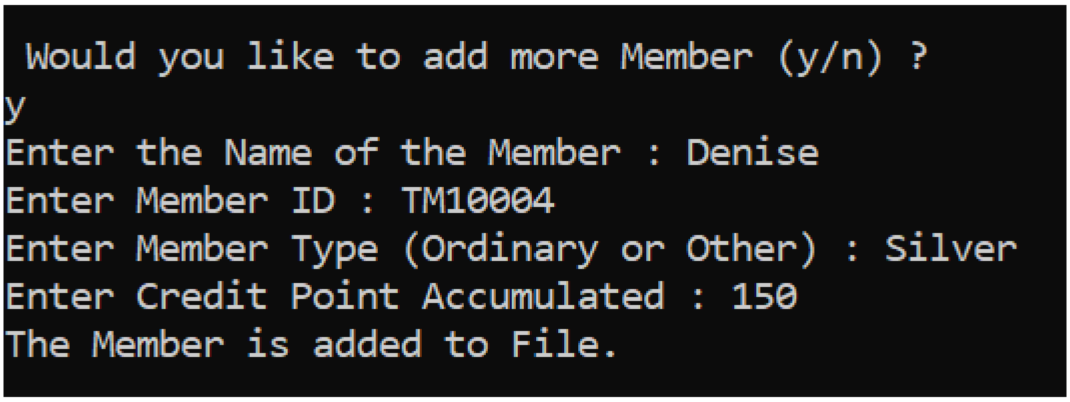
```
Choose Operation from 1-4 : 1
Enter the Name of the Member : Alan
Enter Member ID : TM10001
Enter Member Type (Ordinary or Other) : Gold
Enter Credit Point Accumulated : 500
The Member is added to File.

 Would you like to add more Member (y/n) ?
y
```

Here is the output image for option 1, which adds a new member to a file, is seen here. The member's name, kind, and total credit points must be entered by the user. The system asks whether the user wishes to add another person once they confirm the addition and provide the information for a member named Alan who has 500 points and a gold membership. This shows how to manage member data using a basic command-line tool.



```
Enter the Name of the Member : Brittany
Enter Member ID : TM10002
Enter Member Type (Ordinary or Other) : Ordinary
Your credit point are set to 0 as default.
The Member is added to File.
```

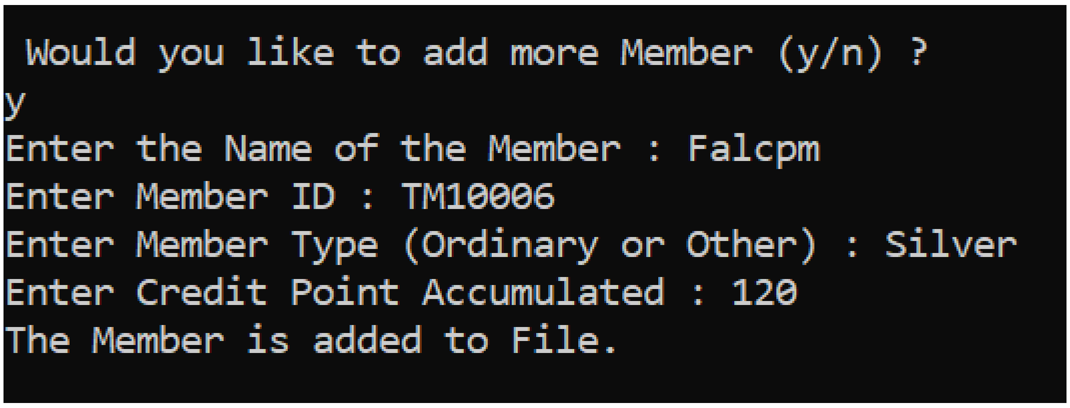
As well as same for Ordinary Members. However, if the user is ordinary member, the system will not as ask for credit points and it'll set 0 as default.



```
 Would you like to add more Member (y/n) ?
y
Enter the Name of the Member : Denise
Enter Member ID : TM10004
Enter Member Type (Ordinary or Other) : Silver
Enter Credit Point Accumulated : 150
The Member is added to File.
```

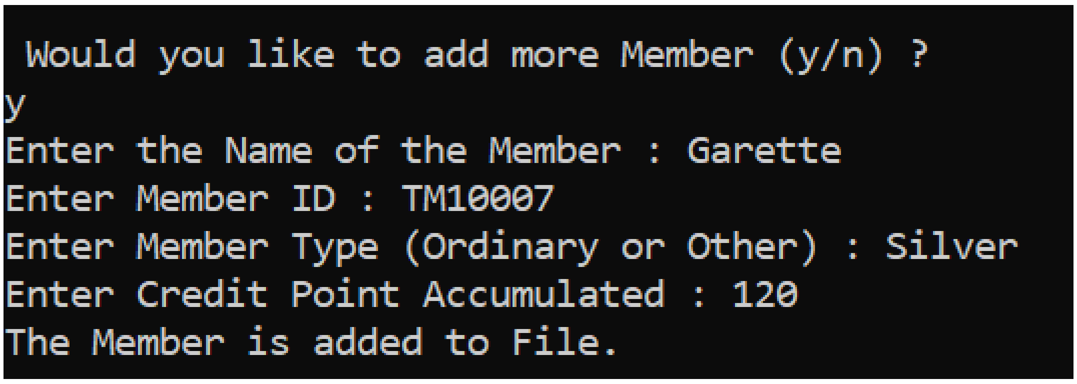
For the members who are silver type, the system will ask to input same as gold member type.

Here is a member whose name is Falcon but the spelling was typed wrong.



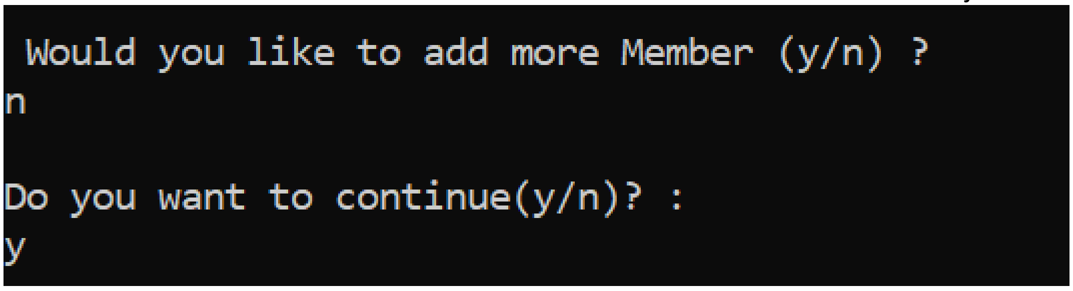
```
 Would you like to add more Member (y/n) ?
y
Enter the Name of the Member : Falcpm
Enter Member ID : TM10006
Enter Member Type (Ordinary or Other) : Silver
Enter Credit Point Accumulated : 120
The Member is added to File.
```

And here is a member who type is member type and credit point wrong.



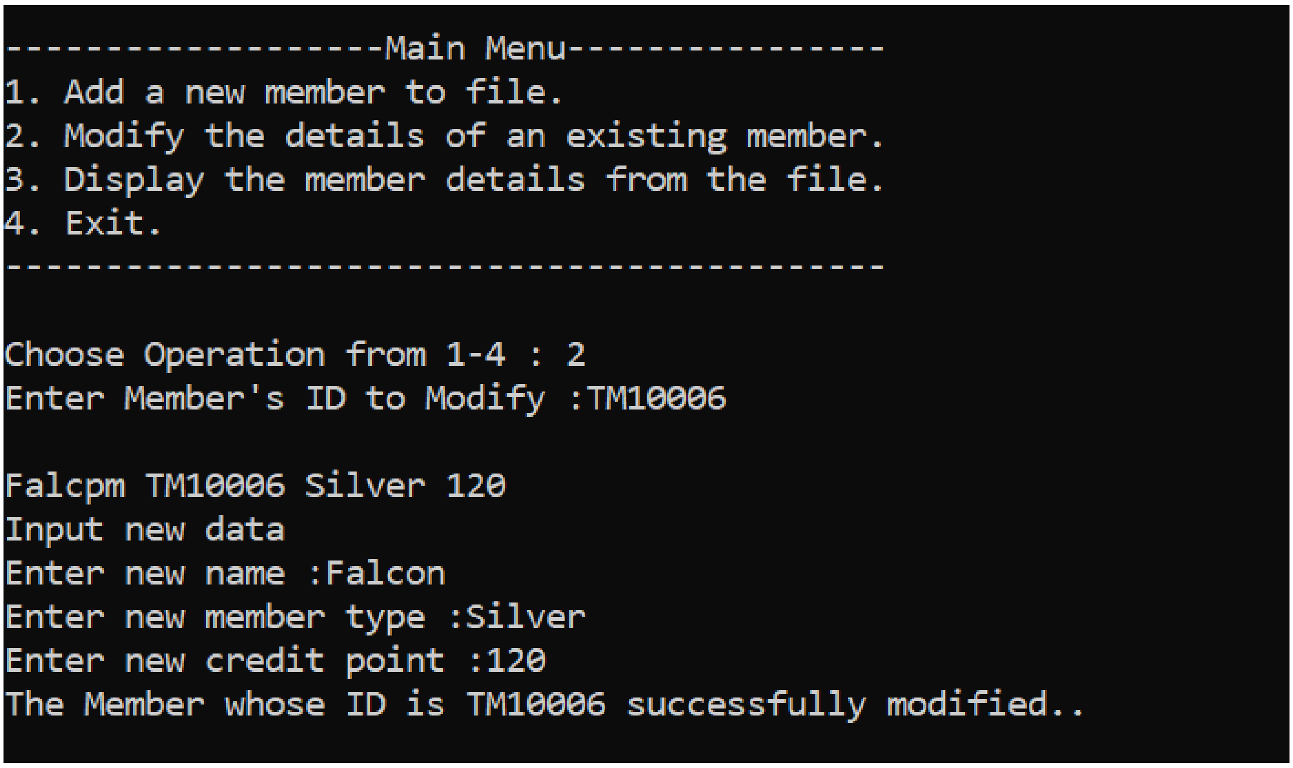
```
 Would you like to add more Member (y/n) ?
y
Enter the Name of the Member : Garette
Enter Member ID : TM10007
Enter Member Type (Ordinary or Other) : Silver
Enter Credit Point Accumulated : 120
The Member is added to File.
```

As both of the members need to modify their errors.



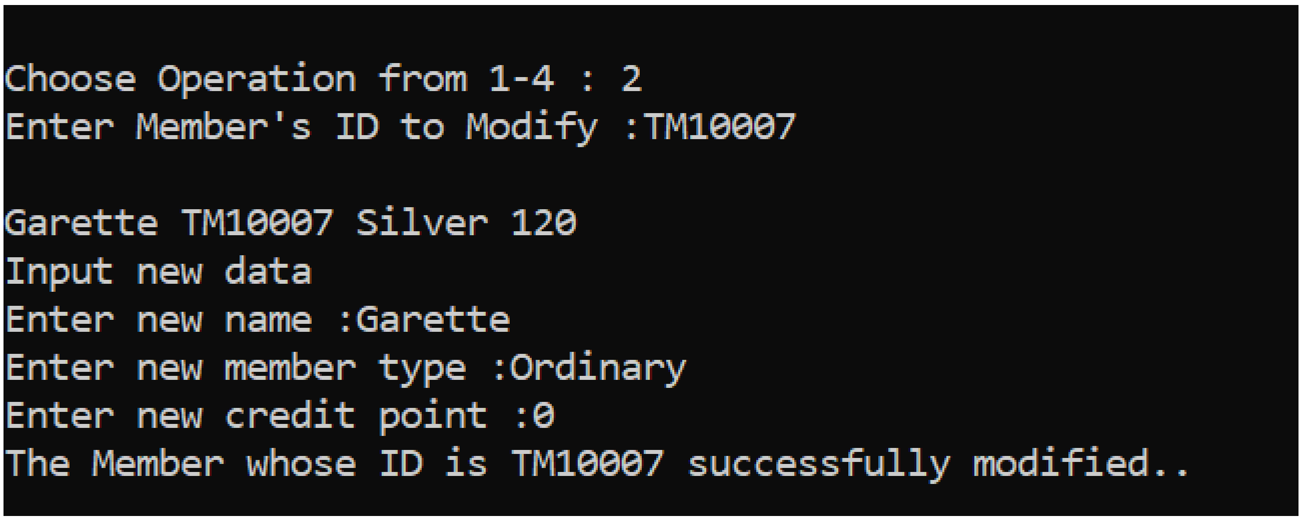
```
 Would you like to add more Member (y/n) ?
n

Do you want to continue(y/n)? :
y
```

After adding a member, the system will ask, if we want to add more members. If we type y or Y, we would be able to add new member. When we type n which is for no, the system will ask if we want to continue the operation. We had error above which need to be modified. So the user have choose option 2 which is modify function.

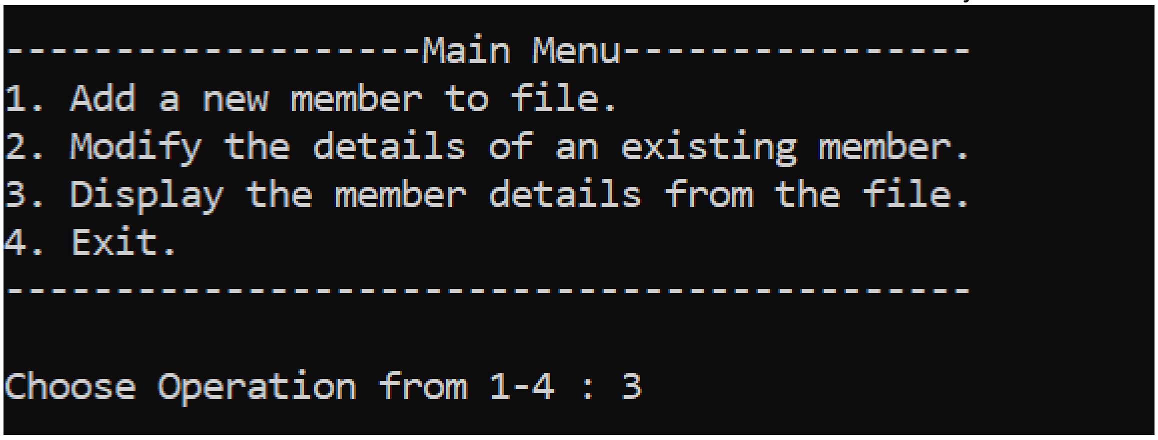


```
-------------------Main Menu----------------
1. Add a new member to file.
2. Modify the details of an existing member.
3. Display the member details from the file.
4. Exit.
--------------------------------------------

Choose Operation from 1-4 : 2
Enter Member's ID to Modify :TM10006

Falcpm TM10006 Silver 120
Input new data
Enter new name :Falcon
Enter new member type :Silver
Enter new credit point :120
The Member whose ID is TM10006 successfully modified..
```

It is ok if we don't remember the main menu to choose the operation, the system will show the main menu again. To input new data, The system will ask member id to modify. After inputting ID, the system will show format as shown above. The system will ask for new name, new member type and new credit point.

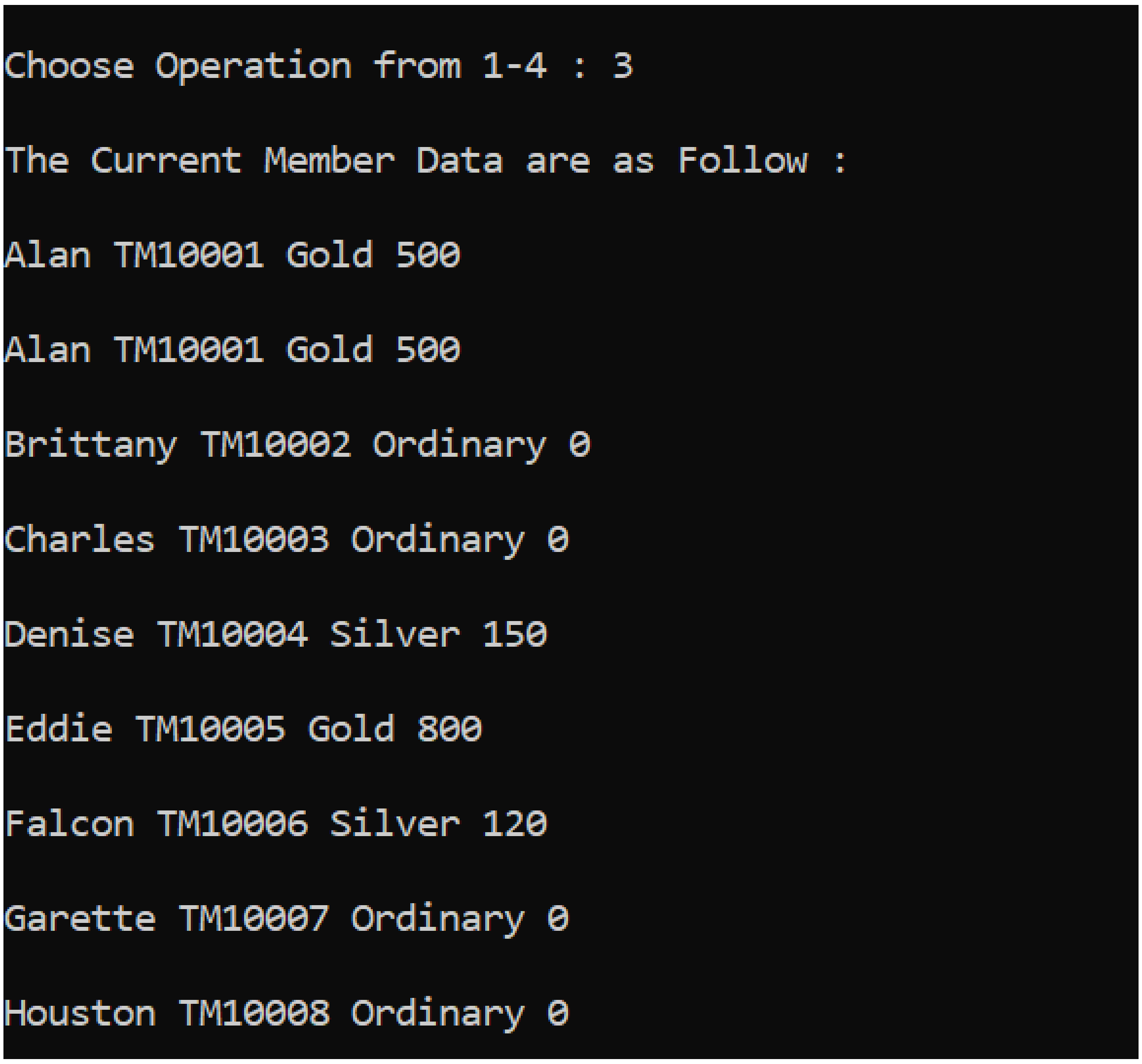


```
Choose Operation from 1-4 : 2
Enter Member's ID to Modify :TM10007

Garette TM10007 Silver 120
Input new data
Enter new name :Garette
Enter new member type :Ordinary
Enter new credit point :0
The Member whose ID is TM10007 successfully modified..
```



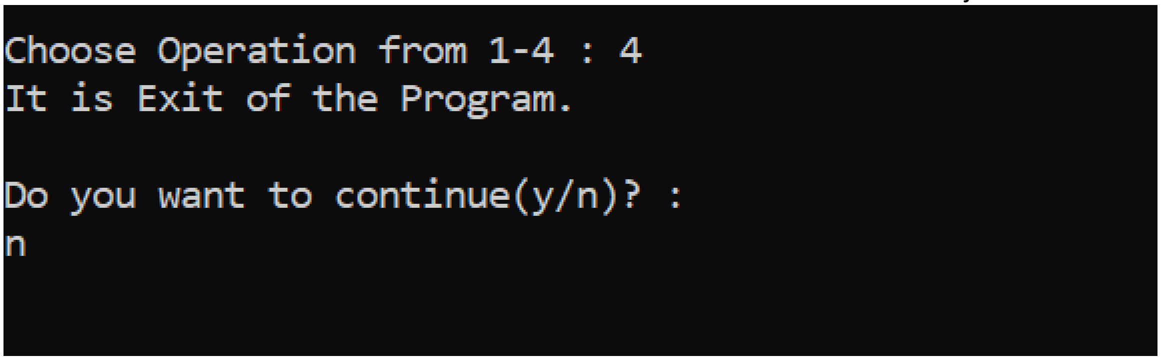
```
-------------------Main Menu----------------
1. Add a new member to file.
2. Modify the details of an existing member.
3. Display the member details from the file.
4. Exit.
--------------------------------------------

Choose Operation from 1-4 : 3
```

After adding members and modifying members, The user now want to display the members.



```
Choose Operation from 1-4 : 3

The Current Member Data are as Follow :

Alan TM10001 Gold 500

Alan TM10001 Gold 500

Brittany TM10002 Ordinary 0

Charles TM10003 Ordinary 0

Denise TM10004 Silver 150

Eddie TM10005 Gold 800

Falcon TM10006 Silver 120

Garette TM10007 Ordinary 0

Houston TM10008 Ordinary 0
```

As we can see the picture above, the user have added all the member and now display it. As in the picture the user have input Alan TM10001 for twice, If we want to modify it we would be able to modify only the first one. This mean if we put the same number for twice we won’t be able to modify the second one.



```
Choose Operation from 1-4 : 4
It is Exit of the Program.

Do you want to continue(y/n)? :
n
```

This option is number four option which is an exit of program.

Task 2

Explanation of Inheritance

A key idea in object-oriented programming (OOP) is inheritance. A class can inherit characteristics and actions (functions) from another class through inheritance. Base classes and derived classes are the two different kinds of classes.

Base Class: The class with all attributes, including member functions and member data, is called the base class (sometimes known as the parent class or superclass).

Derived Class: This class is a descendant of the base class, often known as a child class or subclass. It can override the inherited methods and properties, or it can have new ones. A class that inherits from another class has access to all public and protected members (fields, methods) of the base class.

Benefits of Code Reusability:

Reusing an existing class's code in a new class is made possible via inheritance. As a result, developing and maintaining code takes less time and effort and redundancy is decreased.

Adaptability

It is possible to add new features to classes that already exist without changing them. As a result, the system is easier to expand and more versatile.

Sustainability:

Modifications made to the base class instantly affect the derived classes as well. This facilitates code management and updates.

Adjustability:

Code is arranged into a modular structure with the aid of inheritance, which facilitates management and understanding.

The drawbacks of inheritance

The drawbacks of inheritance include tight coupling, which makes the base class and derived classes highly dependent on one another. Modifications made to the base class may affect derived classes in an unexpected way.

Intricacy:

Complex class hierarchies brought forth by inheritance might make the code more difficult to read and maintain.

Delicate Base Class Issue:

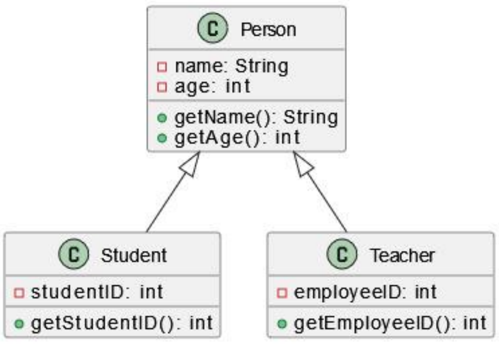
Modifications made to the base class may unintentionally disrupt the functioning of derived classes, particularly if inheritance was not considered while creating the base class.

Overhead:

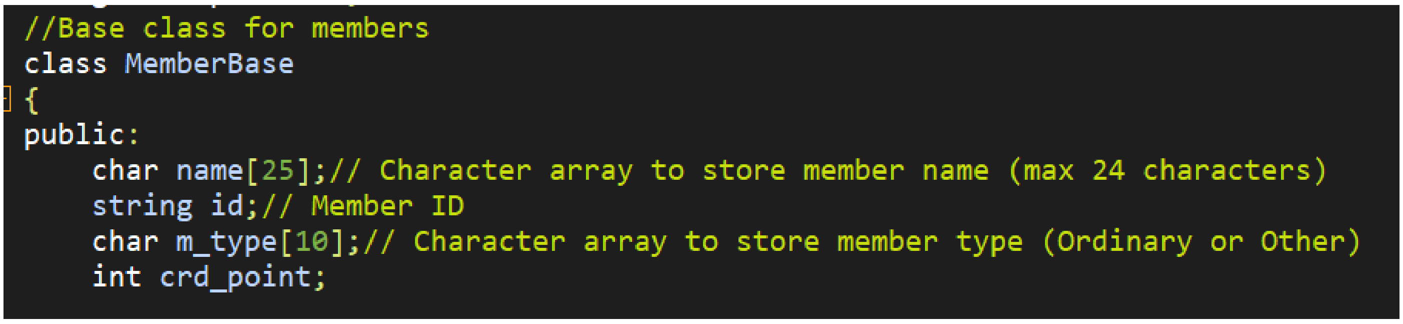
Unnecessary overhead in terms of memory and computation might be introduced by inherited techniques.

Limited Adaptability:

Because inheritance makes it impossible to modify a parent class's properties without also changing all of its child classes, it might reduce flexibility.



The UML class diagram demonstrates an inheritance structure with a base class called Person and two derived classes, Student and Teacher. The Person class provides methods for accessing name, age,

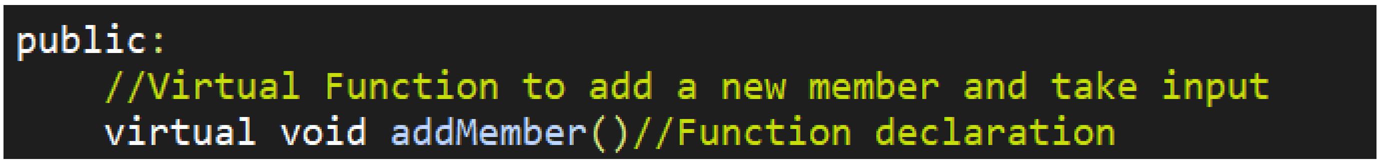


```
//Base class for members
class MemberBase
{
public:
    char name[25];// Character array to store member name (max 24 characters)
    string id;// Member ID
    char m_type[10];// Character array to store member type (Ordinary or Other)
    int crd_point;
```

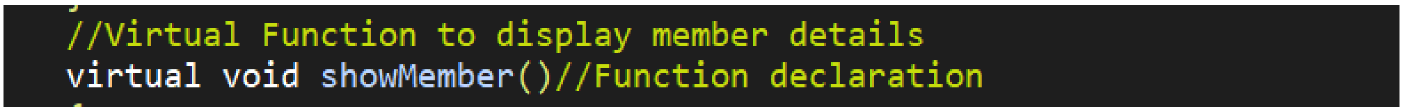
studentID, and other information, while the Student and Teacher classes inherit its properties and methods.

In this picture there is a base class for member which is class member base. A class member base has specifier shown in public and its member data.

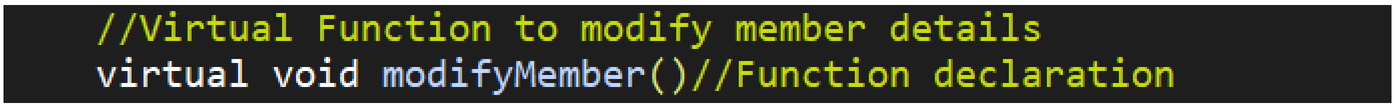
For the member functions, We need to declare member function as virtual in base class, and even though we can write its full function. Because of inheritance, it'd be only declaration.



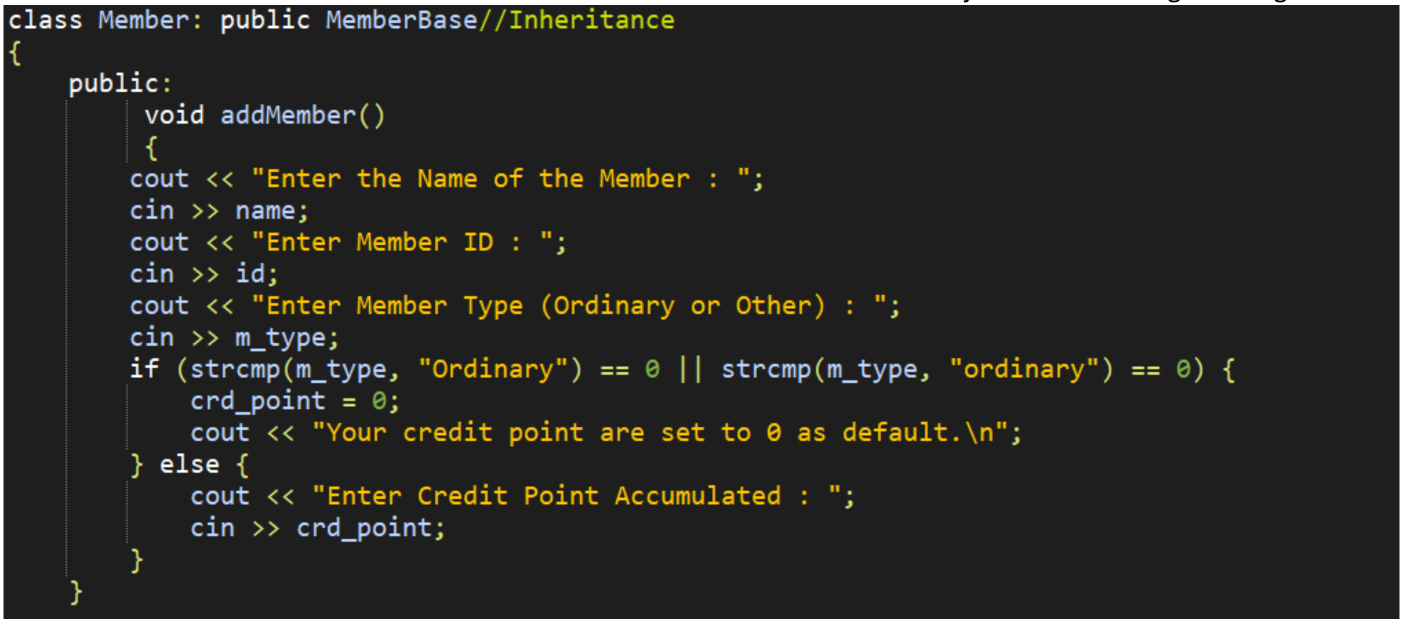
```
public:
    //Virtual Function to add a new member and take input
    virtual void addMember()//Function declaration
```



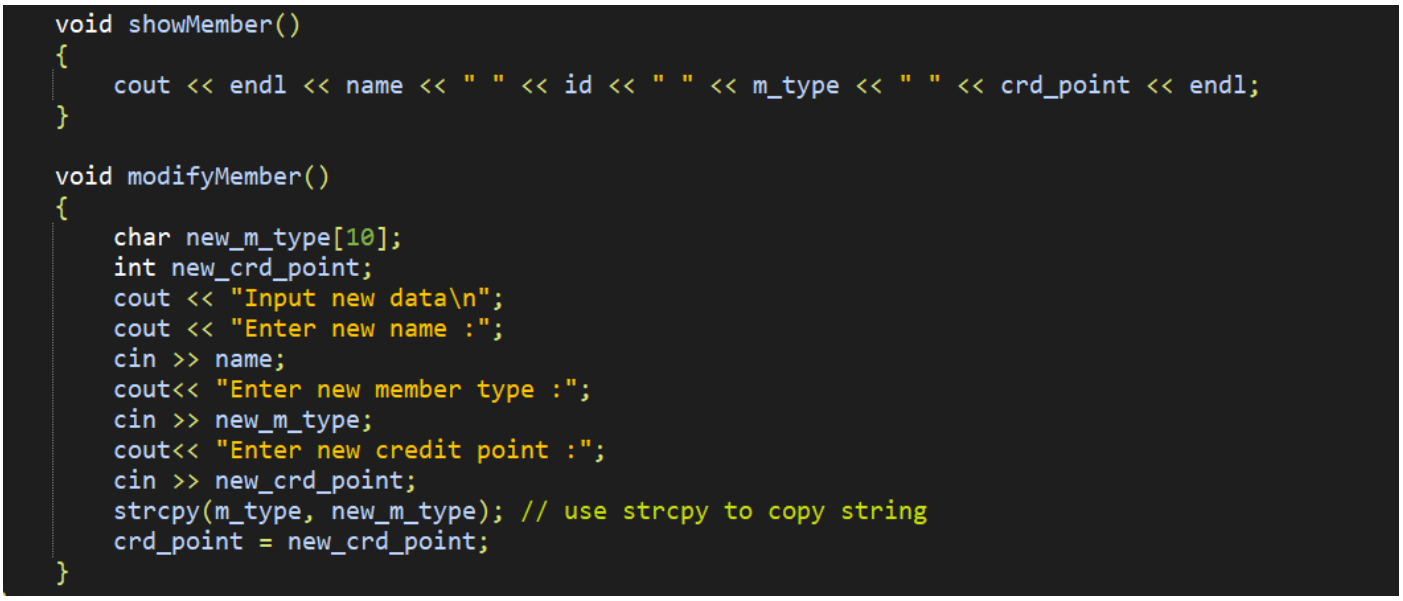
```
    //Virtual Function to display member details
    virtual void showMember()//Function declaration
```



```
    //Virtual Function to modify member details
    virtual void modifyMember()//Function declaration
```



```
class Member: public MemberBase//Inheritance
{
    public:
         void addMember()
         {
        cout << "Enter the Name of the Member : ";
        cin >> name;
        cout << "Enter Member ID : ";
        cin >> id;
        cout << "Enter Member Type (Ordinary or Other) : ";
        cin >> m_type;
        if (strcmp(m_type, "Ordinary") == 0 || strcmp(m_type, "ordinary") == 0) {
            crd_point = 0;
            cout << "Your credit point are set to 0 as default.\n";
        } else {
            cout << "Enter Credit Point Accumulated : ";
            cin >> crd_point;
        }
    }
```



```
   void showMember()
   {
       cout << endl << name << " " << id << " " << m_type << " " << crd_point << endl;
   }

   void modifyMember()
   {
       char new_m_type[10];
       int new_crd_point;
       cout << "Input new data\n";
       cout << "Enter new name :";
       cin >> name;
       cout<< "Enter new member type :";
       cin >> new_m_type;
       cout<< "Enter new credit point :";
       cin >> new_crd_point;
       strcpy(m_type, new_m_type); // use strcpy to copy string
       crd_point = new_crd_point;
   }
```

After the base class, there would be a derived class which is known as sub-class (child class). A child class inherit All the member data and member functions from base class. This member functions are the actual function which will work.

TASK – 3

Polymorphism

Polymorphism is a fundamental concept in OOP which allows objects of different types to be treated as instances of a common superclass. Polymorphism is supported by inheritance, which enables objects to be regarded as instances of their parent class. This makes code more reusable and adaptable. It can be accomplished using either runtime (dynamic) or compile-time (static) methods; method overloading is used for compile-time polymorphism, and method overriding is used for runtime polymorphism. Code is more flexible and reusable when it has polymorphism, which also makes it easier to maintain and expand.

Even though Polymorphism is a powerful concept. It also offers several advantages and some drawbacks.

Advantages:

Code Reusability: Writing more generic and reusable code is made possible via polymorphism. Redundancy can be minimized by using methods to operate on objects of distinct classes.

Flexibility and Maintainability: It improves the flexibility and maintainability of the code. Updates are made easier by the automatic propagation of changes made to derived classes from the base class.

Decreased Coupling: Polymorphism makes a system more modular by coding to an interface rather than a particular implementation, which lessens the reliance between components.

Complicated System Simplification: By allowing objects to be seen as instances of their parent class, it facilitates the management and organization of complicated systems.

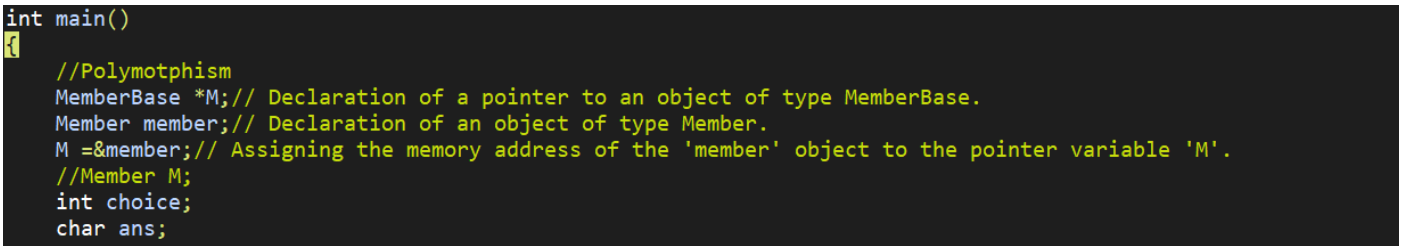
Drawbacks:

Performance cost: Because runtime type verification and method resolution are required, polymorphism—especially dynamic polymorphism, or method overriding—may result in performance cost.

Enhanced Complexity: Although polymorphism makes some things simpler, it can also increase the codebase's complexity and make it more difficult for people who are unfamiliar with the idea to understand.

Debugging Difficulties: Since the actual function being called is determined at runtime, debugging polymorphic code might be more difficult.

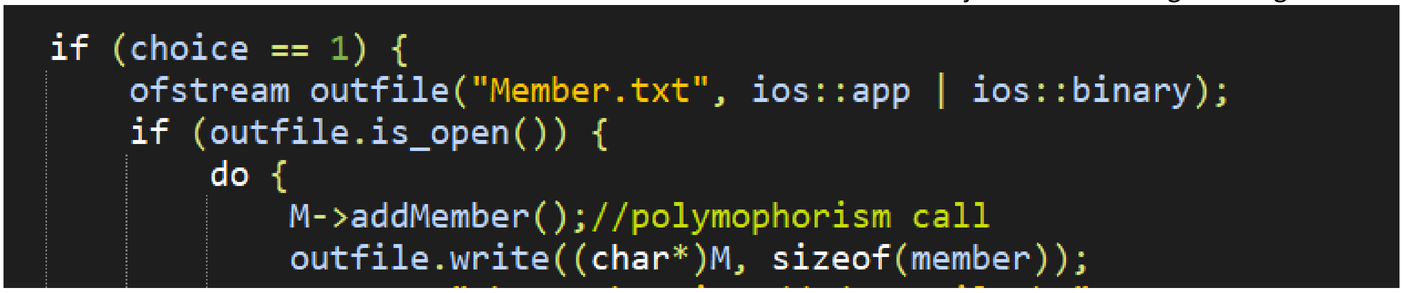
According to the image shown below, the image contains main function. That declare and assigns the address of a Member object to a pointer to a MemberBase object. This demonstrates that polymorphism use the Member object's functions by enabling with the pointer. There are other declarations of variables for a character called ans and an integer called choice. This sample can help you learn how pointers and object assignments are used in C++ polymorphism.



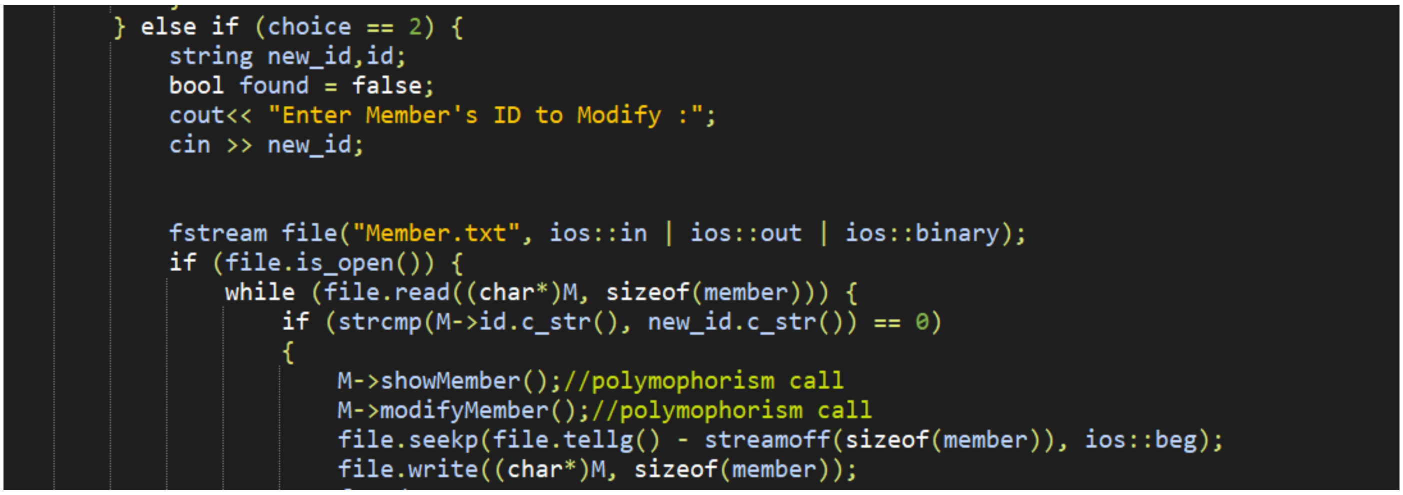
```
int main()
{
    //Polymotphism
    MemberBase *M;// Declaration of a pointer to an object of type MemberBase.
    Member member;// Declaration of an object of type Member.
    M =&member;// Assigning the memory address of the 'member' object to the pointer variable 'M'.
    //Member M;
    int choice;
    char ans;
```

The picture shown below display that allows the user to choose how to handle file operations. An output file stream (ofstream) called outfile is generated to open "Member.txt" in append and binary modes if the variable choice equals 1. The code then verifies that the file was opened successfully.

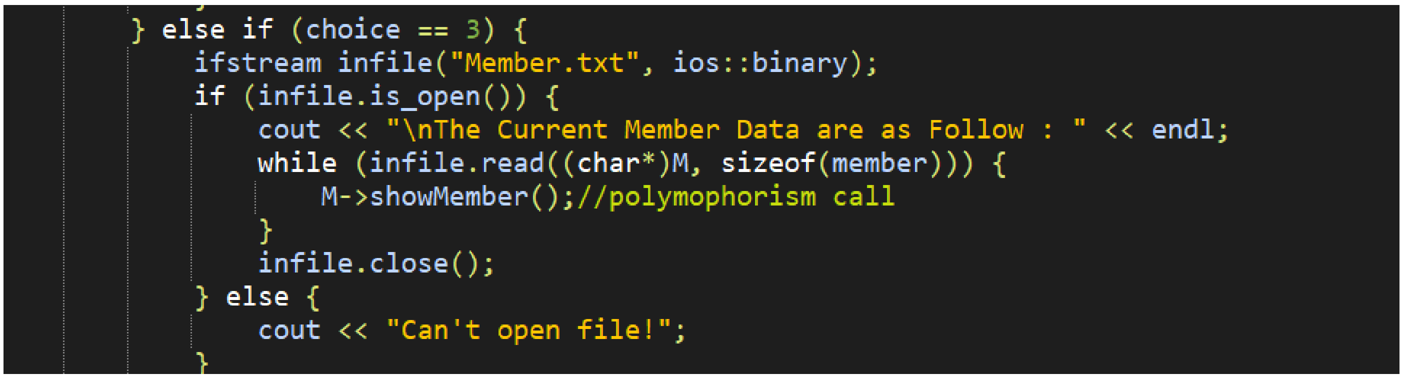
A polymorphic function called M->addMember() is called inside a loop, and outfile is used to write the object's contents to the file.write(sizeof(member)), write((char*)M). This illustrates the use of C++ polymorphism and file management.
As well as for choices 2 and choices 3 which are modify member and display member functions. Polymorphic function called M->modifyMember() and called M->showMember() The object's contents are written to the file using outfile when Member() happens inside of a loop.write((char*)M), write(sizeof(member)). This demonstrates how to leverage file management with C++ polymorphism.



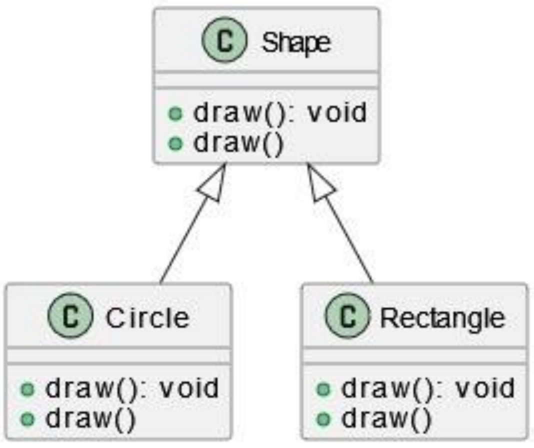
```
if (choice == 1) {
    ofstream outfile("Member.txt", ios::app | ios::binary);
    if (outfile.is_open()) {
        do {
            M->addMember();//polymophorism call
            outfile.write((char*)M, sizeof(member));
```



```
    } else if (choice == 2) {
        string new_id,id;
        bool found = false;
        cout<< "Enter Member's ID to Modify :";
        cin >> new_id;


        fstream file("Member.txt", ios::in | ios::out | ios::binary);
        if (file.is_open()) {
            while (file.read((char*)M, sizeof(member))) {
                if (strcmp(M->id.c_str(), new_id.c_str()) == 0)
                {
                    M->showMember();//polymophorism call
                    M->modifyMember();//polymophorism call
                    file.seekp(file.tellg() - streamoff(sizeof(member)), ios::beg);
                    file.write((char*)M, sizeof(member));
```



```
    } else if (choice == 3) {
        ifstream infile("Member.txt", ios::binary);
        if (infile.is_open()) {
            cout << "\nThe Current Member Data are as Follow : " << endl;
            while (infile.read((char*)M, sizeof(member))) {
                M->showMember();//polymophorism call
            }
            infile.close();
        } else {
            cout << "Can't open file!";
        }
```



This UML class diagram showcases polymorphism in object-oriented programming, with Circle and Rectangle derived classes and a base class called Shape. Draw() method overrides Circle and Rectangle, demonstrating how shapes can share similar interfaces.