Layla Smith, Dan Ha, Ben Anderson, Ryan Niiya, Anh Ho, Linh Nguyen, Thomas Brooks

The Chocoholics Anonymous Simulator

Design Document

Table of Contents

1 Introduction 3

1.1 Purpose and Scope 3

1.2 Target Audience 4

1.3 Terms and Definitions 4

2 Design Considerations 6

2.1 Constraints and Dependencies 6

2.2 Methodology 6

3 System Overview 8

4 System Architecture 10

4.1 Model 10

4.1.1 Member Data & Data Operations 10

4.1.2 Provider Data & Data Operations 10

4.2 View 11

5 Detailed System Design 13

5.1 ChocAn Model Class 13

5.1.1 Data Members of the Model Class 13

5.1.2 Functions of the Model Class 14

5.2 ChocAn Provider Class 15

5.2.1 Data Members of the Provider Class 15

5.2.2 Functions of the Provider Class 15

5.3 Member Class 16

5.3.1 Data Members of the Member Class 17

5.3.2 Functions of the Member Class 17

5.4 Manager Class 18

5.4.1 Data Members of the Manager Class 19

5.4.2 Functions of the Manager Class 19

5.5 Service List Class 20

5.5.1 Data Members of the Service List Class 21
5.5.2 Functions of the Service List Class 21
5.6 Data Structure *21*
5.6.1 Data Members of the Service List Class 22
5.6.1.1 Node Class for Provider 22
5.6.1.2 BST_Provider Class for Managing Tree 23
5.6.2 BST Class for Member 23
5.6.2.1 Node Class for Member 23
5.6.2.2 BST_Member Class for Managing Tree 24
5.6.3 Sorting Data in Data Structure by a Request 24

1 Introduction

This is a design document that will cover the design considerations, system overview, system architecture, and a detailed description of the system design for the Chocoholics Anonymous (ChocAn) software project. In this introduction section, the purpose and scope for the document and project will be described, the target audience outlined, and a section for terms and their respective definitions. For the sake of clarity, any terms that were defined in the requirements document will be defined again in this document.

1.1 Purpose and Scope

The purpose of this document is to define and demonstrate the design considerations that will be undertaken by the team for the ChocAn project. The document will be split up into 5 major sections. Following the introduction section, which is briefly described above, the document will contain design considerations, a system overview of the project, details on the system architecture, and a detailed system design. Firstly, the design considerations section will go over any constraints and dependencies (both functional and non-functional), as well an overview of the modified waterfall engineering methodology being utilized. Following this, the system overview will provide a complete, high level abstraction of the system as a whole. This will prepare readers for the system architecture section, providing diagrams and tables when needed. Next, the system architecture will provide a detailed, high level description of the various subsystems and components that will be developed by the team for the ChocAn simulator. Diagrams will be used in this section to help support the sub-systems described. This section is primarily meant for non-developers, with the next and final section, detailed system design, being more oriented towards software developers. The detailed system design section will provide more detail to the previous system architecture section, providing pseudo code, data structures, and in general a much more in-depth and software oriented approach to the system being developed.

1.2 Target Audience

This document is intended primarily for software developers and system engineers, as the design considerations, system overview, and detailed system design sections will heavily use software terminology and development processes. The detailed system design will be heavy on software design and terms, including pseudocode, data structures, and any other facets of implementation that will be required. The system architecture may also be viewed by customers, to show a general overview of the system that will be developed, but will additionally be primarily directed towards a developer audience.

1.3 Terms and Definitions

Provider: Health care professional who provides services to ChocAn members.

Member: Patients of ChocAn who pay a monthly fee for access to unlimited consultations and treatments. (See Member Card)

Manager: Manages providers at ChocAn.

User: Those who will use the software directly, in this case the Providers and Managers.

Member Card: A plastic card embossed with the member's name and a nine digit member number and incorporating a magnetic strip on which that information is encoded

Service: An activity a provider provides to benefit a member.

Service Code: A six-digit code corresponding to a service provided by providers.

Service Fee: Amount to be paid based on a service given.

Date of Service (DOS): Date (MM-DD-YYY) that a service was given.

Provider Directory: An alphabetically ordered list of service names and corresponding service codes and fees sent to the provider as an email attachment. All providers may access this directory. (See Service Code)

Provider Number: A unique identifier for each provider, used to access the ChocAn terminal.

Stakeholder: A person, company, or business who is involved in the development of the project.

Use Case: Type of interactions between the company and a user.

Electronic Funds Transfer (EFT): Digital movement of funds between one bank account and another, consisting of banking and provider accounts for this project.

ACME Accounting Services: Third party organization that will handle processing ChocAn membership fee payments.

Manager Terminal: Computer system that can run accounting reports and generate summary reports when needed.

Provider Terminal: Computer system that will allow providers to bill and enter in provided services, access fees to be paid, and access the provider directory.

Unified Modeling Language (UML) diagram: Diagram that will be utilized in this document to demonstrate the system architecture of the program.

Binary Search Tree (BST): Categorization of nodes in a hierarchical, tree-like structure. Allows for easy searching based on names or other important information.

2 Design Considerations

This section will overview the important aspects of designing our product. It outlines the required constraints and dependencies at a basic level. Referencing the requirements document on a certain topic would go into greater depth than what this will present. And lastly, the methodology in implementation of our product is described below.

2.1 Constraints and Dependencies

Functional constraints/dependencies:

- Printing a report on request
- Being able to look up members by their respective name/number
- Being able to look up providers by their respective name/number
- Printing out a summary report and the end of each week
- Ability to edit profiles of providers and members
- Records of providers who consult members

Non-functional constraints/dependencies:

- Security of member and provider data
- Reliability of system
- Usability and UI smoothness
- Simultaneous use of multiple users
- Fast response time
- Robust system of preventative data corruption or system failure

2.2 Methodology

In our design methodology, we are going to have common features of the modified waterfall model. We have a requirements document that lines up with the first step in the waterfall model. It outlines all the services, constraints, and goals of the ChocAn project. Then we have our design document, also called the system and software design document. It mainly provides the architecture of our project. Then after that, we will have a test plan document to make sure that each unit meets the specifications that we have

outlined. After this is done, we can implement the software and deliverables through extensive testing for the customer. The only step that we may not touch on as much as the others, would be the operation and maintenance of the product. Since the purpose of the project is to undergo extensive testing, preparation, and written required documentation to create a reliable product. The main purpose for going with this type of model is due to how popular, common, and professional it is. It should in theory create a product that is going to satisfy the customer whereas other methodologies could leave out certain details.

3 System Overview

Chocoholics Anonymous is an organization dedicated to helping people addicted to chocolate through consultations and treatments with health care professionals. The new data processing software will provide an easy to use system for ChocAn that: simplifies the process of inputting sensitive data about ChocAn members, providers, and services; manages and stores this data for ease of access; and prepares/compiles data for preformatted, individualized reports.

In more detail, the new data processing software will allow **providers** to:

- Check validation status of members
- Look up a member profile by a member name/number
- Look up a provider profile by a member name/number
- Charge members for their services
- Request a Provider Directory
- Add or delete members
- Update member or provider records
- Receive a report with information from the provider's form, list of services provided, number of consultations and the total fee for the week.

It will allow **managers** to

- Print a summary report of services in the past week
- Add or delete members/providers
- Update member or provider records

It will allow **members** to

- Request a service or services.
- Receive a report with respective information about the service, date, name, and the members private information.

The data processing software also compiles all this data into a record of electronic funds transfer (EFT) data which is written to the disk. The software is designed to run on a specially designed ChocAn computer terminal which has a card reader for member cards. It is also designed to accept the keyboard and card reader as input. All of this functionality will be available to the user in the form of a menu whose options vary depending on whether the user is a member, a provider, or a manager.

4 System Architecture

The system architecture will follow a modified Model-View-Controller (MVC) pattern. This architecture groups the subsystems into two components: Model and View, each of which handle a separate portion of the system.

4.1 Model

The model shall act as the ChocAn system's data manager, storing data and providing functions that allow users to operate on that data. Members and providers have some very similar data to each other, and thus shared functions for managing that shared data will greatly simplify the interface. These shall be provided in the Model base class, with Provider and Member classes inheriting from it to add their own unique pieces of data and functionality.

The shared interface shall have functions for adding, removing, updating, verifying, and outputting common data such as the member's or provider's name, address, and ID number. Both members and providers shall also be able to view full, detailed reports of the services they have received or provided (respectively) that week any time they wish. As this differs for each individual, separate lists shall be kept for each person.

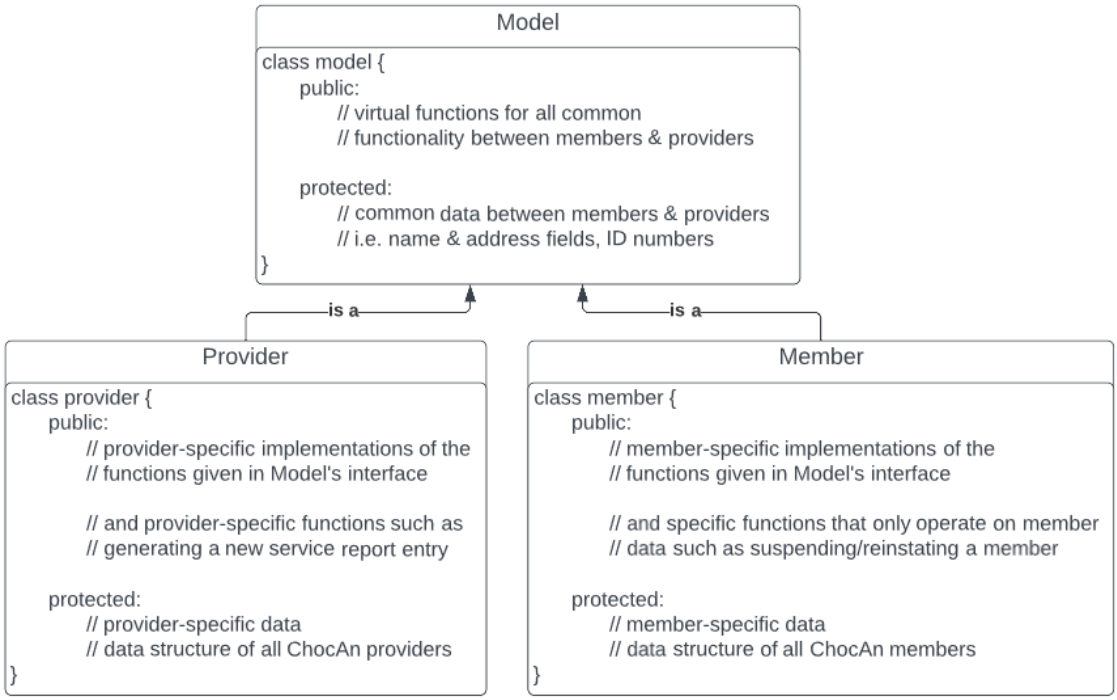
4.1.1 Member Data & Data Operations

In addition to the inherited shared functions, the Member class shall contain functions that operate specifically on member data. Examples of this include suspending and reinstating members according to ChocAn's policies, and the class shall have a special field to track that member's status.

4.1.2 Provider Data & Data Operations

Although there are no use cases that operate exclusively and uniquely on provider data (a display function, for instance, is applicable to both a list of providers and members), the Provider class shall also include methods for creating and editing service reports as only they should be able to have those permissions.

A simple overview of the of the main hierarchy is available below:



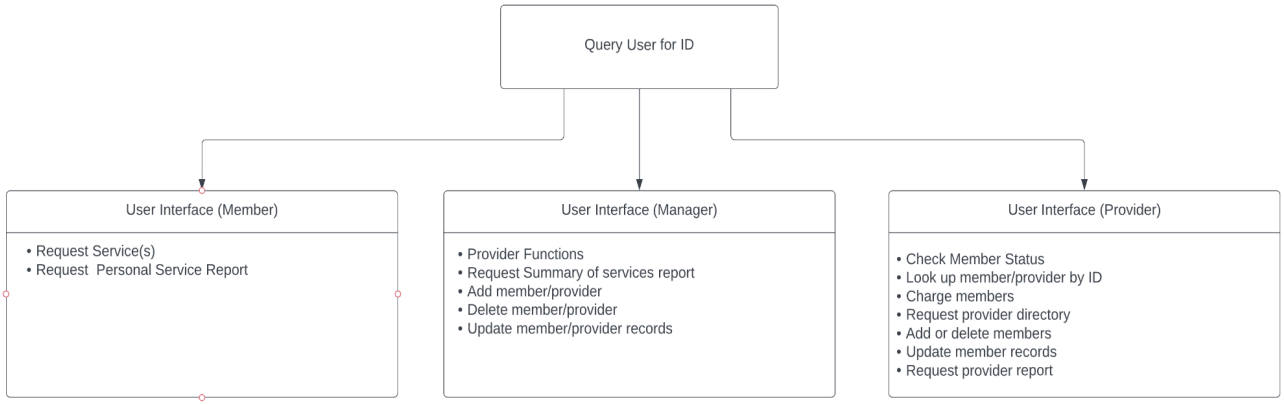
4.2 View

The view shall focus on a visual menu that the user will interact with. The menu will be through a terminal view and will receive standard keyboard input. This menu shall display commands for users to utilize: printing reports, managing member and provider information, recording consultations, and exiting the program.

User permissions will also be respected; upon starting the software, the user shall be prompted for their member/ provider identification number. Depending on who logs into the system, the menu shall reflect the commands they have access to. Members have the lowest level of access and their functions are primarily used to schedule a service from a provider and to get their own service report. Comparatively, providers have a much more robust set of functions. Providers are able to check member status and look up both members and providers, they can charge members for services, update member information, and request several documents. Managers have several extra privileges on

top of providers, namely being the ability to manage both provider and member information, along with requesting complete summary service reports. If an invalid identification number is entered, an error will be displayed and the user shall be prompted to either try again or to exit the program.

Below is a representation of how the interface flow would look:



The user's privilege level also shall be remembered for the rest of the session to further prevent unauthorized access to confidential or private data. Any functions that the user calls while using the system shall confirm that they have the correct privilege level before executing the command.

5 Detailed System Design

Based on the system architecture which has been designed with a modified Model-View-Controller (MVC) pattern for the ChocAn system, we decided to use the structure of the concept of object-oriented programming with a single-inheritance hierarchy to build the ChocAn system. We break down requirements of the design into several classes that help us easier to implement and maintain the system. There are four main classes and a few subclasses supporting the main classes. The main classes include manager, model, provider, and member. The subclasses will be used for data structures of the binary search tree and collected information, which is from providers and members, to export a weekly report. The software system will be implemented fully by C++ language programming.

5.1 ChocAn Model Class

The ChocAn system is a single inheritance hierarchy. The base class of the hierarchy, Model, has two derived classes including the provider class and the member class. The Model class will contain several common variables and functions between the relationship of the Provider class and the Member class provided by ChocAn. This helps to reduce the repetitive code in each class.

5.1.1 Data Members of the Model Class

The Model class will collect all the common information for both providers and members including first name, last name, address, city, state, and zip code. These data variables will stay in " protected" data members in order to share access privately in the hierarchy with their inherited classes only (provider and member).

```
class model
{
public:
        ...
protected:
        string first_name;
        string last_name;
        string address;
        string city;
        string state;
        int zipcode;
        ...

}
```

5.1.2 Functions of the Model Class

This Model class will be handled to add data input from the user into the ChocAn data center. Moreover, this class can be responsible for updating or changing data regards to provider and member information.

```
class model
{
public:
        model();
        model(const model & copy); // copy constructor
        model(string , string , string, int);
        virtual ~model();
        virtual void input();
        virtual void display() const;
        virtual void read() const;
        virtual void update_info();

        void update_address(string new_address, string new_city, string new_state, int new_zipcode);
        void update_email(string );
        bool update_name();
        bool verifyID(int ID_check);
        ...

private:
        string first_name;
        string last_name;
        string address;
        string city;
        string state;
        int zipcode;
        string email;
        ...

}
```

This class manages some functions that are used for both provider and member to store and provide data as requested. There have four main functions including:

- input(): This function is used to get data from users when they enter it from the keyboard.
- display(): The display function is used to display information of a provider or member.
- read(): the read function is used to read and store data from a disk.
- update_info(): This function helps users to update information in the ChocAn data center if needed.

Moreover, the Model class will have other functions that support managing and updating data members for this class.

5.2 ChocAn Provider Class

Since the ChocAn system built an inheritance, the Provider class inherits the features from the base class, Model**.** The reason to keep the Provider class in a separate class is to help provider information more secure by managing their own information and services. Only providers can access their own data members.

5.2.1 Data Members of the Provider Class

What data members should we have for the Provider class? The Provider class is derived from the “Model” class, so it can access public and protected data members in the “Model” class. Furthermore, the Provider class should have their unique data members that include the number of their consultation, the weekly fee, and services’ information from the service list class. Service List is another subclass of the Provider class and contains data members that represent the name of the service, ID service, service cost, and the comments from providers.

```
class provider : public model
{
public:
        ...
private:
        int num_consul; // the number of consultations
        float weekly_fee;

        service_list * service_provided;
        ...

}
```

5.2.2 Functions of the Provider Class

Although the Provider class is derived from the Model class, it also has four main functions such as input(), display(), read(), and update_info(). These functions will specifically work on the provider's data, so we will add more identifying features as the

Provider class needs. Only the Provider class can access and edit certain information which belongs to the provider class. We have a function named display_summary() in which a provider can request to display an alphabetically ordered list of service names and corresponding service codes and fees.

```
class provider : public model
{
public:
        provider ();
        provider (const provider & copy); // copy constructor
        provider (string &, string &, string &, int, int, float);
        ~provider();
        void input();
        void display() const;
        void read() const;
        void update_info();

        void add_provider();
        void add_service_code();
        bool check_service_code(int service_code);
        void display_summary() const;
        void write_file() const;
        bool verify_providerID(int );

        bool operator<(const Provider & toCompare) const; // Overloaded < operator.
        bool operator>(const Provider & toCompare) const;
        bool operator==(const Provider & toCompare) const;

        friend ostream& operator<<(ostream & o, const Provider & toDisplay);
        friend bool operator ==(int Id_toCompare, const Provider & toCompare);
        ...

protected:
        int num_consul; // the number of consultations
        float total_fee; //weekly fee
        int provider_ID;
        service_list * service_provided; // services provided by providers
        ...

}
```

5.3 Member Class

Similarly, the Member class is also derived from the base class, Model. The Member class will be able to access data and function from the Model class. Moreover, it also contains their specific methods and data members.

5.3.1 Data Members of the Member Class

The Member class will have their unique data members including the status of member, the membership ID, and the membership fee. Because of inheriting from the Model class, the Member class will also have access to information about the address info, membership name, and email address.

```
class member : public model
{
public:
.       ...
private:
        string status_mem;
        int mem_ID; //  an ID number of a member
        float fee_mem;
        ...

}
```

5.3.2 Functions of the Member Class

The Member class will be responsible for their own data such as updating the status member, the verification of the membership ID, and the display of its weekly reports. However, the Manager class can access this Member class data in order to check the summary of the weekly report and also write this report to a file.

```
class member : public model
{
public:
        member();
        member(const member & copy); // copy constructor
        member (string , string , string, int);
        ~member ();
        void input();
        void display() const;
        void read() const;
        void update_info();

        bool verify_ID(string );
        bool update_status;
        void display_summary();

        void write_reports_toFile();

        //operator overload
        bool member::operator <(const member & toCompare) // sort the ID member

        ...

protected:
        string status_mem;
        int mem_ID; //  an ID number of a member
        float fee_mem;
        float overdue_fee;
        ...
}
```

5.4 Manager Class

The main purpose of the Manager class is designed to have the highest privilege in the ChocAn system, so managers can access and modify provider and member data in the ChocAn system.

```
class manager
{
public:
        manager();
        manager(const member & copy); // copy constructor
        ~manager ();
        // add members or providers
        void add_member();
        void add_provider();

        //remove members or providers by their ID
        bool remove_provider_byID(int );
        bool remove_member_byID(int );

        bool search_provider(string );
        bool search_member(string );

        void display() const;
        void read() const;
        void update_info();
        void retrieval();
        void write_report(); // Write the EFT data into a Disk
        ...

private:
        string manager_ID;
        BST_provider * manage_p;
        BST_member * manage_m;
        ...
}
```

5.4.1 Data Members of the Manager Class

The Manager class allows access to both the provider terminal and member terminal. The system has designed a concept of an object composition in OOP. The method would give managers permission to access data members and functions in both classes through an object. From that, managers could modify data of both terminals in the ChocAn data system.

5.4.2 Functions of the Manager Class

The ChocAn manager will have the ability to access Provider and member BST through their own terminal. Managers are able to add and remove providers and members through their own functions.

Additionally, we have a retrieval function to retrieve data from the provider terminal and member terminal. How does this function work? A provider will input a provider ID and that method belongs to the BST_provider class. This BST_provider will have a responsibility to search all the trees and verify the provider ID is valid or not. If it's valid, that means the provider exists. Then, we will copy the provider class containing the relevant data and return it. If not valid, an error message will be displayed to let the manager know. Similarly, we apply this method to a member to get member's information. Once the process completed, we could display and retrieve information to help in compiling weekly reports at the ChocAn Data Center. These reports will be written into a disk.

5.5. Service List Class

The service list class is a data abstraction, and it represents a service that providers consult to a member. Service objects represent the information about service provided which the provider provides to members as well as the member receives from the service provider. The service list objects are accessed in the hierarchy primarily.

```
class service_list
{
public:
        service_list();
        service_list(const service_list& copy); // copy constructor
        ~service_list ();

        void input();
        void display() const;
        void update_service_info();

        void print_report_provider();
        void print_report_member(); // print and write EFT report to a file
        bool verify_service_code();
        void update_comments();

        ...

protected:
        int serivce_code;
        float service_cost;
        string service_name;
        string comments;
        string date_service; //DOS  date of service
        string current_date;
        ...

}
```

5.5.1 Data Members of the Service List Class

The services list class will include the information such as service_code, service_cost, service_name, comment. These variables will be built in the protected section of the service list class to assure that they cannot be accessed from outside class.

5.5.2 Functions of the Service List Class

The public section of the service list class includes functions such as, input(), display(), update_service_info(); however, these methods will only be available to the Manager. Providers cannot use these methods, but they can add comments or request a Provider Directory. Service list class is invoked to submit data for a member report/provider report.

5.6 Data Structure

We will use the binary search tree to manage our data structure. Binary search trees are very useful in sorting data. When we implement a binary search tree, our elements can be compared in less than/ greater than the manner in any situation. ChocAn has a requirement that the provider can request a Provider Directory which is an alphabetically ordered list of service names, so a binary search tree can help solve this problem. Moreover, using a binary search tree helps the user to enter as many objects as they want, so the program doesn't limit the data that is entered by the user.

5.6.1 BST Class for Provider

5.6.1.1 Node Class for Provider

We will build a tree of providers, and each provider will be a node in the tree. We will manage the distribution of the nodes in the binary tree by the ID (integer), so the node will go to the left or the right depending on the ID of the provider. We will have a class named provider_node. The section protected provider_node class will have the left pointer of node and right pointer of node. We also have a provider_data pointer in the protected section of provider_node class, so we can set provider * provider_data to point to other functions in base class.

The section public provider_node class will have constructor, destructor, getter and setter functions, and get information from the provider class. The constructor function will help initialize a new object of node type. We will set the left pointer, right pointer, and provider_data pointer to NULL. Destructor function that can help deallocate memory and clean up for the node class and its class member when the object is destroyed. The private member variables that we cannot access, so we will use the getter function to retrieve the variable value and setter function to set the variable value. We

will use the getting information function to create and get information for a new provider from the user into a node.

5.6.1.2 BST_Provider Class for Managing Tree

We will create a class named BST_Provider. This class will have functions that can support adding data into the tree of providers. The protected section of BST_Provider class will have provider_node * root_p. The “root_p” pointer will point to the topmost node in the tree of the providers. The public section of BST_Provider will include constructor, destructor, copy constructor, add provider, remove provider, etc. These functions will help manage the data provider of the ChocAn system.

5.6.2 BST Class for Member

5.6.2.1 Node Class for Member

It is similar to the BST class for providers. We will build a tree of members, and each member will be a node in the trees. We will have a class named member_node. The protected section of member_node class will have the left pointer of node and right pointer of node. We also have a member_data pointer in the protected section of member_node class, so we can use member * member_data to point other functions in the base class.

The public section of member_node class will have constructor, destructor, getter and setter functions, and get information from the member class. The constructor function will help initialize a new object of node type. We will set the left pointer, right pointer and member_data pointer to NULL. Destructor function that can help deallocate memory and clean up for the node class and its class member when the object is destroyed. We cannot access the private member variables, so we will use the getter function to retrieve the variable value and setter function to set the variable value. We will use the getting information function to create and get information for a new provider from the user into a node.

5.6.1.2 BST_Member Class for Managing Tree

We will create a class named BST_Member. This class will have functions that can support adding data into the tree of members. The protected section of BST_Member class will have member_node * root_m. The “root_m” pointer will point to the topmost node in the tree of the member. The public section of BST_Member will include constructor, destructor, copy constructor, add member, remove member, etc. These functions will help manage the data members of the ChocAn system.

5.6.3 Sorting Data in Data Structure by a Request

Our data structure is a binary search tree, so it is very useful for sorting data in our system by request. We can use the Inorder tree traversal method to find data in sorted order.

1. **Provider:**

   We can search/sort data by provider ID easily. A Provider Directory is an alphabetically ordered list of service names and corresponding service codes and fees can be requested by the provider, and ChocAn system will support this requirement. By using binary search trees, it is not difficult to search and sort data by service names.

2. **Member:**

   We can search/sort data by member ID easily. The ChocAn system can support sorting in order of service date for members who have consulted a ChocAn provider during a week. Binary search trees help us solve this problem easily.