PROJECT TITLE - HEALTH INSURANCE MANAGEMENT SYSTEM

TEAM MEMBERS -

1. **NAYONIKA AGARWAL 590027512**
2. **PRINCE BISHT 590027473**

UNIVERSITY - UPES DEHRADUN, SCHOOL OF COMPUTER SCIENCE

PROFESSOR - VINOD KUMAR

COURSE - PROGRAMMING IN C

ABSTRACT

A C-programming-based program called the Health Insurance Management System was created to simplify and automate the administration of health insurance subscribers and their claims. The system offers administrators a simple navigational interface and focuses on effective record handling using file management techniques. By recording necessary information like name, age, insurance plan, and annual claim limit, the program enables the registration of new subscribers. Persistent record keeping is made possible by the automatic assignment of a unique ID to each subscriber and the safe storage of their data in a file.

Additionally, the system gives administrators a structured view of every subscriber who has registered, giving them full visibility into their annual and used claim limits. Claim processing, in which the program compares the requested claim amount to the subscriber's remaining annual limit, is a crucial feature. The system either fully approves, partially approves, or rejects the claim based on this computation. Accurate real-time tracking of insurance usage is ensured by writing the updated data back to the file following each claim transaction.

This project effectively shows how to use data management, conditional logic, file handling, and structures in C programming. By guaranteeing safe storage, precise limit tracking, and dependable claim processing, it mimics the fundamental functions of an actual health insurance system. The system functions as a basic model for advanced insurance management applications and is simple to use and scalable.

PROBLEM DEFINITION

Problem Statement:

Managing health insurance subscribers manually presents several challenges, including misplaced data, calculation errors, and difficulty tracking claims. Using registers or spreadsheets can make it time-consuming to access details, verify claim balances, and update records, thereby increasing the risk of errors in claim approvals. To improve the process, we need an automated system that can:

- Safely store subscriber information

- Generate unique IDs for each subscriber

- Track annual claim limits and used amounts

- Automatically approve, partially approve, or reject claims

- Update records while preserving past data

The aim is to provide quick access to subscriber information and ensure accurate claim processing with minimal manual effort.

Proposed Solution:

A Health Insurance Management System has been developed using the C programming language to address the above issues. This solution employs structures, file handling, and menu-driven programming to simulate the functionalities of a basic insurance system.

Some of our crucial features of the program include -

1. **Subscribers registrations & providing them a unique ID**
2. **Displaying all subscriber registrations in lists**
3. **Claim processing**
4. **File-based data storing**

SYSTEM DESIGN

The Health Insurance Management System is designed using a modular structure. This section explains the internal working of the system through architecture, algorithms, and flowcharts.

ARCHITECTURE

This program mainly consists of four sections -

a) **INPUT SECTION - collects subscribers' details and claims requests.**
b) **PROCESS SECTION - It generates a unique ID for each subscriber, performs claim validation, and calculates the remaining limits.**
c) **FILE - HANDLING SECTION- Stores and updates subscriber records in a permanent data.txt file**
d) **OUTPUT SECTION- Main menu, display subscribers, claim requests**

ALGORITHMS

Some of the algorithms for each essential function used in your code -

Algorithm 1: Adding Subscriber

Step 1: Read the next available ID from the file.
Step 2: Display ID to the user.
Step 3: Take input for name, age, plan, and annual claim limit.
Step 4: Set used amount = 0.
Step 5: Save subscriber details to a file using save_subscriber().
Step 6: Display "Subscriber saved successfully".

Algorithm 2: List Subscribers

Step 1: Open data file using fopen("r").
Step 2: Read each user record using fread().
Step 3: Store each record in an array.

Step 4: Display all fields in table format.
Step 5: Close the file.

Algorithm 3: Process Claim

Step 1: Load all subscribers from the file.
Step 2: Ask the user for a subscriber ID.
Step 3: Search for this ID in the array.
Step 4: If ID not found → Display error and STOP.
Step 5: Ask for a claim amount.
Step 6: Calculate remaining limit = Annual – Used.

Decision:

- **If claim ≤ remaining limit:**
  → Approve fully
  → Update used = used + claim

- **If claim > remaining limit AND remaining > 0:**
  → Partial approval
  → Covered = remaining
  → Subscriber pays extra

- **If remaining limit = 0:**
  → Reject claim

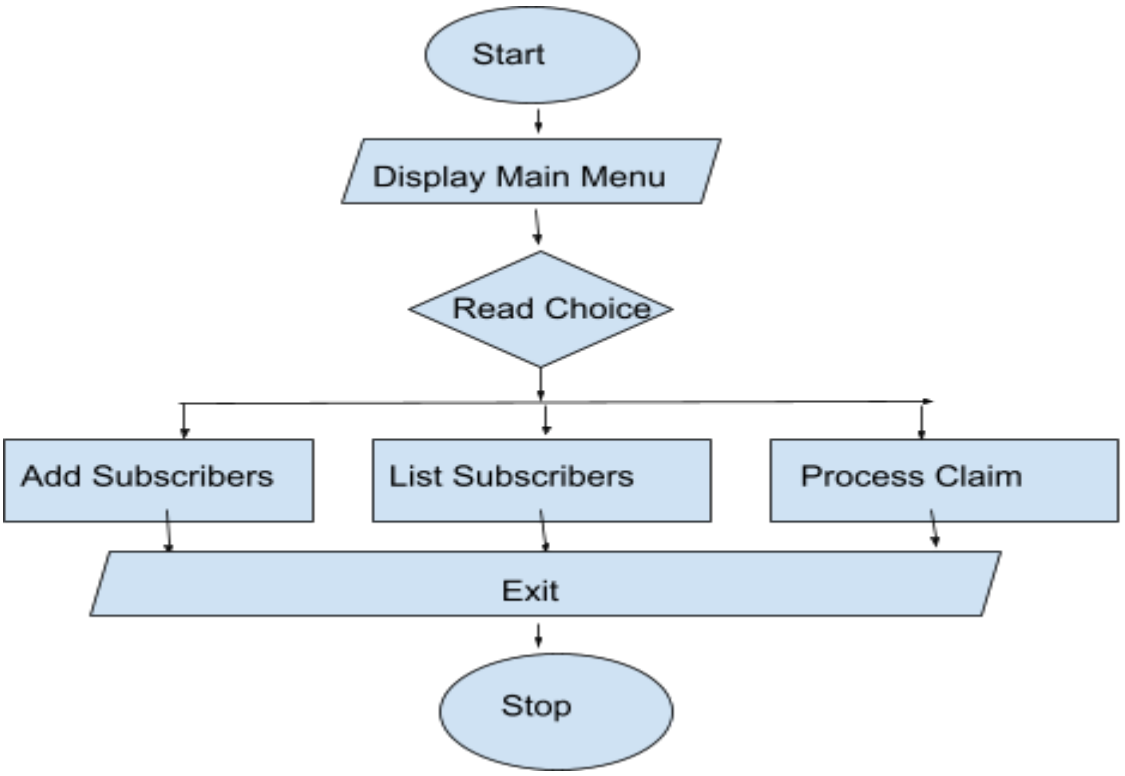
Step 7: Update the data file with the new used amount.

Algorithm 4: Get Next ID

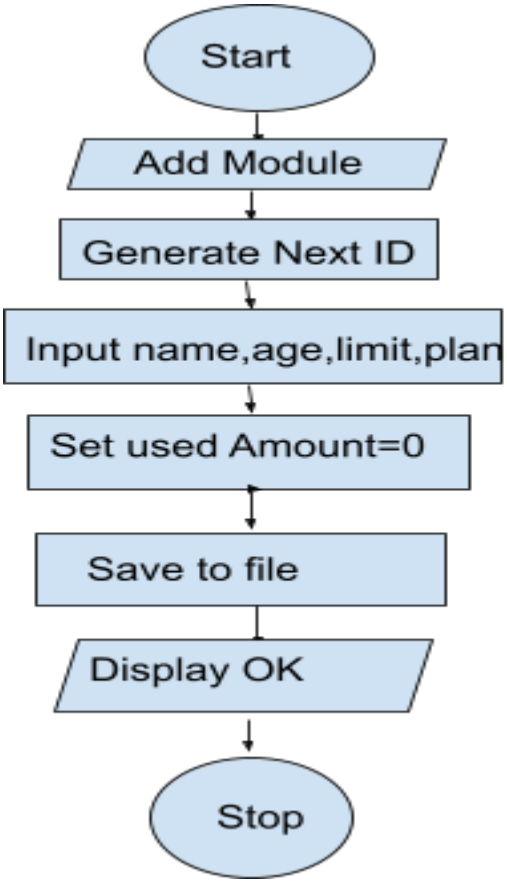
Step 1: Open the data file.
Step 2: Read each subscriber.
Step 3: Track maximum ID value.
Step 4: Return maxID + 1.
Step 5: If the file is empty → Return 1.

FLOWCHARTS

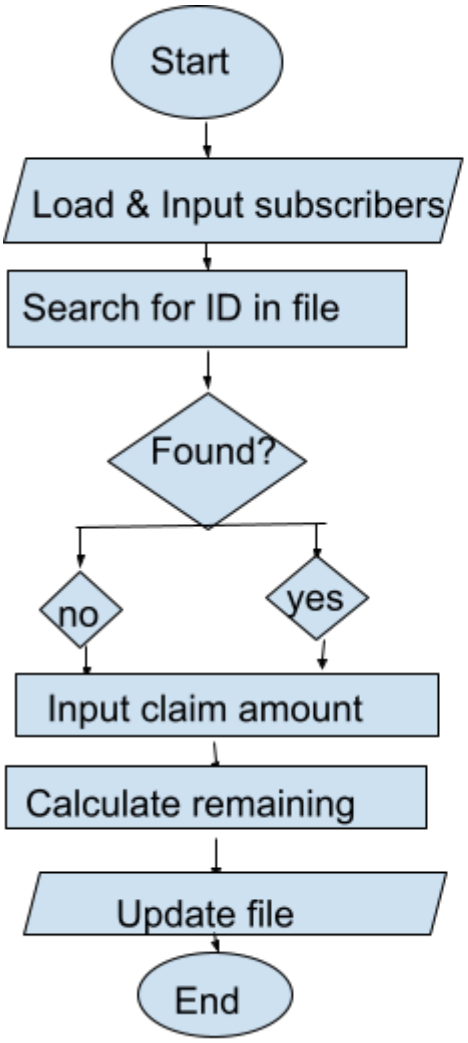
1)Flowchart 1 - MAIN MENU



2)Flowchart 2 - ADD SUBSCRIBERS



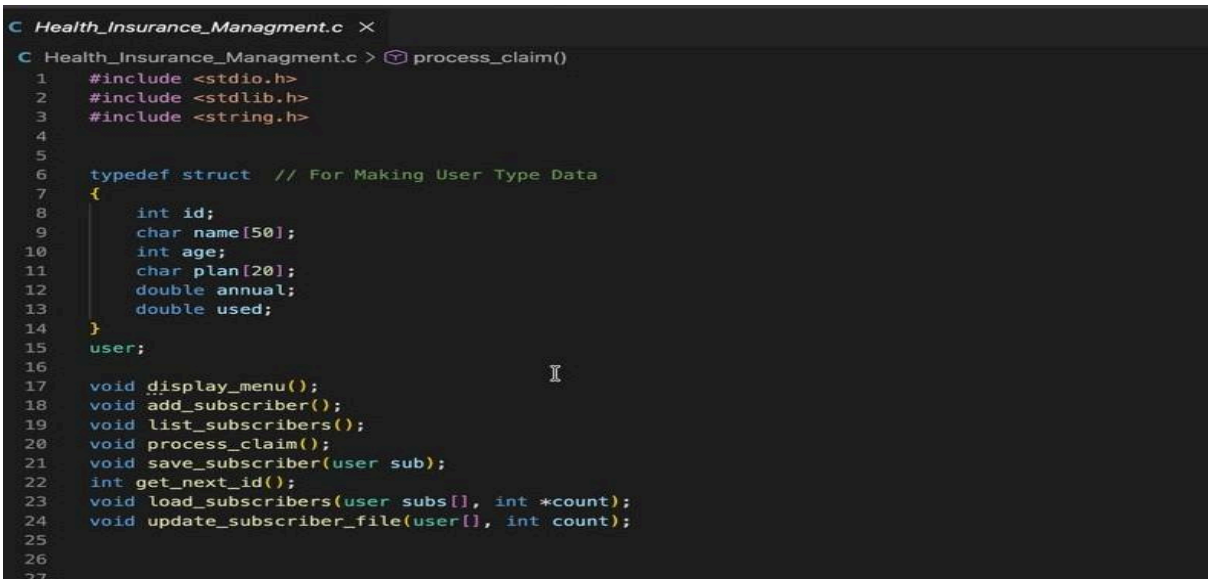
3)Process Claim



IMPLEMENTATION DETAILS

- **The system is implemented in the C programming language.**
- **The code is executed using a standard C compiler (GCC/MinGW/VS Code)**
- **All subscriber data is stored in a text file (Data.txt)**

DATA STRUCTURES- this structure stores subscriber names, ID, age, insurance plan, annual claim limit, and used claim amount.



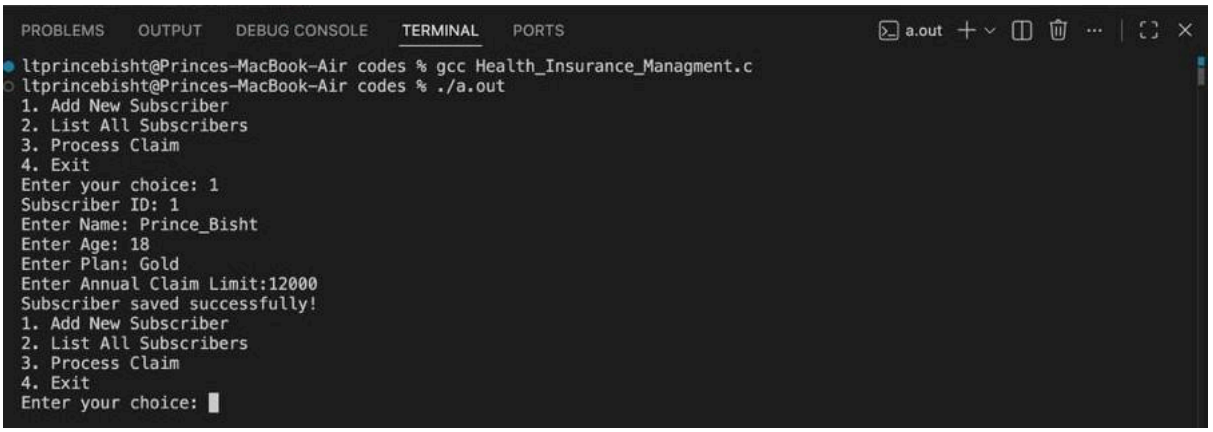
FILE HANDLING- The program performs persistent storage using the file Data.txt. This allows the program to store and retrieve subscriber data even after the program closes.

CONTROL FLOW- The main program uses a menu-driven loop:
1. Add New Subscriber
2. List Subscribers
3. Process Claim
4. Exit
A `do-while` loop ensures the user can keep performing operations until they choose EXIT.

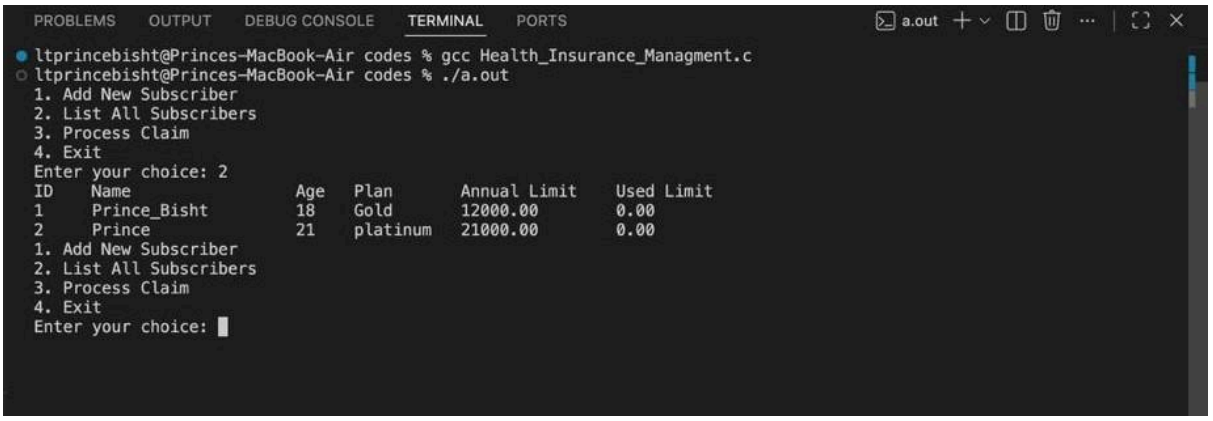
ERROR HANDLING - The program handles errors such as - File not found, Invalid input choice, Invalid claim amount (≤0), Claim exceeding annual limit, Non-existing subscriber ID.

TESTING & DETAILS

Test Case 1: Add Subscriber



Test Case 2: List Subscribers



Test Case 3: Process Claim

CODE OF THE PROJECT-

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


typedef struct  // For Making User Type Data
{
   int id;
   char name[50];
   int age;
   char plan[20];
   double annual;
   double used;
}
user;

void display_menu();
void add_subscriber();
void list_subscribers();
void process_claim();
void save_subscriber(user sub);
int get_next_id();
void load_subscribers(user subs[], int *count);
void update_subscriber_file(user[], int count);




int main() {
   int ch;

   do
   {


      printf("1. Add New Subscriber\n");
      printf("2. List All Subscribers\n");
      printf("3. Process Claim\n");
      printf("4. Exit\n");
      printf("Enter your choice: ");

      scanf("%d", &ch);


       switch (ch)
       {
          case 1:
             add_subscriber();
             break;
          case 2:
             list_subscribers();
```

```
            break;
        case 3:
            process_claim();
            break;
        case 4:
            printf("Exiting the Program!\n");
            break;
        default:
            printf("Invalid choice\n");
      }
   }
   while (ch != 4);


   return 0;
}




int get_next_id()  //To set user ID according to inputs
{
   FILE *data = fopen("Data.txt", "r");
   user sub;
   int max_id = 0;

   if (data == NULL)   //Null means there is no data in the file
   {
      return 1;
   }

   while (fread(&sub, sizeof(user), 1, data))
   {
      if (sub.id > max_id)
      {
         max_id = sub.id;
      }
   }

   fclose(data);
   return max_id + 1;
}


void save_subscriber(user sub) //For Saving New User Data
{
   FILE *data = fopen("Data.txt", "a");
   if (data == NULL)
   {
      printf("Error opening file for writing!\n");
      return;
   }
   fwrite(&sub, sizeof(user), 1, data);
```

```
    fclose(data);
    printf("Subscriber saved successfully!\n");
}


void load_subscribers(user subs[], int *count) // For Accesing Old User Data
{
    FILE *data = fopen("Data.txt", "r");
    user sub;
    *count = 0;

    if (data == NULL)
    {
        return;
    }

    while (fread(&sub, sizeof(user), 1, data))
    {
        if (*count < 100)
        {
            subs[*count] = sub;
            (*count)++;
        }
    }
    fclose(data);
}


void update_subscriber_file(user subs[], int count) //For Updating File With new User
Data
{
    FILE *data = fopen("Data.txt", "w");
    if (data == NULL)
    {
        printf("Error opening file for updating!\n");
        return;
    }
    fwrite(subs, sizeof(user), count, data);
    fclose(data);
}



void add_subscriber() // For Getting New User Data
{
    user new_sub;

    new_sub.id = get_next_id(); // Will Update according to Data entry number

    printf("Subscriber ID: %d\n", new_sub.id);

    printf("Enter Name: ");
    scanf("%s", new_sub.name);
```

```
    printf("Enter Age: ");
    scanf("%d", &new_sub.age);

    printf("Enter Plan: ");
    scanf(" %19s", new_sub.plan);

    printf("Enter Annual Claim Limit:");
    scanf("%lf", &new_sub.annual);

    new_sub.used = 0.0;

    save_subscriber(new_sub);
}

void list_subscribers() //For Printing all the users Data
{
    user subs[100];
    int count = 0;

    load_subscribers(subs, &count);

    if (count == 0)
    {
        printf("No subscribers found\n");
        return;
    }


    printf("%-5s %-20s %-5s %-10s %-15s %s\n",
           "ID", "Name", "Age", "Plan", "Annual Limit", "Used Limit");

    for (int i = 0; i < count; i++)
    {
        printf("%-5d %-20s %-5d %-10s %-15.2lf %.2lf\n",
               subs[i].id,
               subs[i].name,
               subs[i].age,
               subs[i].plan,
               subs[i].annual,
               subs[i].used);
    }
}

void process_claim() //To claim the process
{
    int id;
    double claim_amount;
    user subs[100];
    int count = 0;
    int found_index = -1;

    load_subscribers(subs, &count);
```

```
   printf("Enter Subscriber ID: ");
   scanf("%d", &id);


   for (int i = 0; i < count; i++)
   {
      if (subs[i].id == id)
      {
         found_index = i;
         break;
      }
   }

   if (found_index == -1)  // For Wrong ID or non-Existing ID
   {
      printf("Subscriber with ID %d not found.", id);
      return;
   }

   printf("Subscriber: %s (ID: %d)\n", subs[found_index].name,
subs[found_index].id);
   printf("Current Used Limit: $%.2lf\n", subs[found_index].used);
   printf("Remaining Limit: $%.2lf\n", subs[found_index].annual -
subs[found_index].used);

   printf("Enter Claim Amount: $");
   scanf("%lf", &claim_amount);


   double remaining_limit = subs[found_index].annual - subs[found_index].used;

   if (claim_amount <= 0)
   {
      printf("Claim amount must be positive.\n");
   }
   else if (claim_amount <= remaining_limit)
   {

      subs[found_index].used += claim_amount;
      printf("Claim APPROVED for $ %.2lf\n", claim_amount);
      printf("New Used Limit: $%.2lf\n", subs[found_index].used);
   }
   else
   {

      double covered_amount = remaining_limit;
      double subscriber_payment = claim_amount - remaining_limit;

      if (remaining_limit > 0)
      {
```

```
        subs[found_index].used = subs[found_index].annual;
        printf("Claim Partially Approved\n");
        printf("Covered Amount (Paid by Insurance): $%.2lf\n", covered_amount);
        printf("Subscriber's Responsibility (Must Pay): $%.2lf\n", subscriber_payment);
      }
      else
      {

        printf("Claim REJECTED: Annual limit of $%.2lf has been exhausted.\n",
subs[found_index].annual);
        printf("Subscriber's Responsibility (Must Pay): $%.2lf\n", claim_amount);
      }
    }


    update_subscriber_file(subs, count); //To update the procces Amount of users

}
```