LHospital

A *Turbo* C++ project

A basic management system for a general hospital

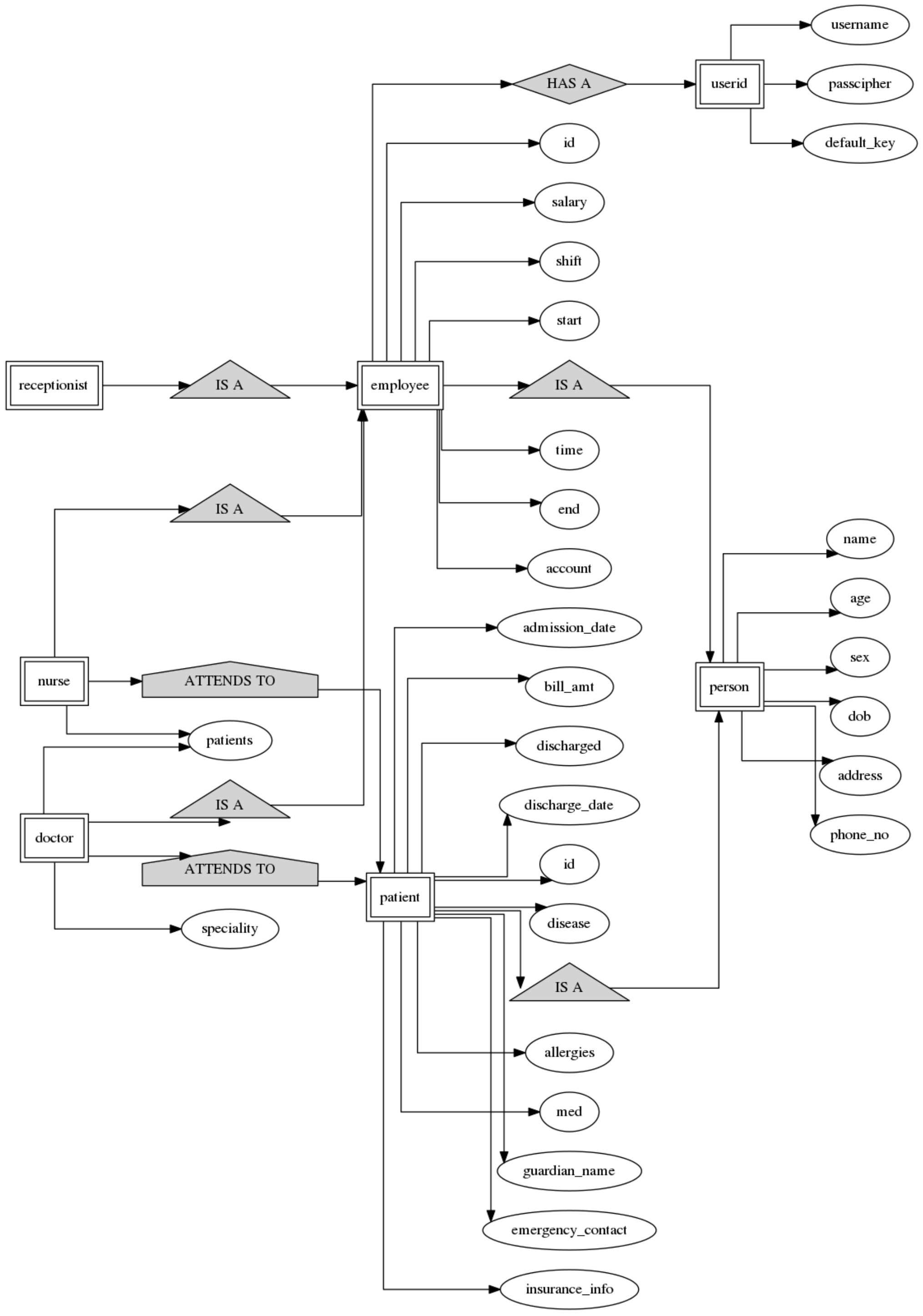
Arpit Saxena 9151996
Anirudh Panigrahi 9151993
Sankalp Gambhir 9152014

Contents

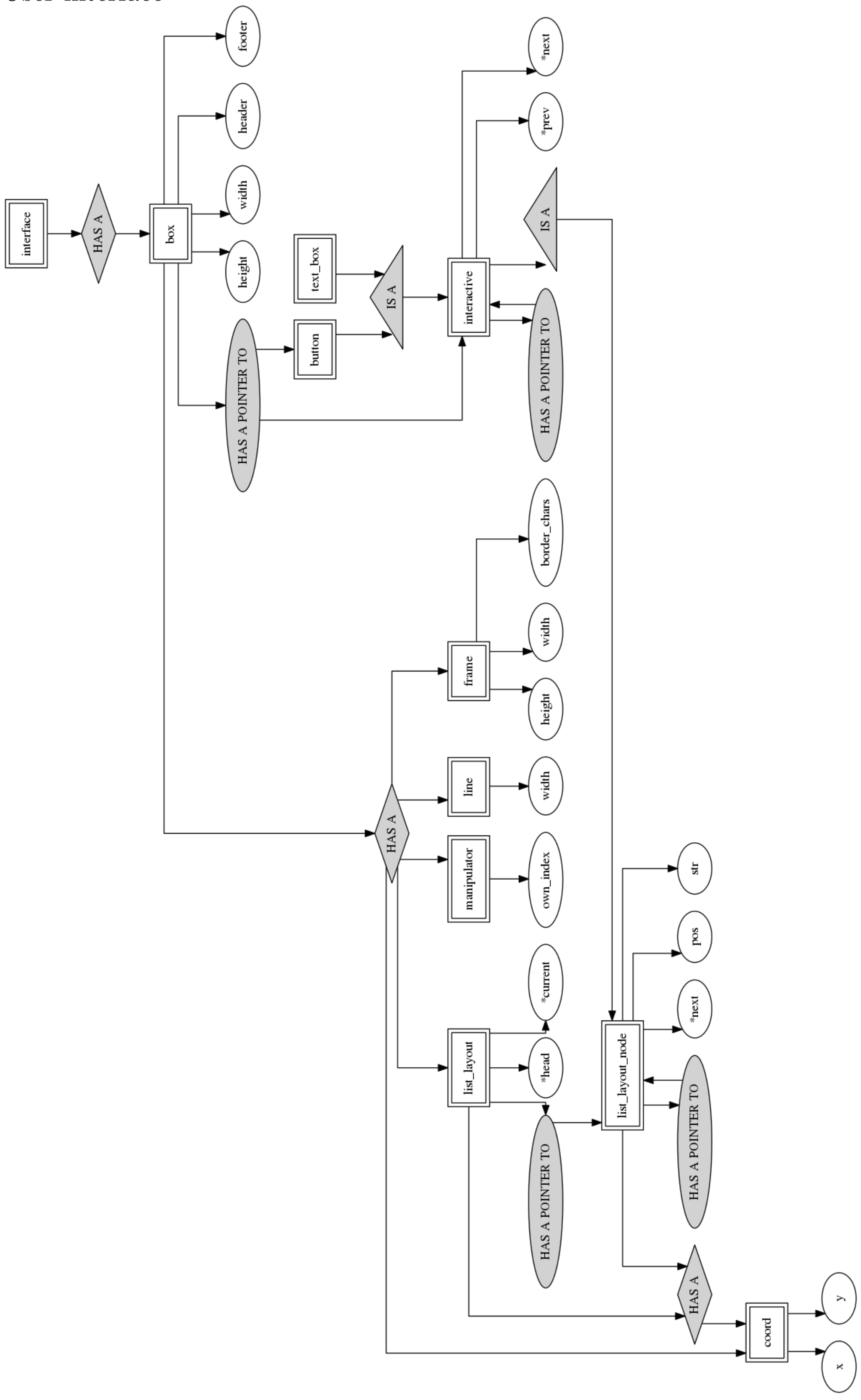
Diagrams

ER diagrams

1. Hospital

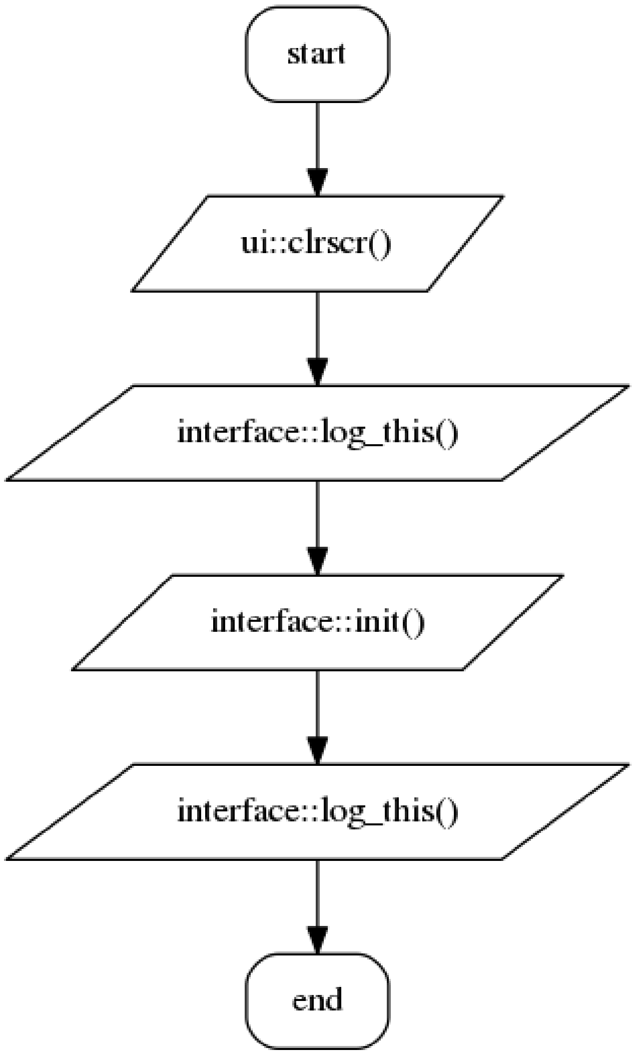


2. User interface



Note: The figure included has been rotated

Flowchart of main()



Source Code

Header files

1. code/iface.hpp

```
/*!
 \file iface.hpp
 \brief Contains prototypes of the functions managing the interface of the
     program
*/

#ifndef INTERFACE_HPP
#define INTERFACE_HPP

#include "ui/ui.hpp"

//!Class containing all the functions that make up the interface of the program
class interface{
    public:
        static void init(); //!>The main interface function of the program; this
            is the functions that runs throughout the program
        static int login_screen();//!>Login screen interface
        static int menu();//!>The first main menu interface(For administrator
            employee only)
        static void patient_management(); //!>Patient management interface(for
            receptionist employees only)
        static void employee_management(); //!>Employee management interface(for
            administrator employees only)
        static void stock_management(); //!>Stock management interface(for
            administrator employees only)

        static void employee_screen(unsigned long); //!>The interface for non-
            administrator employees

        static void error(char*); //!>Prints an error message at the footer of
            interface::window
        static void clear_error(); //!>Clears the error message at the footer of
            interface::window

        static int log_this(char *); //!>Logs a message string into a file log.
            txt

    protected:
        interface(); //!>Objects of this class shouldn't be created
        /*!
        For creating a validation function to use in menus
        to validate the choice input of the menu option to be accessed
        */
        class validate_menu
        {
                static int lowest_choice, greatest_choice; //!>The lower and
                    upper limit of the choices of a menu
                validate_menu(); //!>Objects of this class shouldn't be created
            public:
                static int input(const char *); //!>The validation function that
                    will be passed as an argument to box::operator>>()
                static void set_menu_limits(int, int); //!>Setter; sets
                    lowest_choice and greatest_choice
        };
        /*!Creates a back_func that can be passed as an argument to box::
            setback_func()*/
        class back_func
        {
                back_func(); //!>Objects of this class shouldn't be created
            public:
                static int backbit; //!>1, if shift + bkspc is pressed, 0
                    otherwise
                static int set_backbit(); //!>Setter, passed as an argument to
                    box::setback_func()
        };
        static box window; //!>The main outer window box
};

//!Class containing all the functions that make up the interface of Employee
    management
class emp_mgmt : public interface
{
    public:
        static void view_emp(); //!>Interface of View Employee
        static int view_emp(unsigned long); //!>Creates the interface that shows
            the details of an employee with a particular ID
        static void add_emp(); //!>Interface of Add Employee
        static void remove_emp(); //!>Interface of Remove Employee
        static void edit_emp(); //!>Interface of Edit Employee
        static void pay_emp(); //!>Interface of Pay Employee
        static void pay_all(); //!>Interface of Pay All Employees
    private:
        emp_mgmt(); //!>Objects of this class shouldn't be created
};

#endif /* INTERFACE_HPP */
```

2. code/EMP.HPP

```
/*!
 \file EMP.HPP
 \brief Contains the definitions of the employee class and its derivatives
*/

#ifndef EMP
#define EMP

#include "base.hpp"

enum emp_type {INVALID, OTHERS, DOCTOR, NURSE, RECEPTIONIST};
//!>Identifiers for indication of different types of employees

//!Class storing details of employees of the hospital
class employee : public person{
        int generate_id(); //!>Generates ID of the employee
        static int generate_id_status; //!>0 if the last id generation was
            unsuccessful
        /*!>Basically ensures that id generation is stopped when an error occurs
            in id generation,
        otherwise, the files(max_id.dat, id_list.dat) might start storing
            meaningless data, which
        will affect future id generation*/

    public:
        employee(str, int, Date, address, phone, unsigned long, Time, Time, str =
             "", str = ""); //!>Explicit constructor
        /*!>for all those with user accounts(doctors, nurses, receptionists),
        last 2 arguments are to be provided as well*/
        employee(); //!>Default constructor

        //!@{Getters
        int get_age(); //!>Overridden function
        /*!>Updates the age of the employee and writes the employee object back
            to file before returning age*/
        unsigned long get_salary();
        Time get_shift(int inp1); /*!>\param inp1 times_of type variable that
            indicates starting or ending shift time*/
        unsigned long get_id();
        static int get_generate_id_status();
        transaction * get_last_5_transactions(); //!>Gets the last 5 records
            present in the file TRANS.DAT of the employee's folder
        //!}@

        //!@{Setters
        void set_salary(unsigned long);
        void set_shift(int inp1, Time t1);/*!>\param inp1 times_of type variable
            that indicates starting or ending shift time*/
        //!}@

        userid account; //!>Facilitates login mechanism of the employee
    protected:
        unsigned long id; //!>ID of the employee
        unsigned long salary; //!>Salary of the employee
        Time shift_start; //!>Starting shift time of the employee
        Time shift_end; //!>Ending shift time of the employee
};

//!Class storing details of doctors of the hospital
class doctor : public employee{
    public:
        doctor(str, int, Date, address, phone, unsigned long, Time, Time, int,
            int, str, str); //!>Explicit constructor
        doctor(); //!>Default constructor

        //!@{Getters
        int * get_speciality();
        long * get_patients();
        //!}@

        //!@{Setters
        void set_speciality(int *);
        void set_patients(long *);
        //!}@

    private:
        int speciality[2]; //!>Doctor's specialization
        long patients[10]; //!>Patients currently under care, can take only 10 at
             once
```

```
};

//!Class storing details of nurses of the hospital
class nurse : public employee{
    public:
        nurse(str, int, Date, address, phone, unsigned long, Time, Time, str, str
            ); //!>Explicit constructor
        nurse(); //!>Default constructor
        long * get_patients(); //!>Getter

        void set_patients(long *); //!>Setter
    private:
        long patients[5]; //!>Patients currently under care, can take only 5 at
            once
};

//!Class storing details of receptionists of the hospital
class receptionist : public employee
{
    public:
        receptionist(str, int, Date, address, phone, unsigned long, Time, Time,
            str, str); //!>Explicit constructor
        receptionist(); //!>Default constructor
};

//!Class that generates objects storing the employee type corresponding to each
    id
/*!
This class is used to generate objects storing the employee type corresponding to
     each id,
and then to store these objects to a file EMPLOYEE/ID_LIST.DAT(The ctor itself
    does all this)
This class is used to get the employee type of an employee having a particular id
*/
class id_to_emp
{
        unsigned long id;//!>ID of employee
        int employee_type;//!>Type of employee
    public:
        int status; //!>True whenever the constructor runs successfully and
            succeeds in storing the object to id_list.dat
        id_to_emp(unsigned long, int); //!>Explicit constructor
        id_to_emp(); //!>Default constructor
        static int convert(unsigned long); //!>Converts id to employee type
};

#endif
```

3. code/BASE.HPP

```
/*!
 \file BASE.HPP
 \brief Contains the declarations of the basic structs, classes, typedefs
        and enums to be used in the whole program
*/

#ifndef BASE
#define BASE

```

```
#include "ui/ui.hpp"
#include <fstream.h>
#include <string.h>
#include <dir.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <time.h>
#include <stdlib.h>         //for random() and randomize()

typedef char str[80]; //!>typedef for general strings
typedef char phone[11]; //!>typedef for strings storing phone numbers

enum sex {MALE, FEMALE, TRANS}; //!>Identifiers for different sexes
enum date_type {DAY, MONTH, YEAR}; //!>Identifiers for different parts of a date
enum time_type {HOUR, MINUTE, SECOND}; //!>Identifiers for different parts of a
    time
enum body_parts {BRAIN, HEART, SKIN,
    LUNG, BONE, EYE,
    THROAT, TEETH, STOMACH,
    BLOOD, GUT, GEN}; //!>Identifiers for different parts of the human body
/*!>used for recording specialities of doctors(GEN for general problems)*/

enum address_parts {HOUSE_NO, STREET, CITY, DISTRICT, STATE}; //!>Identifiers for
     different parts of an address
enum times_of {START, END}; //!>Identifiers indicating start or end of something
/*!>(used in get_shift() and set_shift() to get or set starting or ending shift
    time)*/

struct Time{
    unsigned int hour;
    unsigned int minute;
    unsigned int second;

    Time();
    Time(unsigned h, unsigned m, unsigned s);
};//!>Structure facilitating implementation of a time variable

struct Date{
    unsigned int day;
    unsigned int month;
    unsigned int year;

    Date();
    Date(unsigned d, unsigned m, unsigned y);
};//!>Structure facilitating implementation of a date variable

class system
{
    private:
        system();
    public:
        static Date get_date();
        static Time get_time();
};//!>Contains prototypes of functions that return the system date and time

struct address{
    str house_no;
    str street;
    str city;
    str district;
    str state;

    address(const char * = "", const char * = "", const char * = "", const char *
         = "", const char * = "");
};//!>Structure facilitating implementation of an address variable

struct disease{
    str name;
    int type;            //refers to body part affected (LUNG, HEART, etc)
    str symptoms[4];     //symptoms reported by patient
};//!>Structure facilitating implementation a variable storing details of a
    disease

struct insurance{
    str provider;
    unsigned long amount;
    Date expiry;
};//!>Structure facilitating implementation a variable storing insurance details
    of any person

struct medicine{
    int code;
    float price;
    str name;
    float dosage;
    long stock;
};//!>Structure facilitating implementation a variable storing details of a
    medicine

struct transaction{
    float amount;
    str reason;
    Date _date;
    Time _time;
    transaction(float, Date = Date(), Time = Time(), char* = "NA");
    transaction();
};//!>Structure facilitating implementation a variable storing details of a
    transaction

struct procedure{
    str name;
    float cost;
};//!>Structure facilitating implementation a variable storing details of a
    medical procedure

//!Class storing all common data members of a person
/*!
Parent class to all the persons that this program handles, i.e patients,
and all types of employees.
*/
class person{
    public:
        person(str, int, Date, address, phone); //!>Explicit constructor
        person(); //!>Default constructor

        //!@{Getters
        char* get_name();
        int get_age();
        int get_sex();
        Date get_dob();
        address get_address();
        char* get_phone();
        //!}@

        //!@{Setters
        void set_name(char*);
        void set_sex(int);
        void set_dob(Date, Date = system::get_date());
        void set_address(address);
        void set_phone(char*);
        //!}@

    protected:
        str name; //!>Name of the person
        unsigned age; //!>Age of the person
        unsigned sex; //!>Sex of the person
        Date dob; //!>Date of birth of the person
        address adr; //!>Address of the person
        phone phone_no; //!>Phone number of the person

    private:
        void calc_age(Date d = system::get_date()); //!>Calculates age of the person using dob
        /*!>\param d The date with respect to which age is to be calculated(default value is set to be the system date)*/
};

//!Class managing login features of the program
/*!
This class stores a username and a password in encrypted form, besides
the inplementation data. This class uses a vigenere cipher to encrypt the
password and store it.
*/
class userid
{
        str username; //!>Username of the login account
        str passcipher; //!>Encrypted password
        str default_key;//!>Key for making the vigenere cipher
        void makecipher(char *); //!>Makes the vigenere cipher
        void set_key(char *); //!>Sets default_key to a random string
        char * decipher(); //!>deciphers the cipher 'passcipher'

    public:
        userid(char *, char *); //!>Explicit constructor
        userid(); //!>Default constructor
        char * get_username(); //!>Getter
        void set_username(char *); //!>Setter
        int login(char *); //!>\return 1 if the string input in the function is the password, 0 otherwise
};

//!Defines << operator overloads to facilitate printing of some stuff
class enum_to_str
{
        enum_to_str();
    public:
        friend box & operator<<(box &output, sex s);        //!>converts sex enumeration constant into a string and prints it to a box
        friend box & operator<<(box &output, body_parts b); //!>converts body_parts enumeration constant into a string and prints it to a box
        friend box & operator<<(box &output, Time & t);     //!>converts Time variable into a string and prints it to a box
        friend box & operator<<(box &output, Date & d);     //!>converts Date variable into a string and prints it to a box
        friend box & operator<<(box &output, address & a);  //!>converts address variable into a string and prints it to a box
};

#endif
```

4. code/PATIENT.HPP

```
/*!
 \file PATIENT.HPP
 \brief Contains the patient class definition
*/

#ifndef PATIENT
#define PATIENT

#include "base.hpp"

class patient : public person
{
    protected:
        long id;
        disease dis;        //patient's afflictions
        str allergies[2];   //patient's known allergies
        int med[50][2];     //patient's purchased meds & quantities
        str guardian_name;
        str emergency_contact;
        phone emer_contact_no;
        insurance insur_info;
        Date admission_date;
        unsigned long bill_amt;
        int discharged;
        Date discharge_date;
    public:
        patient(str, int, Date, address, phone, disease, str, str, phone, insurance, Date = system::get_date());    //if date_of_admission is the current system date, last argument is not needed
        patient();  // Default constructor
        //'get's
        long get_id();
        disease get_dis();
        char* get_guardian_name();
        char* get_emergency_contact();
        char* get_emer_contact_no();
        insurance get_insur_info();
        int get_admission_date(int);
        unsigned long get_bill_amt();
        int get_med(int, int);
        int get_discharge_date(int);
        transaction get_transaction(int);
        transaction get_transaction();

        //updating functions
        void set_dis(disease);
        void set_guardian_name(char*);
        void set_emergency_contact(char*);
        void set_emer_contact_no(char*);
        void set_insur_info(insurance);
        void set_admission_date(Date);
        void set_bill_amt(unsigned long);
        void set_med(int, int, int);
        void set_discharge_date(Date);
        void discharge();
};

#endif
```

5. code/HOSP.HPP

```
/*!
 \file HOSP.HPP
 \brief Contains prototypes of the hospital management functions
*/

#ifndef HOSP
#define HOSP

#include "base.hpp"
#include "patient.hpp"

//!Stores the no. of days in each month of the year(for hospital::
    get_date_difference())
const int monthDays[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
const int stay_charge = 50;  //!> The charge per day of stay in the hospital($50
    per day)

//!Class containing all the basic hospital management functions used in the
    program
class hospital
{
    public:
        //!@{Hospital finances management functions
        static float get_balance();     //!<Getter function

        //!Deducts the input amount from hospital::balance
        /*!
        \param amt The amount to be deducted from hospital::balance
        \param reason The reason for deduction of money
        \param dt The date of deduction of money
        \param tm The time of deduction of money
        \return A transaction type variable containing details about the amount
            deduction
        */
        static transaction deduct_money(float amt, char* reason, Date dt, Time tm
            );

        //!Adds the input amount to hospital::balance
        /*!
        \param amt The amount to be added to hospital::balance
        \param reason The reason for deposit
        \param dt The date of deposit
        \param tm The time of deposit
        \return A transaction type variable containing details about the deposit
        */
        static transaction add_money(float, char*, Date, Time);

        //!Returns the last 10 transactions of the hospital
        /*!
        Reads the last 10 records from a file transactions.dat
        \return An array of type transaction containing those 10 records
        */
        static transaction* get_transaction();
        //!}@

        //!@{Patient management functions

        //!Gets a patient object from file corresponding to the inputted id
        /*!
        Reads a patient object from a file base.dat that is present in
        a folder with name as the id of the patient. All such folders are
        present in a folder named PATIENT
        \param id The id of the patient to be read
        \return The patient object read from file
        */
        static patient get_patient_by_id(long id);

        //!Writes a patient object to file
        /*!
        Makes a new folder(if it doesn't exist) in PATIENT, whose name is the
        id of the patient object that is to be written, and writes the patient
        object to a file BASE.DAT inside that folder
        \param a The patient object that is to be written to file
        */
        static void write_patient(patient a);

        //!Charges a patient for any service, treatment etc. that the patient
            availed from the hospital
        /*!
        Gets the patient object from file using get_patient_by_id(), and adds a
        transaction type variable to a file TRANS.DAT that is present in the
            folder
        of the patient(i.e the folder having name as the id of the patient)
        \param pat_id ID of the patient
        \param trans The transaction type variable containing details of the
            transaction that is charged to the patient
        */
        static void charge_patient(int pat_id, transaction trans);

        //!Discharges a patient from the hospital
        /*!
        Calls the patient::discharge() function, sets the patient's discharge
        date to the current system date, and writes the patient back to file
        \param temp The object of the patient who is to be discharged
        */
        static void discharge_patient(patient temp);
        static float calc_bill(int); //!>Calculates the cost of the patient's
            stay in the hospital
        //!}@

        //!@{Functions for medicine records management

        //!Gets a medicine from file corresponding to an input code
        /*!
        Reads a medicine type variable from a file STOCK/MED.DAT that has code equal to the
        code inputted to the function
        \param inp_code The code of the required medicine
        \return The medicine corresponding to inp_code
        */
        static medicine get_med_by_code(int inp_code);

        //!Writes a medicine to file
        /*!
        Writes a medicine type variable back to the file STOCK/MED.DAT which stores all
        medicines, after it has been edited, into its position in the file.
        */
        static void write_med(medicine);
        //!}@

        //!@{Employee management functions

        //!Gets an object of an employee(or its derivative) from file corresponding to an input id
        /*!
        Reads an object from a file base.dat that is present in
        a folder with name as the id of the employee. All such folders are
        present in a folder named EMPLOYEE, or EMPLOYEE/DOCTOR, or
        EMPLOYEE/NURSE, or EMPLOYEE/RECEPTIONIST
        This function converts the id inputted to it into the
        employee type, sets the string of the path to the folder containing
        the employee object file, and reads the object from the file to a
        buffer pointed to by a void pointer
        A void pointer is used in this function to handle the different data types
        i.e. employee, doctor, nurse, receptionist that can be input into the
        function as the target parameter.
        \param id The id of the employee to be read
        \param target void pointer pointing to the buffer that stores the object
        \return 1 if the function executed without errors, 0 otherwise
        */
        static int get_employee_by_id(unsigned long id, void * target);

        //!Writes an object of type employee(or its derivative) to file
        /*!
        This function converts the id of the object pointed to by the pointer
        a into the employee type, sets the string of the path to the folder that should
        contain the employee object file, and then makes a new folder(if it doesn't exist)
        in EMPLOYEE, or EMPLOYEE/DOCTOR, or EMPLOYEE/NURSE, or EMPLOYEE/RECEPTIONIST
        (depending upon the type of the object pointed by the input void pointer), whose
        name is the id of the employee object that is to be written, and writes the
        employee object to a file BASE.DAT inside that folder.
        A void pointer is used in this function to handle the different data
```

```
            types
        i.e. employee, doctor, nurse, receptionist that can be input into the
        function as the parameter a.
        \param a void pointer pointing to the object that is to be written to file
        \return 1 if the function executed without errors, 0 otherwise
        */
        static int write_employee(void * a);

        //!Pays salary to an employee having a particular id
        /*!
        Obtains the employee’s object from file using get_employee_by_id(),
        gets the salary of the employee, deducts the salary from hospital::balance
        using hospital::deduct_money(), and then writes the transaction details of
        this payment to a file TRANS.DAT present in the folder of the employee (i.e
        the folder having name as the id of the employee), that is present in folder
        EMPLOYEE, or EMPLOYEE/DOCTOR, or EMPLOYEE/NURSE, or EMPLOYEE/RECEPTIONIST,
        depending on the employee type.
        \param id ID of the employee to whom salary is to be paid
        \param d1 Date of payment of salary
        \param t1 Time of payment of salary
        \return 1 if the function executed without errors, 0 otherwise
        */
        static int pay_salary(unsigned long id, Date d1, Time t1);

        //!Pays salary to all employees whose files are on the disk
        /*!
        Loops the execution of pay_salary(); the maximum no. of times the loop should
        run is determined by max_id, an unsigned long variable stored in a file
        EMPLOYEE/MAX_ID.DAT
        \return 1 if the function executed without errors, 0 otherwise
        */
        static int pay_all_salaries();
        //!}@

        //!@{Internal implementation functions
        static int get_date_difference(Date, Date); //!>Calculates the no. of days between 2 dates
        static int count_leap_years(Date); //!>Calculates the no. of leap years between a certain date and the year 0 AD
        static int date_validity(const char * inp_date); //!>\return date_validity(str_to_date(inp_date))
        static int date_validity(Date); //!>\return 1 if the date supplied to the function is a valid date, 0 otherwise
        static int time_validity(const char * inp_time); //!>\return time_validity(str_to_time(inp_time))
        static int time_validity(Time); //!>\return 1 if the time supplied to the function is a valid time, 0 otherwise
        static Date str_to_date(const char *); //!>converts a string to a Date type variable
        static Time str_to_time(const char *); //!>converts a string to a Time type variable
        static int str_to_sex(char *); //!>converts a string to an enum sex type
```

```
            variable
        //!}@

    private:
        hospital();//!>Objects of this class shouldn't be created

        //! Reads an object from a file on disk and stores it in a buffer
        /*!
        A specific implementation of the fstream::read() function for the
            hospital's
        purposes.
        \param ID ID of the object that is to be read(for error logging purposes
            only)
        \param dest The path string to the file from which the object is to be
            read
        \param size Size in bytes of the object that is to be read
        \param temp Pointer to the buffer at which the read object is to be
            stored
        */
        static int read_from(unsigned long ID, char * dest, int size, char *temp)
            ;
        static double balance; //!>Current balance of the hospital
};

#endif
```

6. code/UI/test.hpp

```
#ifndef TEST_HPP
#define TEST_HPP

void test_weird_error();

int back_func();
void test_back();

void test_all();
void test_listlayout();
void test_textbox();
void test_frame();

#endif /* TEST_HPP */
```

7. code/UI/ui.hpp

```
/*!
 \file ui.hpp
 \brief Contains prototypes of UI functions
*/

#ifndef UI_HPP
#define UI_HPP

#include <conio.h>
#include <stdarg.h>
#include <string.h>
#include <stdio.h>
#include <iostream.h>
#include <ctype.h>
```

```
#include <stdlib.h>
#include <limits.h>
#include <errno.h>
#include <new.h>
#include <process.h>

//! Validator function that's used for validating user input
typedef int (*validator_f)(const char *);

//! For running ui::init() before main (initialising basic stuff)
class init_lib_ui
{
    static int counter; //!< Ensures ui::init() is called only once
    public:
        init_lib_ui(); //!< Ctor
};

//! Static object of type init_lib_ui that is initialised
//! before main is run and thus, ui::init is called
static init_lib_ui init_obj_ui;

//! Manipulator class to manipulate UI functions
/*!
 Objects of this type would be used instead of an enum
 to avoid conflicts with int
 Every manipulator object is identified by its index while
 static index indicates the index to be assigned to the next
 manipulator
*/
class manipulator
{
    static int index; //!< index of a new manipulator object
    int own_index;    //!< index of current manipulator

    public:
        manipulator(); //!< Ctor; assigns index
        int operator==(manipulator); //!< Returns 1 if indexes are same
};

//! Class containing basic UI functions and attributes
class ui
{
    ui();   //!< Private ctor; object of this class shouldn't be created
    public:

        //! Specifies the directions for modifying frame, etc.
        enum dir
        {
            left = 1,
            top = 2,
            right = 4,
            bottom = 8,
            all = 16 //!< When all sides need to be modified
        };
        static int scr_height; //!< Height of screen
        static int scr_width; //!< Width of screen
        static void init(); //!< Sets all static variables
        static void clrscr(); //!< Clears the contents off the screen
        static int tcolor; //!< text color
        static int bcolor; //!< background color
        static manipulator endl; //!< End line and move cursor to next line
        static manipulator centeralign; //!< Center align
        static manipulator rightalign; //!< Right align

        //! This func is called when new is unable to allocate memory
        static void my_new_handler();
};

//! Represents a coordinate
struct coord
{
    int x;  //!< x coordinate
    int y;  //!< y coordinate

    coord(int = 1,int = 1); //!< Sets the coordinate
    coord & operator+=(coord);
    coord & operator-=(coord);
    coord operator+(coord);
    coord operator-(coord);
};

//! Represents the node of a list representing the layout
/*!
 Represents all the information of an element that will be
 printed on the screen. Also points to the next element of the
 screen that will be printed next to it
*/
class list_layout_node
{
    list_layout_node *next;     //!< Pointer to next node
    coord pos;                  //!< Position where to print
    int tcolor;                 //!< Text colour
    int bcolor;                 //!< Background colour
    char str[100];              //!< String to print

    //! How to print the string; mainly for passwords
    int print_type;

    public:
        list_layout_node();     //!< Ctor
        ~list_layout_node();    //!< Dtor

        //!@{ Setter functions
        void setnext(list_layout_node *);
        void setpos(coord);
        void settcolor(int);
        void setbcolor(int);
        void setstr(const char *);
        void setprint_type(int);
        //!@}

        //!@{ Getter functions
        list_layout_node * getnext();
        coord getpos();
        int gettcolor();
        int getbcolor();
        const char * getstr();
        int getprint_type();
        //!@}

        //! Used to distinguish will be printed i.e.
        //! as is or hidden (as passwords)
        enum print_types
        {
            DEFAULT,
            PASSWORD
        };
};

//! A node of the representation of string as a linked list
struct string_node
{
    string_node *next;  //!< Pointer to next node
    string_node *prev;  //!< Pointer to previous node
    char data;          //!< Character stored in string

    string_node();      //!< Ctor
};

//! Represents all interactive information
/*!
 Basically a parent class of all the classes that
 represent the elements of the layout the user can
 interact with.
 Used so that all those elements can be clubbed together
 and the input be taken.
*/
class interactive : public list_layout_node
{
    interactive *prev;      //!< ptr to previous node
    interactive *next;      //!< ptr to next node
    int offset;             //!< offset to y position when printing
    public:
        interactive();      //!< Ctor
        ~interactive();     //!< Dtor

        //! Empty input function that will be overridden by children
        /*!
         \param offset The offset to y position
         \return Action that was performed by the user
        */
        virtual int input(int offset);

        //! Setter function
        void setoffset(int);

        //! Getter function
        int getoffset();

        //! Actions that are performed by user; returned from input func.
        enum actions
        {
            GOTONEXT,
            GOTOPREV,
            CLICKED,
            BACK //!< When shift-bckspc is pressed
        };

        //! Keys that user can press to navigate the form
        enum keys
        {
            TAB,
            ENTER,
            BACKSPACE,
            SHIFT_BACKSPACE,
            SHIFT_TAB,
            HOME,
            END,
            DELETE,
            UP,
            DOWN,
            LEFT,
            RIGHT
        };

        //! Gets key from user and returns code
        /*
         \return Keyname corresponding to enum keys
        */
        static int getkey();
};

//! Represents a text box
/*!
 Inherits from interactive as a text box can be interacted
 with. Gets data from user and stores it as a string that
 can be further converted to the required data type
*/
class text_box : public interactive
{
    //! Represents if the data entered in the text box
    //! should be displayed as is or replaced with asterisks
    int is_password;

    public:
        text_box(); //!< Ctor

        //! Takes input and returns user action
        /*!
         /param offset Offset of y coordinate to print
         /return Action performed by user
        */
        int input(int offset = 0);

        //! Prints string represented by a linked list
        /*
         Takes in the head pointer of the linked list
         string and prints the string by iterating through
         the list. Has no other side effects.
         /param head ptr to head of the linked list
        */
        void print_str(string_node *head);

        //! Setter function
        void setis_password(int);
};
```

```

//! Represents a button that can be clicked
/*!
 Inherits from interactive as a button can be interacted with.
 A user can click the button while it's input function is
 running which will return the user action
*/
class button : public interactive
{
    int tcolor_selected; //!< tcolor when selected
    int bcolor_selected; //!< bcolor when seilected

    public:
        button(); //!< Ctor

        //!@{ Setter functions
        void settcolor_selected(int);
        void setbcolor_selected(int);
        //!@}

        //!@{ Getter functions
        int gettcolor_selected();
        int getbcolor_selected();
        //!@}

        //! Input function
        /*!
         Effectively allows the button to be clicked
         /param offset Offset of y coordinate to print
         /return Action performed by the user
        */
        int input(int offset = 0);

        //! Prints the button
        /*!
         /param isselected Indicates if button is selected or not
        */
        void print(int isselected = 0);
};

//! Represents the layout of the page
/*!
 Incorporates elements like simple nodes as well as other
 interactive elements. This layout can be contained within
 a specific height and the overflowing content can reached
 by scrolling which is also implemented here.
*/
class list_layout
{
    //!@{ Pointers to implement a linked list to elements
    list_layout_node *head; //!< ptr to head node
    list_layout_node *current; //!< ptr to current node
    //!@}

    coord corner_top_left; //!< top left corner of container

    /*!
     Following are used as temporary placeholders till data
     is written to the nodes
    */
    ///!@{
    coord pos;
    int tcolor;
    int bcolor;
    int tcolor_selected;
    int bcolor_selected;
    int tcolor_input;
    int bcolor_input;
    ///!@}

    //!@{ For scrolling implementation
    int height; //!< Height of the layout
    int width; //!< Width of the layout
    int lines_scrolled; //!< Lines currently scrolled
    //!@}

    //! For better verbosity at internal level
    enum print_modes
    {
        DISPLAY,
        HIDE
    };

    //! Prints the layout
    /*!
     Prints the layout by iterating through the internal
     linked list maintained. Has no other side effects
     /param print_mode How to print the data
    */
    void print(int print_mode = DISPLAY);
    public:
        list_layout(); //!< Ctor

        //!@{ Set an element (node)
        list_layout& operator<<(coord); //!< Set coord of node

        //! Set data held by the node
        list_layout& operator<<(const char *);
        //!@}

        //! Set a text box
        /*!
         Sets a text box at the position indicated by pos and
         returns a pointer to it
         /param pos Position at which to set text box
         /param is_pass If the text box has a password, set to 1
         /return pointer to the text box set (casted to interactive *)
        */
        interactive * settext_box(coord pos, int is_pass = 0);

        //! Set a button
        /*!
         Sets a button at the position indicated by pos and
         returns a pointer to it
         /param pos Position at which to set the button
         /param txt The text the button displays
        */
        interactive * setbutton(coord pos, const char *txt);

        //!@{ Setter functions
        void settcolor(int);
        void setbcolor(int);
        void settcolor_selected(int);
        void setbcolor_selected(int);
        void settcolor_input(int);
        void setbcolor_input(int);
        void setcorner_top_left(coord);
        void setheight(int);
        void setwidth(int);
        void setlines_scrolled(int);
        void setpos(coord);
        //!@}

        //!@{ Getter functions
        int getheight();
        int getwidth();
        int getlines_scrolled();
        coord getpos();
        coord getcorner_top_left();
        //!@}

        void display(); //!< Display the layout
        void hide(); //!< Hide the layout
        void clear(); //!< Deletes contents of the layout
};

//! Represents a border
/*!
 Basically represents a border with characters that can be
 customised to suit the requirements.
*/
class frame
{
    char border_chars[8];   //!< chars used to draw border
    int tcolor;             //!< text color
    int bcolor;             //!< background color

    //! Represents what part of frame is visible.
    int sides_visibility[8];
    int frame_visibility;   //!< Frame visible or not
    coord corner_top_left;  //!< coord of top left corner

    //!@{These include the border characters too
    int height;             //!< height
    int width;              //!< width
    //!@}

    //! Internal pmt used by operator<<
    int state;

    //! Sets the visibility of the side
    /*!
     /param side Specifies the side using ui::dir
     /param visib Set the visibility of the side
    */
    void setside_visibility(int side, int visib);

    //! Converts the ui::dir code into internally usable code
    int convert(int);

    //! Prints the frame
    /*!
     /param f_visib If 1, frame is printed; hidden if it's 0
    */
    void print(int f_visib = 1);

    public:

        //! Used to set the visibility mode of the frame
        /*
         all: -----
              |   |
              -----
         nosides: ------

                  ------
        */
        enum visibility_modes
        {
            all = 1,
            nosides = 2
        };

        //! Ctor
        /*!
         /param corner_top_left Top left corner of frame
         /param width Width of the frame
         /param height Height of the frame
        */
        frame(coord corner_top_left = coord(1,1), int width =
        ui::scr_width, int height = ui::scr_height - 1);

        void display(); //!< Display the frame
        void hide();    //!< Hides the frame

        //! Sets the visibility mode of the frame
        void setvisibility_mode(int);

        //!@{ operator<<
        frame & operator<<(int); //!<Sets state

        //! Sets border_char according to state
        frame & operator<<(char);
        //!@}

        //!@{ Getter functions
        int getheight();
        int getwidth();
        coord getcorner_top_left();

        //! Returns 1 if visible; 0 = not visible
        int getframe_visibility();
        int gettcolor();
        int getbcolor();
        char getborder_char(int);
        int getside_visibility(int);
        //!@}

        //!@{ Setter functions
        void setheight(int);
        void setwidth(int);
        void settcolor(int);
        void setbcolor(int);
        void setcorner_top_left(coord);
        //!@}
};

//! Info related to a text box
/*!
 Stores information related to a text box
 Such as what type to convert it's data to
 and where to store it
*/
struct info_tbox
{
    text_box * tbox;    //!< ptr to text_box whose info is stored

    //! Data type to convert the string stored in text box to
    int type;
    void * data_store;  //!< Where to store converted data

    /*!
     A validation function that's used to validate the
     string stored in the text box to see if it is of
     the required type before converting it.
     /param str The string to validate
     /param return 1, if string is validate; 0, otherwise
    */
    int (*validator)(const char *str);

    //! The data types the string stored in text box represents
    /*!
     Whenever a text box is set, the pointer to the place where
     final data has to be stored is converted to a void* and
     the data type is stored.
     So, void* in different cases is:

     data type     | What void* was
     ------------- | --------------------
     INT           | int *
     LONG          | long *
     UNSIGNED_LONG | unsigned long *
     STRING        | char *
     CHAR          | char *
     DOUBLE        | double *
     FLOAT         | float *
     PASSWORD      | char *
    */
    enum data_types
    {
        INT,
        LONG,
        UNSIGNED_LONG,
        STRING,
        CHAR,
        DOUBLE,
        FLOAT,
        PASSWORD,
        OTHER //!< Not supported at the moment
    };

    info_tbox();    //!< Ctor

    //! Sets data to the data_store
    /*!
     Gets the string stored in the text box, validates
     it using the validation function and then converts
     the string to the required data type and stores it in
     the required space
     /return 1 on success, 0 on invalid data
    */
    int setdata();
};

/*!
 Contains default validation functions of type
 int f(char *)
 that take in a string and return 1 if the string
 is valid and 0, otherwise
*/
class validation
{
    validation(); //!< Object of this class is not allowed
    public:

        //!@{ Default validation functions
        static int vint(const char *);
        static int vlong(const char *);
        static int vunsigned_long(const char *);
        static int vstring(const char *);
        static int vchar(const char *);
        static int vdouble(const char *);
        static int vfloat(const char *);
        //!@}

        /*!
         Get the default validator function for the type
         specified. If func is not NULL, returns default
         function, else returns v
        */
        static validator_f getvalidator(int type,
                                  validator_f func);
};

/*!
 Represents a line with the three strings depiciting
 left, middle and right aligned stuff respectively
*/
struct line
{
    //!@{ Parts of the line
    char left[100];   //!< left aligned
    char middle[100]; //!< centre aligned
    char right[100];  //!< right aligned
    //!@}

    int width;  //!< width of line
    int tcolor; //!< text color
    int bcolor; //!< background color
    coord corner_top_left; //!< coord of top left corner

    line(); //!< Ctor
    void display(); //!< Display the line
    void hide();    //!< Hide the line
    void clear();   //!< Delete the data stored

    private:
        void print(int); //!< Print the line according to arg
};

/*!
 Default Back function for use in the class box.
 Can't declare it as member function as member functions
 are not inherently addresses and setting it as a member function
 was causing unsolvable problems
*/
int default_back_func();

//! A box that has a border and a layout
/*!
 Basically incorporates all the elements into a single
 entity that the user will interact with.
 Basically looks like
 ------------------  <-- Frame
 | -------------- |
 | |          <--------Layout (No border)
 | |            | |
 | -------------- <-----Padding (between layout and frame)
 ------------------
*/
class box
{
    int height;     //!< Height of the box
    int width;      //!< Width of the box
    int padding;    //!< Padding between frame and layout

    /*!
     Wraps a string with specified number of characters
     in each line
     /param str String to wrap. Will be modified
     /param length Number of chars in a line
     /param return_one_line Sets string to have only one line
     /return Number of lines after wrapping
    */
    int wrap(char str[], int length, int return_one_line = 0);

    //! Sets the tbox
    /*!
     Sets the textbox in the layout and also stores the
     correpsonding data in a tbox that is stored in the array
     /param data_type Type of data in text box
     /param ptr Pointer to the data store to set in tbox
    */
    void set_tbox(int data_type, void *ptr);

    //!@{ Lists of interactives and text boxes
    interactive * list_interactive[30];
    info_tbox list_tbox[30];
    int index_interactive; //!< Index of element to set next
    int index_tbox; //!< Index of element to set next
    //!@}

    //! Clicking this button exits the loop
    button * exit_btn;

    //!@{ Toggles that help setting required info in layout
    int center_toggle;
    int default_toggle;
    int right_toggle;
    int header_toggle;
    int footer_toggle;
    int password_toggle;
    //!@}

    char default_text[100]; //!< Default text to set in textbox

    /*!
     A temporary variable that stores validator func till it
     is stored in the required place.
    */
    int (*temp_validator)(const char *);

    //!@{ Header and footer
    line header;
    line footer;
    //!@}

    /*!
     The function is called when the user performs a back func
     while interacting with any interactive
     /return 1, if loop exits on back; 0, if it does nothing
    */
    int (*back_func)();

    protected:
        coord pos_pointer;  //!< Pos of pointer in box
        list_layout layout; //!< Layout in which data is stored
        coord corner_top_left; //!< Coord of top left corner

    public:

        //!@{ Manipulators can be used to alter function of <<
        static manipulator setheader;
        static manipulator setfooter;
        static manipulator setpassword;
        //!@}

        frame f;    //!< Border of the box

        //! Ctor
        /*!
         Initialises all the variables of the class
         /param corner_top_left The top left corner
         /param width Width of box (includes border)
         /param height Height of box (includes border)
        */
        box(coord corner_top_left = coord(1,1),
            int width = ui::scr_width,
            int height= ui::scr_height - 1);

        //!@{ Getter functions
        coord getcorner_top_left();
        int getheight();
        int getwidth();
        int getpadding();
        //!@}

        //!@{ Setter functions
        void setcorner_top_left(coord);
        void setheight(int);
        void setpadding(int);
        void settcolor(int);
        void setbcolor(int);
        void settcolor_selected(int);
        void setbcolor_selected(int);
        void settcolor_input(int);
        void setbcolor_input(int);
        void setback_func( int(*f)(void) );
        //!@}

        //!@{ operator<< is used for adding data to the box's
        //!   layout that will be printed
        box & operator<<(char *);
        box & operator<<(char);
        box & operator<<(int);
        box & operator<<(long);
        box & operator<<(unsigned long);
        box & operator<<(double);
        box & operator<<(float);
        box & operator<<(manipulator);
        //!@}

        //!@{ operator>> is used for basically setting a text
        //!   box at the place where pos_pointer is currently
        //!   at
        box & operator>>(char *&);
        box & operator>>(char &);
        box & operator>>(int &);
        box & operator>>(long &);
        box & operator>>(unsigned long &);
        box & operator>>(double &);
        box & operator>>(float &);
        box & operator>>(manipulator);

        //! Using this before another >> will set this func
        //! as the validator of that text box
        box & operator>>(int (*)(const char *));
        //!@}

        void setexit_button(char *);

        //!@{ Sets default for the next text box and
        //!   clears it after the next text box has been
        //!   set
        void setdefault(char *);
        void setdefault(char);
        void setdefault(int);
        void setdefault(long);
        void setdefault(unsigned long);
        void setdefault(double);
        void setdefault(float);
        //!@}

        /*!
         Sets the box to loop, effectively enabling
         all the text boxes and buttons. Also enables
         scrolling
        */
        void loop();

        void display(); //!< Display the box
        void hide();    //!< Hide the box
        void clear();   //!< Delete the contents of the box

        //!@{ Functions to set header and footer
        void setheader_tcolor(int); //!< set header color
        void setfooter_tcolor(int); //!< set footer color
        void clear_header(); //!< Delete contents of header
        void clear_footer(); //!< Delete contents of footer
        //!@}
};

#endif /* UI_HPP */
```

C++ files (.cpp)

1. code/iface3.cpp

```
#include <fstream.h>
#include "base.hpp"
#include "iface.hpp"
#include "hosp.hpp"
#include "emp.hpp"

void interface::employee_management()
{
    const int menu_corner_top_left_y = 5;
    coord c(ui::scr_width * 0.2, menu_corner_top_left_y);
    int ch;
    while(1)
    {
        interface::clear_error();
        box menu(c, ui::scr_width * 0.6, ui::scr_height - 6 );
        menu.settcolor(GREEN);
        menu << ui::centeralign << "Employee Management" << ui::endl << ui::endl;
        menu.settcolor(ui::tcolor);
        menu << "1. View employee data" << ui::endl
             << "2. Add new employee" << ui::endl
             << "3. Remove existing employee" << ui::endl
             << "4. Edit employee data" << ui::endl
             << "5. Pay salary to individual employee" << ui::endl
             << "6. Pay salary to all employees" << ui::endl
             << "7. Back" << ui::endl
             << ui::endl << "Enter your choice: ";
        menu.settcolor_input(YELLOW);
        validate_menu::set_menu_limits(1, 7);
        menu >> validate_menu::input >> ch;
        menu << ui::endl;
        menu.setexit_button("Submit");
        menu.loop();
        menu.hide();
        switch (ch)
        {
            case 1:
            {
                emp_mgmt::view_emp();
                break;
            }
            case 2:
            {
                emp_mgmt::add_emp();
                break;
            }
            case 3:
            {
                emp_mgmt::remove_emp();
                break;
            }
            case 4:
            {
                emp_mgmt::edit_emp();
                break;
            }
            case 5:
            {
                emp_mgmt::pay_emp();
                break;
            }
            case 6:
            {
                emp_mgmt::pay_all();
                break;
            }
            case 7:
            {
                return;
            }
        }
    }
}

void interface::employee_screen(unsigned long id)
{
    void * temp = malloc( sizeof(doctor) ); //as doctor has the greatest size
        among employee, doctor, nurse and receptionist classes
    if(temp == NULL)
    {
        interface::log_this("interface::employee_screen() : Not enough memory to
            allocate buffer void * temp = malloc( sizeof(doctor) )");
        interface::error("Out of memory!! Check log");
        getch();
        return;
    }
    if(!hospital::get_employee_by_id(id, temp))
    {
        interface::error("ID not found or error while reading from file!");
        getch();
        free(temp);
        return;
    }
    employee *e = (employee *) temp;
    const int menu_corner_top_left_y = 5;
    coord c(ui::scr_width * 0.2, menu_corner_top_left_y);
    int ch;
    str heading = "Welcome, ";
    strcat( heading, e->get_name() );
    strcat(heading, "!");
    while(1)
    {
        interface::clear_error();
        box menu(c, ui::scr_width * 0.6, ui::scr_height - 6 );
        menu.settcolor(GREEN);
        menu << ui::centeralign << heading << ui::endl << ui::endl;
        menu.settcolor(ui::tcolor);
        menu << "1. View profile" << ui::endl
             << "2. Change login details" << ui::endl
             << "3. View last 5 transactions" << ui::endl;
        emp_type type_of_emp = id_to_emp::convert(id);
        if(type_of_emp == RECEPTIONIST)
        {
            menu << "4. Manage patients" << ui::endl
                 << "5. Exit" << ui::endl;
        }
        else
        {
            menu << "4. Exit" << ui::endl;
        }
        menu << ui::endl << "Enter your choice: ";
        menu.settcolor_input(YELLOW);
        if(type_of_emp == RECEPTIONIST)
        {
            validate_menu::set_menu_limits(1, 5);
        }
        else
        {
            validate_menu::set_menu_limits(1, 4);
        }
        menu >> validate_menu::input >> ch;
        menu << ui::endl;
        menu.setexit_button("Submit");
        menu.loop();
        menu.hide();
        switch(ch)
        {
            case 1:
            {
                if( !emp_mgmt::view_emp(id) )
                {
                    interface::error("Failed to display profile!");
                    getch();
                }
                break;
            }
            case 2:
            {
                int ch;
                while(1)
                {
                    box menu3(c, ui::scr_width * 0.6, ui::scr_height - 6);
                    menu3.settcolor(GREEN);
                    menu3 << ui::centeralign << "Change login details" << ui::
                        endl << ui::endl;
                    menu3.settcolor(WHITE);
                    menu3 << "1. Change User ID" << ui::endl
                          << "2. Change Password" << ui::endl
                          << "3. Back" << ui::endl
                          << "Enter your choice: ";
                    menu3.settcolor(ui::tcolor);
                    menu3.settcolor_input(YELLOW);
                    validate_menu::set_menu_limits(1, 3);
                    menu3 >> validate_menu::input >> ch;
                    menu3 << ui::endl;
                    menu3.setexit_button("Submit");
                    menu3.loop();
                    menu3.hide();
                    switch(ch)
                    {
                        case 1:
                        {
                            str new_username;
                            box menu4( menu3.getcorner_top_left(), menu3.getwidth
```

```cpp
                            (), menu3.getheight() );
                        menu4.settcolor(GREEN);
                        menu4 << ui::centeralign << "Change login details" << ui::endl << ui::endl;
                        menu4.settcolor(WHITE);
                        menu4 << "Change User ID" << ui::endl;
                        menu4.settcolor(ui::tcolor);
                        menu4 << "User ID: ";
                        menu4.setdefault( e->account.get_username() );
                        menu4.settcolor_input(YELLOW);
                        menu4 >> new_username;
                        menu4.setexit_button("Submit");
                        menu4.setback_func(back_func::set_backbit);
                        menu4.loop();
                        menu4.hide();
                        if(back_func::backbit)
                        {
                            back_func::backbit = 0;
                            break;
                        }
                        e->account.set_username(new_username);
                        const int notice_height = 10;
                        box notice( menu4.getcorner_top_left(), menu4.getwidth(), notice_height );
                        notice.settcolor(GREEN);
                        notice << ui::centeralign << "Change login details" << ui::endl << ui::endl;
                        if( !hospital::write_employee(temp) )
                        {
                            notice.settcolor(RED);
                            notice << "Failed to write new user ID to file! Check log" << ui::endl;
                        }
                        else
                        {
                            notice.settcolor(GREEN);
                            notice << "User ID changed successfully!" << ui::endl;
                        }
                        notice.setexit_button("Back");
                        notice.loop();
                        notice.hide();
                        goto loop_exit;
                    }
                    case 2:
                    {
                        str curr_pwd, new_pwd;
                        for(int i = 0; i < 3; ++i)
                        {
                            box menu4( menu3.getcorner_top_left(), menu3.getwidth(), menu3.getheight() );
                            menu4.settcolor(GREEN);
                            menu4 << ui::centeralign << "Change login details" << ui::endl << ui::endl;
                            menu4.settcolor(WHITE);
                            menu4 << "Change Password" << ui::endl;
                            menu4.settcolor(ui::tcolor);
                            menu4 << "Enter current password: ";
                            menu4.settcolor_input(YELLOW);
```

```
                                menu4 >> box::setpassword >> curr_pwd;
                                menu4.setexit_button("Submit");
                                menu4.setback_func(back_func::set_backbit);
                                menu4.loop();
                                menu4.hide();
                                if(back_func::backbit)
                                {
                                    break;
                                }
                                if( e->account.login(curr_pwd) )
                                {
                                    interface::clear_error();
                                    break;
                                }
                                interface::error("Invalid password!! Try again...");
                            }
                            if(back_func::backbit)
                            {
                                back_func::backbit = 0;
                                break;
                            }
                            if(i == 3)
                            {
                                const int notice_height = 10;
                                box notice( menu3.getcorner_top_left(), menu3.getwidth(), notice_height);
                                notice.settcolor(GREEN);
                                notice << ui::centeralign << "Change login details" << ui::endl << ui::endl;
                                notice.settcolor(RED);
                                notice << "Since you entered the wrong password too many times, you have been logged out. "
                                       << "Hit the button below to exit the program." << ui::endl << ui::endl;
                                notice.setexit_button("Exit");
                                notice.loop();
                                notice.hide();
                                free(temp);
                                return;
                            }
                            box menu5( menu3.getcorner_top_left(), menu3.getwidth(), menu3.getheight() );
                            menu5.settcolor(GREEN);
                            menu5 << ui::centeralign << "Change login details" << ui::endl << ui::endl;
                            menu5.settcolor(WHITE);
                            menu5 << "Change Password" << ui::endl;
                            menu5.settcolor(ui::tcolor);
                            menu5 << "Enter new password: ";
                            menu5.settcolor_input(YELLOW);
                            menu5 >> box::setpassword >> new_pwd;
                            menu5.setexit_button("Submit");
                            menu5.setback_func(back_func::set_backbit);
                            menu5.loop();
                            menu5.hide();
                            if(back_func::backbit)
                            {
                                back_func::backbit = 0;
```

```cpp
                                break;      //At the "Enter new password" page, when shift+bkspc is pressed, control will go back to "Change login details" menu.
                            }
                            e->account = userid( e->account.get_username(), new_pwd );
                            const int notice2_height = 13;
                            box notice2( menu3.getcorner_top_left(), menu3.getwidth(), notice2_height );
                            notice2.settcolor(GREEN);
                            notice2 << ui::centeralign << "Change login details" << ui::endl << ui::endl;
                            if( !hospital::write_employee(temp) )
                            {
                                notice2.settcolor(RED);
                                notice2 << "Failed to write new password to file! Check log" << ui::endl;
                            }
                            else
                            {
                                notice2.settcolor(GREEN);
                                notice2 << "Password changed successfully!" << ui::endl;
                            }
                            notice2.settcolor(ui::tcolor);
                            notice2 << "Please logout and login again by exiting the program and restarting it." << ui::endl
                                    << "Press the button below to exit the program." << ui::endl;
                            notice2.setexit_button("Exit");
                            notice2.loop();
                            notice2.hide();
                            free(temp);
                            return;
                        }
                        case 3:
                        {
                            goto loop_exit;
                        }
                    }
                }
                loop_exit:
                break;
            }
            case 3:
            {
                transaction * t = e->get_last_5_transactions();
                if( t == NULL )
                {
                    interface::error("Error while reading or writing to file! Check log");
                    getch();
                    break;
                }
                coord c2(1, 4);
                box menu2(c2, (ui::scr_width / 2), ui::scr_height - 5);
                box sidemenu(( c2 + coord((ui::scr_width / 2) - 1, 0)), (ui::scr_width / 2) + 1, ui::scr_height - 5);
                menu2.f << ( ui::top | ui::left ) << (char)204
```

```
                        << ( ui::bottom | ui::left ) << (char)204
                        << ( ui::top | ui::right ) << (char)203
                        << ( ui::bottom | ui::right ) << (char)202;
                menu2.f.display();
                sidemenu.f << ( ui::top | ui::left ) << (char)203
                           << ( ui::bottom | ui::left ) << (char)202
                           << ( ui::top | ui::right ) << (char)185
                           << ( ui::bottom | ui::right ) << (char)185;
                sidemenu.f.display();
                menu2.settcolor(GREEN);
                menu2 << ui::centeralign << "View last 5 transactions" << ui::endl << ui::endl;
                menu2.settcolor(ui::tcolor);
                for(int i = 0; i < 5; ++i)
                {
                    if( t[i].amount == 0 && !strcmp(t[i].reason, "NA") &&
                        t[i]._date.day == 0 && t[i]._date.month == 0 && t[i]._date.year == 0
                        && t[i]._time.hour == 25 && t[i]._time.minute == 0 && t[i]._time.second == 0 )
                    {
                        break;
                    }
                    if(i < 3)
                    {
                        menu2 << i + 1 << ". " << t[i]._date << ", " << t[i]._time << ui::endl
                              << "Amount: " << t[i].amount << ui::endl
                              << "Reason: " << t[i].reason << ui::endl;
                    }
                    else
                    {
                        sidemenu << i + 1 << ". " << t[i]._date << ", " << t[i]._time << ui::endl
                                 << "Amount: " << t[i].amount << ui::endl
                                 << "Reason: " << t[i].reason << ui::endl;
                    }
                }
                free(t);
                if(i <= 3)
                {
                    menu2.setexit_button("Back");
                    menu2.loop();
                }
                else
                {
                    sidemenu.setexit_button("Back");
                    sidemenu.loop();
                }
                menu2.hide();
                sidemenu.hide();
                window.f.display();
                break;
            }
            case 4:
            {
                if(type_of_emp == RECEPTIONIST)
                {
                    interface::patient_management();
                    break;
                }
                else
                {
                    free(temp);
                    return;
                }
            }
            case 5:
            {
                free(temp);
                return;
            }
        }
    }
 }

emp_mgmt::emp_mgmt()
{}

void emp_mgmt::view_emp()
{
    const int menu2_height = 10;
    box menu2( coord(ui::scr_width * 0.2, 5), ui::scr_width * 0.6, menu2_height);
    menu2.settcolor(GREEN);
    menu2 << ui::centeralign << "Employee Management" << ui::endl << ui::endl;
    menu2.settcolor(WHITE);
    menu2 << "View employee data" << ui::endl;
    menu2.settcolor(ui::tcolor);
    menu2 << "Enter employee's id: ";
    unsigned long id;
    menu2.settcolor_input(YELLOW);
    menu2 >> id;
    menu2 << ui::endl;
    menu2.setexit_button("Submit");
    menu2.setback_func(back_func::set_backbit);
    menu2.loop();
    menu2.hide();
    if(back_func::backbit)
    {
        back_func::backbit = 0;
        return;
    }
    view_emp(id);
}

int emp_mgmt::view_emp(unsigned long id)
{
    void * temp = malloc( sizeof(doctor) ); //as doctor has the greatest size
        among employee, doctor, nurse and receptionist classes
    if(temp == NULL)
    {
        interface::log_this("emp_mgmt::view_emp(int) : Not enough memory to
            allocate buffer void * temp = malloc( sizeof(doctor) )");
        interface::error("Out of memory!! Check log");
        getch();
        return 0;
    }
    if(!hospital::get_employee_by_id(id, temp))
```

```cpp
    {
        interface::error("ID not found or error while reading from file!");
        getch();
        free(temp);
        return 0;
    }
    employee *e = (employee *) temp;
    box menu3( coord(ui::scr_width * 0.2, 5), ui::scr_width * 0.6, ui::scr_height - 6 );
    menu3.settcolor(GREEN);
    menu3 << ui::centeralign << "Employee Management" << ui::endl << ui::endl;
    menu3.settcolor(WHITE);
    menu3 << "Employee Details: " << ui::endl;
    menu3.settcolor(ui::tcolor);
    menu3 << "ID: " << e->get_id() << ui::endl;
    menu3 << "Name: " << e->get_name() << ui::endl;
    menu3 << "Age: " << e->get_age() << ui::endl;
    menu3 << "Sex: " << (sex)e->get_sex() << ui::endl;
    menu3 << "Date of Birth: " << e->get_dob() << ui::endl;
    menu3 << "Address: " << e->get_address() << ui::endl;
    menu3 << "Phone no.: " << e->get_phone() << ui::endl;
    menu3 << "Salary: " << e->get_salary() << ui::endl;
    menu3 << "Shift timings: Starts - " << e->get_shift(START) << ui::endl;
    menu3 << "--------------: Ends - " << e->get_shift(END) <<ui::endl;
    switch( id_to_emp::convert( e->get_id() ) )
    {
        case INVALID:   //Test this case, menu3.hide() not working properly
        {
            menu3.clear();
            int menu3_height = 9;
            menu3.setheight(menu3_height);
            menu3.settcolor(GREEN);
            menu3 << ui::centeralign << "Employee Management" << ui::endl << ui::endl;
            menu3.settcolor(WHITE);
            menu3 << "Employee Details: " << ui::endl;
            menu3.settcolor(RED);
            menu3 << "Invalid ID!!" << id_to_emp::convert( e->get_id() );
            menu3.settcolor(ui::tcolor);
            menu3.setexit_button("Back");
            menu3.loop();
            menu3.hide();
            break;
        }
        case OTHERS:
        case RECEPTIONIST:  //there are no extra data members in class receptionist
        {
            menu3.setexit_button("Back");
            menu3.loop();  //   menu3.clear();  int w = window.getwidth(), m = menu3.getwidth(); menu3<<w<<' '<<m;  getch();
            menu3.hide();
            break;
        }
        case DOCTOR:
        {
            doctor *d = (doctor *)temp;
            menu3.hide();
            menu3.setcorner_top_left( coord( 1, menu3.getcorner_top_left().y ) );
            menu3.display();
            menu3.f << ( ui::top | ui::left ) << (char)204
                    << ( ui::bottom | ui::left ) << (char)204;
            menu3.f.display();
            box sidemenu( menu3.getcorner_top_left() + coord( menu3.getwidth() -
                1, 0 ), ( ui::scr_width - menu3.getwidth() + 1 ),  menu3.getheight
                () );
            sidemenu.f << ( ui::top | ui::left ) << (char)203
                       << ( ui::bottom | ui::left ) << (char)202
                       << ( ui::top | ui::right ) << (char)185
                       << ( ui::bottom | ui::right ) << (char)185;
            sidemenu.f.display();
            sidemenu << "Speciality(s)" << ui::endl;
            for(int i = 0; i < 2 && d->get_speciality()[i] <= GEN; ++i)
            {
                sidemenu << i + 1 << ". " << (body_parts)d->get_speciality()[i]
                    << ui::endl;
            }
            if(!i)
            {
                sidemenu << "None" << ui::endl;
            }
            sidemenu << "Patients currently under care:" << ui::endl;
            for(i = 0; d->get_patients()[i] && i < 10; ++i)
            {
                sidemenu << i + 1 << ". " << hospital::get_patient_by_id( d->
                    get_patients()[i] ).get_name() << ui::endl;
            }
            if(!i)
            {
                sidemenu << "None" << ui::endl;
            }
            sidemenu.setexit_button("Back");
            sidemenu.loop();
            menu3.hide();
            sidemenu.hide();
            window.f.display();
            break;
        }
        case NURSE:
        {
            nurse *n = (nurse *)temp;
            menu3.hide();
            menu3.setcorner_top_left( coord( 1, menu3.getcorner_top_left().y ) );
            menu3.display();
            menu3.f << ( ui::top | ui::left ) << (char)204
                    << ( ui::bottom | ui::left ) << (char)204;
            menu3.f.display();
            box sidemenu( menu3.getcorner_top_left() + coord( menu3.getwidth() -
                1, 0 ), ( ui::scr_width - menu3.getwidth() + 1 ),  menu3.getheight
                () );
            sidemenu.f << ( ui::top | ui::left ) << (char)203
                       << ( ui::bottom | ui::left ) << (char)202
                       << ( ui::top | ui::right ) << (char)185
                       << ( ui::bottom | ui::right ) << (char)185;
            sidemenu.f.display();
            sidemenu << "Patients currently under care:" << ui::endl;
            for(int i = 0; n->get_patients()[i] && i < 5; ++i)
            {
                sidemenu << i + 1 << ". " << hospital::get_patient_by_id( n->
                    get_patients()[i] ).get_name() << ui::endl;
            }
            if(!i)
            {
                sidemenu << "None" << ui::endl;
            }
            sidemenu.setexit_button("Back");
            sidemenu.loop();
            menu3.hide();
            sidemenu.hide();
            window.f.display();
            break;
        }
    }
    free(temp);
    return 1;
}

void emp_mgmt::add_emp()
{
    int ch;
    str name, dob_str, adr_hno, adr_street, adr_city, adr_dist, adr_state,
        shift_start_str, shift_end_str, uid, pwd;
    unsigned sex_choice;
    Date dob;
    address adr;
    phone phn_no;
    unsigned long salary;
    Time shift_start, shift_end;
    int speciality[2];
    const coord menu2_corner_top_left = coord(ui::scr_width * 0.2, 5);
    const int menu2_width = ui::scr_width * 0.6;
    menu2:
    {
        const int menu2_height = 17;
        box menu2(menu2_corner_top_left, menu2_width, menu2_height);
        menu2.settcolor(GREEN);
        menu2 << ui::centeralign << "Employee Management" << ui::endl << ui::endl
            ;
        menu2.settcolor(WHITE);
        menu2 << "Add new employee" << ui::endl;
        menu2.settcolor(ui::tcolor);
        menu2 << "Step 1: Select employee type" << ui::endl << ui::endl
              << "1. Doctor" << ui::endl
              << "2. Nurse" << ui::endl
              << "3. Receptionist" << ui::endl
              << "4. Others" << ui::endl << ui::endl
              << "Enter your choice: ";
        validate_menu::set_menu_limits(1, 4);
        menu2.settcolor_input(YELLOW);
        menu2 >> validate_menu::input >> ch;
        menu2 << ui::endl;
        menu2.setexit_button("Submit");
        menu2.setback_func(back_func::set_backbit);
        menu2.loop();
        menu2.hide();
        if(back_func::backbit)
        {
            back_func::backbit = 0;
            return;
        }
    }
    menu3:
    {
        box menu3( menu2_corner_top_left, menu2_width, ui::scr_height - 6 );
        menu3.settcolor(GREEN);
        menu3 << ui::centeralign << "Employee Management" << ui::endl << ui::endl;
        menu3.settcolor(WHITE);
        menu3 << "Add new employee" << ui::endl;
        menu3.settcolor(ui::tcolor);
        menu3 << "Step 2: Add employee details" << ui::endl << ui::endl;
        menu3.settcolor_input(YELLOW);
        menu3 << "Name: ";
        menu3 >> name;
        menu3 << "Sex: 1. Male | 2. Female | 3. Transsexual" << ui::endl
              << "---- Enter your choice: ";
        validate_menu::set_menu_limits(1, 3);
        menu3 >> validate_menu::input >> (int)sex_choice;
        menu3 << "Date of Birth(DD/MM/YYYY): ";
        menu3 >> hospital::date_validity >> dob_str;
        menu3 << "Address: " << ui::endl;
        menu3 << (char)26 << "House no.: ";
        menu3 >> adr_hno;
        menu3 << (char)26 << "Street: ";
        menu3 >> adr_street;
        menu3 << (char)26 << "City: ";
        menu3 >> adr_city;
        menu3 << (char)26 << "District: ";
        menu3 >> adr_dist;
        menu3 << (char)26 << "State: ";
        menu3 >> adr_state;
        menu3 << "Phone no.: ";
        menu3 >> phn_no;
        menu3 << "Salary: ";
        menu3 >> salary;
        menu3 << "Shift timings: Starts - (HH:MM:SS)";
        menu3 >> hospital::time_validity >> shift_start_str;
        menu3 << "-------------: Ends - (HH:MM:SS)";
        menu3 >> hospital::time_validity >> shift_end_str;
        menu3.setexit_button("Submit");
        menu3.setback_func(back_func::set_backbit);
        menu3.loop();
        menu3.hide();
        if(back_func::backbit)
        {
            back_func::backbit = 0;
            goto menu2;
        }
        --sex_choice;
        dob = hospital::str_to_date(dob_str);
        adr = address(adr_hno, adr_street, adr_city, adr_dist, adr_state);
        shift_start = hospital::str_to_time(shift_start_str);
        shift_end = hospital::str_to_time(shift_end_str);
    }
    menu4:

```

```
    if(ch != 4)
    {
        box menu4( menu2_corner_top_left, menu2_width, ui::scr_height - 6 );
        menu4.settcolor(GREEN);
        menu4 << ui::centeralign << "Employee Management" << ui::endl << ui::endl
            ;
        menu4.settcolor(WHITE);
        menu4 << "Add new employee" << ui::endl;
        menu4.settcolor(ui::tcolor);
        menu4.settcolor_input(YELLOW);
        menu4 << "Step 3: Add login details" << ui::endl << ui::endl;
        menu4 << "User ID: ";
        menu4 >> uid;
        menu4 << "Password: ";
        menu4 >> box::setpassword >> pwd;
        menu4 << ui::endl;
        menu4.setexit_button("Submit");
        menu4.setback_func(back_func::set_backbit);
        menu4.loop();
        menu4.hide();
    }
    if(back_func::backbit)
    {
        back_func::backbit = 0;
        goto menu3;
    }
    if(ch == 1)
    {
        coord c(1, 4);
        box menu5(c, (ui::scr_width / 2), ui::scr_height - 5);
        box inp_box(( c + coord((ui::scr_width / 2) - 1, 0)), (ui::scr_width / 2)
             + 1, ui::scr_height - 5);
        menu5.f << ( ui::top | ui::left ) << (char)204
                  << ( ui::bottom | ui::left ) << (char)204
                  << ( ui::top | ui::right ) << (char)203
                  << ( ui::bottom | ui::right ) << (char)202;
        menu5.f.display();
        inp_box.f << ( ui::top | ui::left ) << (char)203
               << ( ui::bottom | ui::left ) << (char)202
               << ( ui::top | ui::right ) << (char)185
               << ( ui::bottom | ui::right ) << (char)185;
        inp_box.f.display();
        menu5 << ui::centeralign << "Employee Management" << ui::endl << ui::endl
            ;
        menu5.settcolor(WHITE);
        menu5 << "Add new employee" << ui::endl;
        menu5.settcolor(ui::tcolor);
        menu5 << "Step 4: Add doctor details" << ui::endl << ui::endl;
        menu5 << "Specialization of doctor (max 2)" << ui::endl
              << "Choose from the following list: " << ui::endl;
        for(int i = 0; i <= GEN; ++i)
        {
            if(i <= 8)
            {
                menu5 << i << ". " << (body_parts)i << ui::endl;
            }
            else
            {
                inp_box << i << ". " << (body_parts)i << ui::endl;
            }
        }
        inp_box.settcolor_input(YELLOW);
        inp_box << "Enter the number corresponding to the required entry in the 2
             fields below" << ui::endl;
        validate_menu::set_menu_limits(BRAIN, GEN);
        inp_box << (char)26;    inp_box >> validate_menu::input >> speciality[0];
        inp_box << (char)26;    inp_box >> validate_menu::input >> speciality[1];
        inp_box << ui::endl;
        inp_box.setexit_button("Submit");
        inp_box.setback_func(back_func::set_backbit);
        inp_box.loop();
        menu5.hide();
        inp_box.hide();
        window.f.display();
    }
    if(back_func::backbit)
    {
        back_func::backbit = 0;
        goto menu4;
    }
    void * temp = NULL;
    unsigned long id;
    switch (ch)
    {
        case 1:
        {
            doctor x(name, sex_choice, dob, adr, phn_no, salary, shift_start,
                shift_end, speciality[0], speciality[1], uid, pwd);
            temp = &x;
            id = x.get_id();
            break;
        }
        case 2:
        {
            nurse x(name, sex_choice, dob, adr, phn_no, salary, shift_start,
                shift_end, uid, pwd);
            temp = &x;
            id = x.get_id();
            break;
        }
        case 3:
        {
            receptionist x(name, sex_choice, dob, adr, phn_no, salary,
                shift_start, shift_end, uid, pwd);
            temp = &x;
            id = x.get_id();
            break;
        }
        case 4:
        {
            employee x(name, sex_choice, dob, adr, phn_no, salary, shift_start,
                shift_end);
            temp = &x;
            id = x.get_id();
            break;
        }
    }
    const int notice_height = 12;
    box notice( menu2_corner_top_left, menu2_width, notice_height );
    notice.settcolor(GREEN);
    notice << ui::centeralign << "Employee Management" << ui::endl << ui::endl;
    if(!hospital::write_employee(temp))
    {
        notice.settcolor(RED);
        notice << "Employee addition unsuccessful!!";
        notice.setexit_button("Exit");
        notice.loop();
        notice.hide();
        return;
    }
    notice << "Employee added successfully!!" << ui::endl;
    notice.settcolor(WHITE);
    notice << "Hit the button below to display the details you entered: " << ui::
        endl;
    notice.settcolor(ui::tcolor);
    notice << ui::endl;
    notice.setexit_button("View employee...");
    notice.loop();
    notice.hide();
    view_emp(id);
}

void emp_mgmt::remove_emp()
{
    const coord menu2_corner_top_left = coord(ui::scr_width * 0.2, 5);
    const int menu2_width = ui::scr_width * 0.6;
    unsigned long id;
    char ch;
    menu2:
    {
        const int menu2_height = 10;
        box menu2(menu2_corner_top_left, menu2_width, menu2_height);
        menu2.settcolor(GREEN);
        menu2 << ui::centeralign << "Employee Management" << ui::endl << ui::endl
            ;
        menu2.settcolor(WHITE);
        menu2 << "Remove existing employee" << ui::endl;
        menu2.settcolor(ui::tcolor);
        menu2 << "Enter employee's id: ";
        menu2.settcolor_input(YELLOW);
        menu2 >> id;
        menu2 << ui::endl;
        menu2.setexit_button("Submit");
        menu2.setback_func(back_func::set_backbit);
        menu2.loop();
        menu2.hide();
    }
    if(back_func::backbit)
    {
        back_func::backbit = 0;
        return;
    }
    notice:
    {
        const int notice_height = 14;
        box notice(menu2_corner_top_left, menu2_width, notice_height);
        notice.settcolor(GREEN);
        notice << ui::centeralign << "Employee Management" << ui::endl << ui::
            endl;
        notice.settcolor(WHITE);
        notice << "Hit the button below to display the details of the employee
            you want to remove: " << ui::endl;
        notice.settcolor(ui::tcolor);
        notice << ui::endl;
        notice.setexit_button("View employee...");
        notice.setback_func(back_func::set_backbit);
        notice.loop();
        notice.hide();
    }
    if(back_func::backbit)
    {
        back_func::backbit = 0;
        goto menu2;
    }
    if( !view_emp(id) )
    {
        return;
    }
    notice2:
    {
        const int notice2_height = 14;
        box notice2( menu2_corner_top_left, menu2_width, notice2_height );
        notice2.settcolor(GREEN);
        notice2 << ui::centeralign << "Employee Management" << ui::endl << ui::
            endl;
        notice2.settcolor(WHITE);
        notice2 << "Are you sure you want to remove this employee?(y/n): " << ui
            ::endl;
        notice2.settcolor_input(YELLOW);
        notice2 >> ch;
        notice2.settcolor(ui::tcolor);
        notice2 << ui::endl;
        notice2.setexit_button("Submit");
        notice2.setback_func(back_func::set_backbit);
        notice2.loop();
        notice2.hide();
    }
    if(back_func::backbit)
    {
        back_func::backbit = 0;
        goto notice;
    }
    if(ch == 'n' || ch == 'N')
    {
        return;
    }
    const int notice3_height = 14;
    box notice3( menu2_corner_top_left, menu2_width, notice3_height );
    notice3.settcolor(GREEN);
    notice3 << ui::centeralign << "Employee Management" << ui::endl << ui::endl;
    notice3.settcolor(RED);
    str path;
    switch(id_to_emp::convert(id))
    {
        case INVALID:
            interface::log_this("emp_mgmt::remove_emp() : No file with zero id
                exists\nFunction aborted");
            notice3 << "Invalid ID supplied!! Check log" << ui::endl;
            notice3.setexit_button("Back");
            notice3.loop();
            notice3.hide();
            return;
        case OTHERS:
            sprintf(path, "employee/%lu", id);
            break;
        case DOCTOR:
            mkdir("employee/doctor");
            sprintf(path, "employee/doctor/%lu", id);
            break;
        case NURSE:
            mkdir("employee/nurse");
            sprintf(path, "employee/nurse/%lu", id);
            break;
        case RECEPTIONIST:
            mkdir("employee/receptionist");
            sprintf(path, "employee/receptionist/%lu", id);
            break;
    }
    int remove_status;
    str file;
    strcpy(file, path);
    strcat(file, "/base.dat");
    if( remove(file) == -1)
    {
        str log_str;
        sprintf(log_str, "emp_mgmt::remove_emp() : Failed to delete base.dat file
             of id %lu\nFunction aborted", id);
        interface::log_this(log_str);
        notice3 << "Failed to delete file of employee!!" << ui::endl;
        notice3.setexit_button("Back");
        notice3.loop();
        notice3.hide();
        return;
    }
    if( rmdir(path) == -1)
    {
        str log_str;
        sprintf(log_str, "emp_mgmt::remove_emp() : Failed to delete folder of id
            %lu", id);
        interface::log_this(log_str);
    }
    notice3.settcolor(GREEN);
    notice3 << "Employee deletion successful!!" << ui::endl;
    notice3.setexit_button("Back");
    notice3.loop();
    notice3.hide();
}

void emp_mgmt::edit_emp()
{
    void * temp = malloc( sizeof(doctor) ); //as doctor has the greatest size
        among employee, doctor, nurse and receptionist classes
    if(temp == NULL)
    {
        interface::log_this("emp_mgmt::edit_emp() : Not enough memory to allocate
            buffer void * temp = malloc( sizeof(doctor) )");
        interface::error("Out of memory!! Check log");
        getch();
        return;
    }
    str name, dob_str, adr_hno, adr_street, adr_city, adr_dist, adr_state,
        shift_start_str, shift_end_str, uid, pwd, default_dob_str,
        default_shift_str;
    unsigned sex_choice;
    Date dob;
    address adr;
    phone phn_no;
    unsigned long salary, id;
    Time shift_start, shift_end;
    const coord menu2_corner_top_left(ui::scr_width * 0.2, 5);
    const int menu2_width = ui::scr_width * 0.6;
    menu2:
    {
        const int menu2_height = 10;
        box menu2(menu2_corner_top_left, menu2_width, menu2_height);
        menu2.settcolor(GREEN);
        menu2 << ui::centeralign << "Employee Management" << ui::endl << ui::endl
            ;
        menu2.settcolor(WHITE);
        menu2 << "Edit employee data" << ui::endl;
        menu2.settcolor(ui::tcolor);
        menu2 << "Step 1: Enter employee's id: ";
        menu2.settcolor_input(YELLOW);
        menu2 >> id;
        menu2 << ui::endl;
        menu2.setexit_button("Submit");
        menu2.setback_func(back_func::set_backbit);
        menu2.loop();
        menu2.hide();
    }
    if(back_func::backbit)
    {
        back_func::backbit = 0;
        free(temp);
        return;
    }
    if(!hospital::get_employee_by_id(id, temp))
    {
        interface::error("ID not found or error while reading from file!");
        getch();
        free(temp);
        return;
    }
    notice:
    {
        const int notice_height = 14;
        box notice(menu2_corner_top_left, menu2_width, notice_height);
        notice.settcolor(GREEN);
        notice << ui::centeralign << "Employee Management" << ui::endl << ui::
            endl;
        notice.settcolor(WHITE);
        notice << "Details of the employee will now be shown with the existing
            data filled. "
              << "Change the data fields that you require to change, and leave
```

```
                    the other data fields as they are. "
                << "When you are finished, press Submit to submit the new details.
                    " << ui::endl;
        notice.settcolor(ui::tcolor);
        notice << ui::endl;
        notice.setexit_button("View employee...");
        notice.setback_func(back_func::set_backbit);
        notice.loop();
        notice.hide();
    }
    if(back_func::backbit)
    {
        back_func::backbit = 0;
        goto menu2;
    }
    employee *e = (employee *) temp;
    menu3:
    {
        const int menu3_height = 18;
        box menu3( menu2_corner_top_left, menu2_width, menu3_height );
        menu3.settcolor(GREEN);
        menu3 << ui::centeralign << "Employee Management" << ui::endl << ui::endl
            ;
        menu3.settcolor(WHITE);
        menu3 << "Edit employee data" << ui::endl;
        menu3.settcolor(ui::tcolor);
        menu3 << "Step 2: Edit employee details" << ui::endl << ui::endl;

        menu3.settcolor_input(YELLOW);
        menu3 << "Name: ";
        menu3.setdefault( e->get_name() );
        menu3 >> name;
        menu3 << "Sex: 1. Male | 2. Female | 3. Transsexual" << ui::endl
              << "---- Enter your choice: ";
        validate_menu::set_menu_limits(1, 3);
        menu3.setdefault( e->get_sex() + 1 );
        menu3 >> validate_menu::input >> (int)sex_choice;
        menu3 << "Date of Birth(DD/MM/YYYY): ";
        sprintf(default_dob_str, "%u/%u/%u", e->get_dob().day, e->get_dob().month
            , e->get_dob().year);
        menu3.setdefault( default_dob_str );
        menu3 >> hospital::date_validity >> dob_str;
        menu3 << "Address: " << ui::endl;
        menu3 << (char)26 << "House no.: ";
        menu3.setdefault( e->get_address().house_no );
        menu3 >> adr_hno;
        menu3 << (char)26 << "Street: ";
        menu3.setdefault( e->get_address().street );
        menu3 >> adr_street;
        menu3 << (char)26 << "City: ";
        menu3.setdefault( e->get_address().city );
        menu3 >> adr_city;
        menu3 << (char)26 << "District: ";
        menu3.setdefault( e->get_address().district );
        menu3 >> adr_dist;
        menu3 << (char)26 << "State: ";
        menu3.setdefault( e->get_address().state );
        menu3 >> adr_state;
        menu3 << "Phone no.: ";
```

```cpp
        menu3.setdefault( e->get_phone() );
        menu3 >> phn_no;
        menu3 << "Salary: ";
        menu3.setdefault( e->get_salary() );
        menu3 >> salary;
        menu3 << "Shift timings: Starts - (HH:MM:SS)";
        sprintf(default_shift_str, "%u:%u:%u", e->get_shift(START).hour, e->get_shift(START).minute, e->get_shift(START).second );
        menu3.setdefault( default_shift_str );
        menu3 >> hospital::time_validity >> shift_start_str;
        menu3 << "-------------: Ends - (HH:MM:SS)";
        sprintf(default_shift_str, "%u:%u:%u", e->get_shift(END).hour, e->get_shift(END).minute, e->get_shift(END).second );
        menu3.setdefault( default_shift_str );
        menu3 >> hospital::time_validity >> shift_end_str;
        menu3.setexit_button("Submit");
        menu3.setback_func(back_func::set_backbit);
        menu3.loop();
        menu3.hide();
    }
    if(back_func::backbit)
    {
        back_func::backbit = 0;
        goto notice;
    }
    --sex_choice;
    dob = hospital::str_to_date(dob_str);
    adr = address(adr_hno, adr_street, adr_city, adr_dist, adr_state);
    shift_start = hospital::str_to_time(shift_start_str);
    shift_end = hospital::str_to_time(shift_end_str);
    e->set_name(name);
    e->set_sex(sex_choice);
    e->set_dob(dob);
    e->set_address(adr);
    e->set_phone(phn_no);
    e->set_salary(salary);
    e->set_shift(START, shift_start);
    e->set_shift(END, shift_end);
    if(id_to_emp::convert(id) == DOCTOR)
    {
        coord c(1, 4);
        doctor *d = (doctor *)temp;
        box menu4(c, (ui::scr_width / 2), ui::scr_height - 5);
        box inp_box(( c + coord((ui::scr_width / 2) - 1, 0)), (ui::scr_width / 2) + 1, ui::scr_height - 5);
        menu4.f << ( ui::top | ui::left ) << (char)204
                  << ( ui::bottom | ui::left ) << (char)204
                  << ( ui::top | ui::right ) << (char)203
                  << ( ui::bottom | ui::right ) << (char)202;
        menu4.f.display();
        inp_box.f << ( ui::top | ui::left ) << (char)203
               << ( ui::bottom | ui::left ) << (char)202
               << ( ui::top | ui::right ) << (char)185
               << ( ui::bottom | ui::right ) << (char)185;
        inp_box.f.display();
        menu4 << ui::centeralign << "Employee Management" << ui::endl << ui::endl;
        menu4.settcolor(WHITE);
        menu4 << "Edit employee data" << ui::endl;
        menu4.settcolor(ui::tcolor);
        menu4 << "Step 3: Edit doctor details" << ui::endl << ui::endl;
        int speciality[2];
        menu4 << "Specialization of doctor (max 2)" << ui::endl
              << "Choose from the following list: " << ui::endl;
        for(int i = 0; i <= GEN; ++i)
        {
            if(i <= 8)
            {
                menu4 << i << ". " << (body_parts)i << ui::endl;
            }
            else
            {
                inp_box << i << ". " << (body_parts)i << ui::endl;
            }
        }
        inp_box.settcolor_input(YELLOW);
        inp_box << "Enter the number corresponding to the required entry in the 2
             fields below" << ui::endl;
        validate_menu::set_menu_limits(BRAIN, GEN);
        inp_box << (char)26;    inp_box.setdefault(d->get_speciality()[0]);
            inp_box >> validate_menu::input >> speciality[0];
        inp_box << (char)26;    inp_box.setdefault(d->get_speciality()[1]);
            inp_box >> validate_menu::input >> speciality[1];
        inp_box << ui::endl;
        inp_box.setexit_button("Submit");
        inp_box.setback_func(back_func::set_backbit);
        inp_box.loop();
        menu4.hide();
        inp_box.hide();
        window.f.display();
        d->set_speciality(speciality);
    }
    if(back_func::backbit)
    {
        back_func::backbit = 0;
        goto menu3;
    }
    const int notice2_height = 12;
    box notice2(menu2_corner_top_left, menu2_width, notice2_height);
    notice2.settcolor(GREEN);
    notice2 << ui::centeralign << "Employee Management" << ui::endl << ui::endl;
    if(!hospital::write_employee(temp))
    {
        notice2.settcolor(RED);
        notice2 << "Employee edit unsuccessful!!";
        notice2.setexit_button("Exit");
        notice2.loop();
        notice2.hide();
        free(temp);
        return;
    }
    notice2 << "Employee edited successfully!!" << ui::endl;
    notice2.settcolor(WHITE);
    notice2 << "Hit the button below to display the details you entered: " << ui
        ::endl;
    notice2.settcolor(ui::tcolor);
    notice2 << ui::endl;
    notice2.setexit_button("View employee...");
    notice2.loop();
    notice2.hide();
    view_emp(id);
    free(temp);
}

void emp_mgmt::pay_emp()
{
    unsigned long id;
    char ch;
    const coord menu2_corner_top_left = coord(ui::scr_width * 0.2, 5);
    const int menu2_width = ui::scr_width * 0.6;
    const int menu2_height = 10;
    menu2:
    {
        box menu2(menu2_corner_top_left, menu2_width, menu2_height);
        menu2.settcolor(GREEN);
        menu2 << ui::centeralign << "Employee Management" << ui::endl << ui::endl
            ;
        menu2.settcolor(WHITE);
        menu2 << "Pay salary to individual employee" << ui::endl;
        menu2.settcolor(ui::tcolor);
        menu2 << "Enter employee's id: ";
        menu2.settcolor_input(YELLOW);
        menu2 >> id;
        menu2 << ui::endl;
        menu2.setexit_button("Submit");
        menu2.setback_func(back_func::set_backbit);
        menu2.loop();
        menu2.hide();
    }
    if(back_func::backbit)
    {
        back_func::backbit = 0;
        return;
    }
    notice:
    {
        const int notice_height = 14;
        box notice(menu2_corner_top_left, menu2_width, notice_height);
        notice.settcolor(GREEN);
        notice << ui::centeralign << "Employee Management" << ui::endl << ui::
            endl;
        notice.settcolor(WHITE);
        notice << "Hit the button below to display the details of the employee
            you want to pay salary to: " << ui::endl;
        notice.settcolor(ui::tcolor);
        notice << ui::endl;
        notice.setexit_button("View employee...");
        notice.setback_func(back_func::set_backbit);
        notice.loop();
        notice.hide();
    }
    if(back_func::backbit)
    {
        back_func::backbit = 0;
        goto menu2;
    }
    if( !view_emp(id) )
    {
        return;
    }
    {
        const int notice2_height = 14;
        box notice2( menu2_corner_top_left, menu2_width, notice2_height );
        notice2.settcolor(GREEN);
        notice2 << ui::centeralign << "Employee Management" << ui::endl << ui::
            endl;
        notice2.settcolor(WHITE);
        notice2 << "Are you sure you want to pay salary to this employee?(y/n): "
             << ui::endl;
        notice2.settcolor_input(YELLOW);
        notice2 >> ch;
        notice2.settcolor(ui::tcolor);
        notice2 << ui::endl;
        notice2.setexit_button("Submit");
        notice2.setback_func(back_func::set_backbit);
        notice2.loop();
        notice2.hide();
    }
    if(back_func::backbit)
    {
        back_func::backbit = 0;
        goto notice;
    }
    if(ch == 'n' || ch == 'N')
    {
        return;
    }
    const int notice3_height = 14;
    box notice3( menu2_corner_top_left, menu2_width, notice3_height );
    notice3.settcolor(GREEN);
    notice3 << ui::centeralign << "Employee Management" << ui::endl << ui::endl;
    notice3.settcolor(RED);
    if( !hospital::pay_salary(id, system::get_date(), system::get_time()) )
    {
        notice3 << "Failed to pay salary to the employee! Check log";
        notice3.setexit_button("Back");
        notice3.loop();
        notice3.hide();
        return;
    }
    notice3.settcolor(GREEN);
    notice3 << "Pay salary successful!!" << ui::endl;
    notice3.setexit_button("Back");
    notice3.loop();
    notice3.hide();
}

void emp_mgmt::pay_all()
{
    char ch;
    const int menu2_height = 11;
    box menu2(coord(ui::scr_width * 0.2, 5), ui::scr_width * 0.6, menu2_height);
    menu2.settcolor(GREEN);
    menu2 << ui::centeralign << "Employee Management" << ui::endl << ui::endl;
    menu2.settcolor(WHITE);
    menu2 << "Pay salary to all employees" << ui::endl;
    menu2.settcolor(ui::tcolor);
    menu2 << "Are you sure you want to pay salary to all employees?(y/n): ";
    menu2.settcolor_input(YELLOW);
    menu2 >> ch;
    menu2 << ui::endl;
    menu2.setexit_button("Submit");
    menu2.loop();
    menu2.hide();
    if(ch == 'n' || ch == 'N')
    {
        return;
    }
    const int notice_height = 10;
    box notice( menu2.getcorner_top_left(), menu2.getwidth(), notice_height ); notice.hide();
    box notice2( notice.getcorner_top_left(), notice.getwidth(), notice.getheight() );
    notice2.settcolor(GREEN);
    notice2 << ui::centeralign << "Employee Management" << ui::endl << ui::endl;
    notice2.hide(); notice.display();
    notice.settcolor(GREEN);
    notice << ui::centeralign << "Employee Management" << ui::endl << ui::endl;
    notice.settcolor(ui::tcolor);
    notice << "Pay all salaries in progress..." << ui::endl;
    if( !hospital::pay_all_salaries() )
    {
        notice.hide();
        notice2.settcolor(RED);
        notice2 << "Failed to pay salary to all employees! Check log";
        notice2.setexit_button("Back");
        notice2.loop();
        notice2.hide();
        return;
    }
    notice.hide();  notice2.display();
    notice2 << "Pay all salaries successful!!" << ui::endl;
    notice2.setexit_button("Back");
    notice2.loop();
    notice2.hide();
}
```

2. code/BASE.CPP

```
#include "base.hpp"

/////////////////////////////////////////////
//// Function definitions for class person

person::person(str inp1, int inp2, Date inp3, address inp4, phone inp5)
{
    strcpy(name, inp1);
    sex = inp2;
    dob = inp3;
    adr = inp4;
    strcpy(phone_no, inp5);
    calc_age();
}

```

```
person::person()
{
    strcpy(name, "");
    dob = Date();
    strcpy(phone_no, "");
}

char* person::get_name()
{
    return name;
}

int person::get_age()
{
    return age;
}

int person::get_sex()
{
    return sex;
}

Date person::get_dob()
{
    return dob;
}

address person::get_address()
{
    return adr;
}

char* person::get_phone()
{
    return phone_no;
}

void person::calc_age(Date dnow)
{
    if(dnow.month > dob.month || dnow.month == dob.month && dnow.day >= dob.day)
    {
        age = dnow.year - dob.year;
    }
    else
    {
        age = dnow.year - dob.year - 1;
    }
}

void person::set_name(char* a)
{
    strcpy(name, a);
}

void person::set_sex(int a)
{
    sex = a;
}

```

```
void person::set_dob(Date bday, Date dnow)
{
    dob = bday;
    calc_age(dnow);
}

void person::set_address(address a)
{
    adr = a;
}

void person::set_phone(char* a)
{
    strcpy(phone_no, a);
}

Time::Time()
{
    hour = 25;
    minute = 0;
    second = 0;
}

Time::Time(unsigned h, unsigned m, unsigned s)
{
    hour = h;
    minute = m;
    second = s;
}

Date::Date()
{
    day = 0;
    month = 0;
    year = 0;
};

Date::Date(unsigned d, unsigned m, unsigned y)
{
    if( d<=31 && m <=12)
    {
        day = d;
        month = m;
        year = y;
    }
    else
    {
        day = 0;
        month = 0;
        year = 0;
    }
}

Date system::get_date()
{
    time_t t = time(0);
    struct tm *now = localtime(&t);
    Date dnow(now->tm_mday, (now->tm_mon + 1) ,(now->tm_year + 1900));
    return dnow;
}

Time system::get_time()
{
    time_t t = time(0);
    struct tm *now = localtime(&t);
    Time tnow(now->tm_hour, now->tm_min ,now->tm_sec);
    return tnow;
}

address::address(const char *hno, const char *strt, const char *cty, const char *
    dist, const char *stat)
{
    strcpy(house_no, hno);
    strcpy(street, strt);
    strcpy(city, cty);
    strcpy(district, dist);
    strcpy(state, stat);
}

userid::userid(str name ,str plaintext) //plaintext is the unencrypted password
{
    strcpy(username, name);
    set_key(plaintext);
    makecipher(plaintext);
}

userid::userid()
{
    strcpy(username, "");
    strcpy(passcipher, "");
}

void userid::makecipher(str plaintext)
{
    int len = strlen(plaintext);
    int keylen = strlen(default_key);
    for(int i = 0; i < len; ++i)
    {
        int plntext_i = (int)plaintext[i] + 127;
        int key_i = (int)default_key[i % keylen] + 127;
        passcipher[i] = (char) ( ( (plntext_i + key_i) % 256 ) - 127);
    }
    passcipher[i] = '\0';
}

void userid::set_key(char * plaintext)
{
    randomize();
    int len = strlen(plaintext);
    int keylen = random(len/2 + 1) + len/2; //so that the key is not too short
    for (int i = 0; i <=keylen; ++i)
    {
        default_key[i] = (char)( random(256) - 127 );
    }
    default_key[i] = '\0';
}

char * userid::decipher()
```

```
{
    str decryptedpass;
    int len = strlen(passcipher);
    int keylen = strlen(default_key);
    for(int i = 0; i < len; ++i)
    {
        int cipher_i = (int)passcipher[i] + 127;
        int key_i = (int)default_key[i % keylen] + 127;
        decryptedpass[i] = (char) ( ( (cipher_i - key_i + 256) % 256 ) - 127);
    }
    decryptedpass[i] = '\0';
    return decryptedpass;
}

char * userid::get_username()
{
    return username;
}

void userid::set_username(char * inp)
{
    strcpy(username, inp);
}

int userid::login(char * password)
{
    if(!strcmp(password, decipher()))
        return 1;
    else
        return 0;
}

transaction::transaction(float a, Date d, Time t, char * b)
{
    amount = a;
    strcpy(reason, b);
    _date = d;
    _time = t;
}

transaction::transaction()
{
    amount = 0;
    strcpy(reason, "NA");
    _date = Date();
    _time = Time();
}

box & operator<<(box &output, sex s)
{
    switch(s)
    {
        case MALE:
            return output << "Male";
        case FEMALE:
            return output << "Female";
        case TRANS:
            return output << "Transsexual";
        default:
            return output << "Invalid";
    }
}

box & operator<<(box &output, body_parts b)
{
    switch(b)
    {
        case BRAIN:
            return output << "Brain";
        case HEART:
            return output << "Heart";
        case SKIN:
            return output << "Skin";
        case LUNG:
            return output << "Lung";
        case BONE:
            return output << "Bone";
        case EYE:
            return output << "Eye";
        case THROAT:
            return output << "Throat";
        case TEETH:
            return output << "Teeth";
        case STOMACH:
            return output << "Stomach";
        case BLOOD:
            return output << "Blood";
        case GUT:
            return output << "Gastrointestinal tract";
        case GEN:
            return output << "General ailments";
        default:
            return output << "Invalid";
    }
}

box & operator<<(box &output, Time & t)
{
    return output << (unsigned long)t.hour << ':' << (unsigned long)t.minute << '
        :' << (unsigned long)t.second;
}

box & operator<<(box &output, Date & d)
{
    return output << (unsigned long)d.day << '/' << (unsigned long)d.month << '/'
         << (unsigned long)d.year;
}

box & operator<<(box &output, address & a)
{
    return output << a.house_no << ", " << a.street << ", "
            << a.city << ", " << a.district << ", " << a.state;
}
```

3. code/HOSP.CPP

```
#include "hosp.hpp"
```

```
#include "iface.hpp"
#include "emp.hpp"
#include <fstream.h>

////////////////////////////////////////////////
////////////////////////////////////////////////// Function definitions for class
    hospital

float hospital::get_balance(){
    return balance;
}

transaction hospital::deduct_money(float amt, char* reason, Date d, Time t){
    hospital::balance -= amt;

    ofstream hosp_finances ("transactions.dat", ios::out | ios::binary | ios::app
        );

    transaction temp = transaction( (-1)*amt, d, t, reason);

    hosp_finances.write( (char*) (&temp) , sizeof(transaction) );

    hosp_finances.close();

    return temp;
}

transaction hospital::add_money(float amt, char* reason, Date d, Time t){
    hospital::balance += amt;

    ofstream hosp_finances ("transactions.dat", ios::out | ios::binary | ios::app
        );

    transaction temp = transaction( (-1)*amt,d, t, reason);

    hosp_finances.write( (char*) (&temp) , sizeof(transaction) );

    hosp_finances.close();

    return temp;
}

transaction* hospital::get_transaction(){
    transaction a[10];

    ifstream hosp_finances ("transactions.dat", ios::in | ios::binary);

    hosp_finances.seekg( (-1) * sizeof(transaction) , hosp_finances.end );

    for(int i = 0; i < 10; i++){
        hosp_finances.read( (char *) &a[i] , sizeof(transaction) );
        hosp_finances.seekg( hosp_finances.tellg() - ( 2 * sizeof(transaction) )
            );
    }

    return a;
}

patient hospital::get_patient_by_id(long id){
    patient a;

    str temp;

    sprintf(temp, "patient/%lu/base.dat", id);

    int i = hospital::read_from(id, (char *) &a, sizeof(patient), temp);

    if(!i){
        interface::error("File read error!!");
        getch();
    }

    return a;

}

void hospital::write_patient(patient a){
    str temp, temp2;
    sprintf(temp, "patient/%lu/base.dat", a.get_id());
    sprintf(temp2, "patient/%lu", a.get_id());
    mkdir("patient");
    mkdir(temp2);
    ofstream patient_file ( temp , ios::out | ios::binary );

    if(patient_file){
        patient_file.write( (char*) &a , sizeof(patient) );
    }
    else{
        interface::error("Patient file access failure!!");
    }
    if(patient_file.fail()){
        interface::error("Patient file write failure!!");
    }
    patient_file.close();
}

void hospital::charge_patient(int pat_id, transaction trans){
    patient temp_pat = hospital::get_patient_by_id(pat_id);

    str temp;
    sprintf(temp, "patient/%d/trans.dat", temp_pat.get_id());
    ofstream patient_file ( temp , ios::out | ios::binary | ios::app );
    patient_file.write( (char*) &trans , sizeof(transaction) );
    patient_file.close();

    hospital::write_patient(temp_pat);
}

void hospital::discharge_patient(patient temp){
    temp.discharge();
    temp.set_discharge_date( system::get_date() );
    hospital::write_patient(temp);
}

float hospital::calc_bill(int stay){
    return stay * ::stay_charge;
}

```

```cpp
medicine hospital::get_med_by_code(int inp_code){
    fstream meds ("stock/med.dat", ios::in | ios::binary);

    medicine temp;

    if(inp_code < 1 || inp_code > 100){
        temp.code = 0;
        temp.price = 0;
        temp.dosage = 0;
        temp.stock = 0;
        strcpy(temp.name, "Shell Medicine");

        interface::error("Invalid medicine code!!");

        return temp;
    }

    for(int i = 0; i<100; i++){
        meds.read((char*) &temp, sizeof(medicine));
        if(temp.code == inp_code){
            break;
        }
    }

    return temp;
}

void hospital::write_med(medicine inp_med){
    fstream med_file ("stock/med.dat", ios::in | ios::out | ios::binary);
    med_file.seekg(0);

    int success = 0;

    while (!success){
        medicine a;
        med_file.read( (char*) &a, sizeof(medicine) );
        if(a.code==inp_med.code){
            med_file.seekg( med_file.tellg() - sizeof(medicine) );
            med_file.write( (char*) &a, sizeof(medicine) );
            success++;
        }
    }

}

int hospital::get_employee_by_id(unsigned long ID, void * target)
{
    if(target == NULL)
    {
        interface::log_this("hospital::get_employee_by_id() : NULL pointer
           supplied to function\nFunction aborted");
        return 0;
    }
    str temp;
    int size_of_target;
    switch(id_to_emp::convert(ID))
    {
        case INVALID:
            interface::log_this("hospital::get_employee_by_id() : Invalid id
                supplied to function\nFunction aborted");
            return 0;
        case OTHERS:
            sprintf(temp, "employee/%lu/base.dat", ID);
            size_of_target = sizeof(employee);
            break;
        case DOCTOR:
            sprintf(temp, "employee/doctor/%lu/base.dat", ID);
            size_of_target = sizeof(doctor);
            break;
        case NURSE:
            sprintf(temp, "employee/nurse/%lu/base.dat", ID);
            size_of_target = sizeof(nurse);
            break;
        case RECEPTIONIST:
            sprintf(temp, "employee/receptionist/%lu/base.dat", ID);
            size_of_target = sizeof(receptionist);
            break;
    }
    int i = hospital::read_from( ID, (char*) target, size_of_target, temp );
    if(!i)
    {
        target = NULL;
        return 0;
    }
    return 1;
}

int hospital::write_employee(void * a)
{
    if(a == NULL)
    {
        interface::log_this("hospital::write_employee() : NULL pointer supplied
            to function\nFunction aborted");
        return 0;
    }
    mkdir("employee");
    str temp;
    int size_of_target;
    employee *x = (employee *) a;
    const unsigned long ID = x->get_id();
    switch(id_to_emp::convert(ID))
    {
        case INVALID:
            interface::log_this("hospital::write_employee() : Object with ID zero
                 cannot be written to file\nFunction aborted");
            return 0;
        case OTHERS:
            sprintf(temp, "employee/%lu", ID);
            size_of_target = sizeof(employee);
            break;
        case DOCTOR:
            mkdir("employee/doctor");
            sprintf(temp, "employee/doctor/%lu", ID);
            size_of_target = sizeof(doctor);
            break;
        case NURSE:
            mkdir("employee/nurse");
            sprintf(temp, "employee/nurse/%lu", ID);
```

```cpp
            size_of_target = sizeof(nurse);
            break;
        case RECEPTIONIST:
            mkdir("employee/receptionist");
            sprintf(temp, "employee/receptionist/%lu", ID);
            size_of_target = sizeof(receptionist);
            break;
    }
    mkdir(temp);
    strcat(temp, "/base.dat");
    ofstream fout ( temp , ios::out | ios::binary);
    if(!fout)
    {
        interface::log_this("hospital::write_employee() : Employee data file could not be created or accessed\nFunction aborted");
        return 0;
    }
    fout.write( (char *) a , size_of_target );
    if(fout.fail())
    {
        interface::log_this("hospital::write_employee() : Error while writing to file (fout.fail())\nFunction aborted");
        return 0;
    }
    return 1;
}

int hospital::pay_salary(unsigned long id, Date d1, Time t1)
{
    void * e = malloc( sizeof(doctor) );
    if(e == NULL)
    {
        interface::log_this("hospital::pay_salary() : Not enough memory to allocate buffer void * temp = malloc( sizeof(doctor) )");
        interface::error("Out of memory!! Check log");
        getch();
        return 0;
    }
    str temp;
    switch(id_to_emp::convert(id))
    {
        case INVALID:
            interface::log_this("hospital::pay_salary() : Invalid id supplied to function\nFunction aborted");
            return 0;
        case OTHERS:
            sprintf(temp, "employee/%lu/trans.dat", id);
            break;
        case DOCTOR:
            sprintf(temp, "employee/doctor/%lu/trans.dat", id);
            break;
        case NURSE:
            sprintf(temp, "employee/nurse/%lu/trans.dat", id);
            break;
        case RECEPTIONIST:
            sprintf(temp, "employee/receptionist/%lu/trans.dat", id);
            break;
    }
    if(!hospital::get_employee_by_id(id, e))
    {
        interface::log_this("hospital::pay_salary() : Employee not found or error while reading file\nFunction aborted");
        free(e);
        return 0;
    }
    unsigned long inp1;
    char inp2[100] = "Salary paid to ";
    employee * emp = (employee *)e;
    inp1 = emp->get_salary();
    strcat(inp2, emp->get_name());
    transaction t = hospital::deduct_money(inp1, inp2, d1, t1);
    free(e);

    ofstream fout ( temp ,ios::binary | ios::app );
    if(!fout)
    {
        interface::log_this("hospital::pay_salary() : Employee data file could not be created or accessed\nFunction aborted");
        return 0;
    }
    fout.write((char *) &t, sizeof(transaction));
    if(fout.fail())
    {
        interface::log_this("hospital::pay_salary() : Error while writing to file (fout.fail())\nFunction aborted");
        return 0;
    }
    return 1;
}

int hospital::pay_all_salaries()
{
    Date d1 = system::get_date();
    Time t1 = system::get_time();
    unsigned long max_id;
    ifstream fin;
    fin.open("employee/max_id.dat", ios::binary);
    if(!fin)
    {
        interface::log_this("hospital::pay_all_salaries() : No employees found or cannot access file max_id.dat\nFunction aborted");
        return 0;
    }
    else
    {
        fin.read((char *) &max_id, sizeof(unsigned long));
        if(fin.fail())
        {
            interface::log_this("hospital::pay_all_salaries() : Error while reading file max_id.dat(fin.fail())\nFunction aborted");
            return 0;
        }
        if(!employee::get_generate_id_status())
        {   //if generate_id_status is zero, then no id is generated after max_id + 1
            //Thus, the following loop should run max_id + 1 times
            ++max_id;
        }
        for(unsigned long i = 1; i <= max_id; ++i)
        {
            int a = hospital::pay_salary(i, d1, t1);
            if(!a)
            {
                str log_msg;
                sprintf(log_msg, "hospital::pay_all_salaries() : Failed to pay
                    salary of id %lu...\nSkipped", i);
                interface::log_this(log_msg);
            }
        }
    }
    return 1;
}

int hospital::get_date_difference(Date dt1, Date dt2)
{

    long int n1 = dt1.year*365 + dt1.day;

    for (int i=0; i<dt1.month - 1; i++){
        n1 += monthDays[i];
    }
    n1 += hospital::count_leap_years(dt1);

    long int n2 = dt2.year*365 + dt2.day;

    for (i=0; i<dt2.month - 1; i++){
            n2 += monthDays[i];
    }
    n2 += count_leap_years(dt2);

    return (n2 - n1);
}

int hospital::count_leap_years(Date d)
{
    int years = d.year;

    if (d.month <= 2){
        years--;        // checking whether to count the current year
    }

    return (years / 4) - (years / 100) + (years / 400);
}

int hospital::date_validity(const char * inp_date){
    return hospital::date_validity(hospital::str_to_date(inp_date));
}

int hospital::date_validity(Date inp_date){
     if(inp_date.year % 4 == 0 && inp_date.month == 2 &&
        inp_date.day == 29){
            return 1;
    }
    if (
        inp_date.month > 12 ||
        inp_date.day > monthDays[inp_date.month - 1])
    {
        return 0;
    }
    else{
        return 1;
    }
}

int hospital::time_validity(const char * inp_time)
{
    return time_validity( str_to_time(inp_time) );
}

int hospital::time_validity(Time t)
{
    if( t.hour > 24 || t.minute > 59 || t.second > 59)
    {
        return 0;
    }
    return 1;
}

Date hospital::str_to_date(const char * inp_date){
    int counter = 0;
    int count = 0;
    int input[3];
    input[0] = input[1] = input[2] = 0;
    while(counter < 3){
        char ch[12];
        ch[0] = '/';
        for(int i = 1; i < 7; i++){
            ch[i] = inp_date[count];
            count++;
            if(ch[i] == '/' || ch[i] == '\\' || ch[i] == 0 || ch[i] == '-'){
                if(ch[i] == 0 && count < 11){
                    interface::error("Invalid date!");
                    return Date (99, 99, 9999);
                }
                ch[i] = '/';
                int temp = i-1, temp2 = 0;
                while(ch[temp] != '/'){
                    input[counter] += ( pow(10, temp2) * ((int)ch[temp] - (int)'0
                        ') );
                    temp--;
                    temp2++;
                }
                counter++;
            }
        }
    }

    return Date(input[0], input[1], input[2]);
}

Time hospital::str_to_time(const char * inp_time)
{
///////In this function invalid time(25:00:00) is returned if time is in
    incorrect format//////////
    char inp[3][3] = {"25", "0", "0"};
    int inp_x = 0, inp_y = 0;
    Time null(25, 0, 0);
    if( strlen(inp_time) > 8 || strlen(inp_time) < 5 || inp_time[strlen(inp_time)
        - 1] == ':')
    {
        return null;
    }
    for(int i = 0; i < strlen(inp_time); ++i)
    {
        if(inp_time[i] == ':' && inp_y != 0)
        {
            inp[inp_x][inp_y] = '\0';
            ++inp_x;
            inp_y = 0;
            continue;
        }
        else if( (inp_y == 0 && inp_time[i] == ':') || inp_y > 1
                || (inp_time[i] < '0' || inp_time[i] > '9') )
        {
            return null;
        }
        inp[inp_x][inp_y] = inp_time[i];
        ++inp_y;
    }
    char *endptr;
    null.hour = (unsigned int) strtol(inp[0], &endptr, 10);
    null.minute = (unsigned int) strtol(inp[1], &endptr, 10);
    null.second = (unsigned int) strtol(inp[2], &endptr, 10);
    return null;
}

int hospital::str_to_sex(char* s){
    if( strcmp(s, "M")  )   { return 0; }
    else if(    strcmp(s, "F")  )   { return 1; }
    else { return 2; }
}

int hospital::read_from(unsigned long ID, char * dest, int size, char * temp)
{
    ifstream fin ( temp , ios::in | ios::binary );
    if(!fin)
    {
        char errmsg[200];
        sprintf(errmsg, "hospital::read_from() : Employee with id %lu not found\
            nFunction aborted", ID);
        interface::log_this(errmsg);
        return 0;
    }
    fin.read( dest, size );
    if(fin.fail())
    {
        interface::log_this("hospital::read_from() : Error while reading from
            file (fin.fail())\nFunction aborted");
        return 0;
    }
    fin.close();
    return 1;
}

////////////////////////////////////////////////////////////
```

```
/////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////

double hospital::balance = 10000000.0;
```

4. code/MIAN.CPP

```
#include "iface.hpp"
#include <conio.h>
#include "hosp.hpp"
#include "emp.hpp"

void main()
{
    clrscr();
/*///////////////Administrator object creator///////////
    address yay("", "", "", "", "");
    employee x("Administrator", 3, Date(), yay, "", 0, Time(), Time(), "admin", "
        password");
    hospital::write_employee(&x);
////////////////////////////////////////////////////////*/

    interface::log_this("Program initiated\n\n");

    interface::init();

    interface::log_this("Program terminated\n\n");
}
```

5. code/iface.cpp

```
#include <fstream.h>
#include "base.hpp"
#include "iface.hpp"
#include "hosp.hpp"
#include "emp.hpp"

////////////////////////////////////////////
//////// Function definitions for interface

void interface::stock_management(){
    coord c(ui::scr_width / 3, ui::scr_height / 3);
    box menu (c, ui::scr_width / 3, ui::scr_height / 2.2);

    int ch = 0;

    menu << "1. Sale"
            << ui::endl << "2. Purchase"
            << ui::endl << "3. Stock check"
            << ui::endl << "4. Go to main menu"
            << ui::endl << ui::endl << "Choice : ";
    menu.setdefault(1);
    menu.settcolor_input(YELLOW);
    validate_menu::set_menu_limits(1, 4);
    menu >> validate_menu::input >> ch;

    menu << ui::endl;
    menu.setexit_button("Submit");

    menu.loop();
    menu.hide();

    interface::clear_error();

    switch(ch){
        case 1:
        {

            medicine temp;
            temp.code = 0;

            while(temp.code == 0){
                coord c(ui::scr_width / 3, ui::scr_height / 3);
                box sale_menu (c, ui::scr_width / 3, ui::scr_height / 3);
                sale_menu.settcolor_input(YELLOW);
                sale_menu << ui::centeralign << "Medicine Sale" << ui::endl;
                sale_menu << "Code : ";
                sale_menu.setdefault(42);
                sale_menu >> temp.code;
                sale_menu << ui::endl;
                sale_menu.setexit_button("Submit");
                sale_menu.loop();
                sale_menu.hide();

                temp = hospital::get_med_by_code(temp.code);
            }

            int quantity = -2;
            patient temp_patient;
            long pat_id;

            while(quantity < 0 || quantity > 100){
                coord c(ui::scr_width / 3, ui::scr_height / 3);
                box sale_menu (c, ui::scr_width / 3, ui::scr_height / 2);
                sale_menu.settcolor_input(YELLOW);
                sale_menu << ui::centeralign << "Medicine Sale" << ui::endl;
                sale_menu << "Name : " << temp.name
                          << ui::endl << "Price : $" << temp.price
                          << ui::endl << ui::endl
                          << "Patient ID : ";
                sale_menu.setdefault(786);
                sale_menu >> pat_id;
                sale_menu << ui::endl << "Quantity : ";
                sale_menu.setdefault(1);
                sale_menu >> quantity;
                sale_menu.setexit_button("Submit");
                sale_menu.loop();
                sale_menu.hide();

                temp_patient = hospital::get_patient_by_id(pat_id);
                if(temp_patient.get_id() == 0){
                    quantity = -1;
                    interface::error("Invalid patient ID!!");
                    continue;
                }
                interface::error("Invalid quantity!!");
            }

            interface::clear_error();

            temp.stock -= quantity;

            for(int i = 0; i < 50; i++){
                if(temp_patient.get_med(i, 0) == temp.code ||
                        temp_patient.get_med(i,0) == 0){
                            temp_patient.set_med(i, temp.code, temp_patient.
                                get_med(i, 1) + quantity);
                        }
            }

            hospital::write_patient(temp_patient);
            hospital::write_med(temp);

            break;
        }

        case 2:
        {
            medicine temp;
            temp.code = 0;

            while(temp.code == 0){
                coord c(ui::scr_width / 3, ui::scr_height / 3);
                box purchase_menu (c, ui::scr_width / 3, ui::scr_height / 3);
                purchase_menu.settcolor_input(YELLOW);
                purchase_menu << ui::centeralign << "Medicine Purchase" << ui::
                    endl;
                purchase_menu << "Code : ";
                purchase_menu.setdefault(42);
                purchase_menu >> temp.code;
                purchase_menu << ui::endl;
                purchase_menu.setexit_button("Submit");
                purchase_menu.loop();
                purchase_menu.hide();

                temp = hospital::get_med_by_code(temp.code);
            }

            int quantity = -2;

            while(quantity < 0 || quantity > 5000){
                coord c(ui::scr_width / 3, ui::scr_height / 3);
                box purchase_menu (c, ui::scr_width / 3, ui::scr_height / 2);
                purchase_menu.settcolor_input(YELLOW);
                purchase_menu << ui::centeralign << "Medicine Purchase" << ui::
                    endl;
                purchase_menu << "Name : " << temp.name
                            << ui::endl << "Price : $" << temp.price
                            << ui::endl << ui::endl << "Quantity : ";
                purchase_menu.setdefault(1);
                purchase_menu >> quantity;
                purchase_menu.setexit_button("Submit");
                purchase_menu.loop();
                purchase_menu.hide();

                interface::error("Invalid quantity!!");
            }

            interface::clear_error();

            temp.stock += quantity;
            hospital::deduct_money(temp.price * quantity, "Medicine purchase",
                system::get_date(), system::get_time());
            hospital::write_med(temp);

            break;
        }

        case 3:
        {
            medicine temp;
            temp.code = 0;

            while(temp.code == 0){
                coord c(ui::scr_width / 3, ui::scr_height / 3);
                box stock_menu (c, ui::scr_width / 3, ui::scr_height / 3);
                stock_menu.settcolor_input(YELLOW);
                stock_menu << ui::centeralign << "Stock check" << ui::endl;
                stock_menu << "Code : ";
                stock_menu.setdefault(42);
                stock_menu >> temp.code;
                stock_menu << ui::endl;
                stock_menu.setexit_button("Submit");
                stock_menu.loop();
                stock_menu.hide();

                temp = hospital::get_med_by_code(temp.code);
            }

            coord c(ui::scr_width / 3, ui::scr_height / 3);
            box stock_menu (c, ui::scr_width / 3, ui::scr_height / 2);
            stock_menu.settcolor_input(YELLOW);
            stock_menu << ui::centeralign << "Medicine Details" << ui::endl;
            stock_menu << "Name : " << temp.name
                        << ui::endl << "Price : $" << temp.price
                        << ui::endl << "Dosage : " << temp.dosage << " ml"
                        << ui::endl << "Quantity in stock : " << temp.stock
                        << ui::endl;
            stock_menu.setexit_button("Okay");
            stock_menu.loop();
            stock_menu.hide();

            break;
        }
    }

}

int interface::validate_menu::input(const char * ch)
{
    char *endptr;
    int a = (int) strtol(ch, &endptr, 10);
    if(!validation::vint(ch) || a < lowest_choice || a > greatest_choice)
    {
        return 0;
    }
    else
    {
        return 1;
    }
}

void interface::validate_menu::set_menu_limits(int a, int b)
{
    lowest_choice = a;
    greatest_choice = b;
}

int interface::validate_menu::lowest_choice = 0;
int interface::validate_menu::greatest_choice = 0;

int interface::back_func::set_backbit()
{
    backbit = 1;
    return 1;
}

int interface::back_func::backbit = 0;

void interface::error(char* err){
    window.clear_footer();
    window.setfooter_tcolor(RED);
    window << box::setfooter << ui::centeralign
        << err;
}

void interface::clear_error(){
    window.clear_footer();
    window.setfooter_tcolor(GREEN);
    window << box::setfooter << ui::centeralign
        << "Everything looks OK";
}

int interface::log_this(char * message)
{
    Date dnow = system::get_date();
    Time tnow = system::get_time();
    char text[300];
    sprintf(text, "$ [%u-%u-%u %u:%u:%u +0530]: ", dnow.day, dnow.month, dnow.
        year, tnow.hour, tnow.minute, tnow.second);
    strcat(text, message);
    ofstream fout;
    fout.open("log.txt", ios::out | ios::app);
    if(!fout)
        return 0;
    fout << text << endl;
    if(fout.fail())
        return 0;
    fout.close();
    return 1;
}

interface::interface(){}
```

```

box interface::window;
```

6. code/iface2.cpp

```
#include <fstream.h>
#include "base.hpp"
#include "iface.hpp"
#include "hosp.hpp"
#include "emp.hpp"

void interface::init(){
    window.hide();
    window.display();
    window.settcolor(WHITE);
    window << ui::centeralign << "LHOSPITAL";
    window.settcolor(ui::tcolor);
    window.setfooter_tcolor(GREEN);

    Date current_date = system::get_date();
    Time current_time = system::get_time();

    str curr_date, curr_time;
    sprintf(curr_date, "%d/%d/%d", current_date.day, current_date.month,
        current_date.year);
    sprintf(curr_time, "%d:%d", current_time.hour, current_time.minute);

    window << box::setheader << curr_date << box::setheader << ui::rightalign
            << curr_time << box::setfooter << ui::centeralign
            << "Everything looks OK";
    int id;
    do
    {
        id = interface::login_screen();
        if(id && id_to_emp::convert(id) != OTHERS || id == 1)   //so that general
             employees (except administrator) do
        {                                                       // not
            accidentally login(as they have been assigned
            interface::clear_error();                           // username and
                password as "", "")
            break;
        }
    }while(1);
    if(id == 1) //if user logging in is administrator
    {
        int choice = 0;

        while(1){
            choice = interface::menu();

            switch(choice){
                case 1:
                    interface::employee_management();
                    break;
                case 2:
                    interface::stock_management();
                    break;
                case 3:
```

```cpp
                    return;
                }
            }
        }
        else
        {
            switch(id_to_emp::convert(id))
            {
                case INVALID:
                    interface::error("You have an invalid id generated. Create a new account");
                    break;
                case DOCTOR:
                case NURSE:
                case RECEPTIONIST:
                    interface::employee_screen(id);
                    break;
            }
        }
}

int interface::login_screen()
{
    const int login_screen_height = 9;
    coord c(ui::scr_width / 3, ui::scr_height / 3);
    box login_box (c, ui::scr_width / 3, login_screen_height);

    str uid, pwd;

    login_box.settcolor_input(YELLOW);
    login_box << "User ID : ";
    login_box >> uid;
    login_box << ui::endl << "Password : ";
    login_box >> box::setpassword >> pwd;
    login_box << ui::endl;
    login_box.setexit_button("Login");
    login_box.loop();
    login_box.hide();
    unsigned long max_id;
    ifstream fin;
    fin.open("employee/max_id.dat", ios::binary);
    if(!fin)
        max_id = 1;
    else
    {
        fin.read((char *) &max_id, sizeof(unsigned long));
        if(fin.fail())
        {
            interface::error("ERROR WHILE READING FROM FILE!!! ");
            getch();
            return 0;
        }
    }
    fin.close();
    void * x = malloc( sizeof(doctor) );
    for(unsigned long id = 1; id <= max_id; ++id)
    {
        if(x == NULL)
        {
            interface::log_this("interface::login_screen() : Not enough memory to allocate buffer void * temp = malloc( sizeof(doctor) )");
            interface::error("Out of memory!! Check log");
            getch();
            return 0;
        }
        if(!hospital::get_employee_by_id(id, x))
        {
            char log_msg[300];
            sprintf(log_msg, "interface::login_screen() : Error in reading file of id %lu (hospital::get_employee_by_id(id, x) returned 0), could be due to invalid login details entered", id);
            interface::log_this(log_msg);
        }
        employee * e = (employee *)x;
        if(!strcmp(e->account.get_username(), uid) && e->account.login(pwd))
        {
            interface::clear_error();
            free(x);
            return id;
        }
    }
    interface::error("Invalid login details!!");
    free(x);
    return 0;
}

int interface::menu(){
    coord c(ui::scr_width / 3, ui::scr_height / 3);
    box menu (c, ui::scr_width / 3, ui::scr_height / 2.2 + 1);

    int ch;
    menu << ui::endl << "1. Employee management"
         << ui::endl << "2. Stock management"
         << ui::endl << "3. Exit"
         << ui::endl << ui::endl << "Choice : ";
    menu.settcolor_input(YELLOW);
    validate_menu::set_menu_limits(1, 3);
    menu >> validate_menu::input >> ch;

    menu << ui::endl;
    menu.setexit_button("Submit");

    menu.loop();
    menu.hide();

    return ch;
}

void interface::patient_management(){
    int ch = 0;

    coord c(ui::scr_width / 3, ui::scr_height / 3);
    box menu (c, ui::scr_width / 3, ui::scr_height / 2.2);

    menu << "1. Patient admission"
            << ui::endl << "2. Patient discharge"
            << ui::endl << "3. Edit patient details"
            << ui::endl << "4. Go to main menu"
            << ui::endl << ui::endl << "Choice : ";
    menu.setdefault(1);
    menu.settcolor_input(YELLOW);
    validate_menu::set_menu_limits(1,4);
    menu >> validate_menu::input >> ch;

        menu << ui::endl;
        menu.setexit_button("Submit");

        menu.loop();
        menu.hide();

    switch(ch){
        case 1:
        {
            coord c(ui::scr_width / 4, ui::scr_height / 4);
            box form (c, ui::scr_width / 2, ui::scr_height / 1.5);
            form.settcolor_input(YELLOW);

            str inp_name, inp_sex_str, inp_dob_str
                , inp_phone, inp_guard_name, inp_emer_contact
                , inp_emer_phone, inp_insur_expiry, inp_admdate_str;

            address inp_adr;
            disease inp_dis;
            insurance inp_insur;

            form << "Enter data for the patient :" << ui::endl
                    << ui::endl << "Name : ";
            form >> inp_name;

            form << ui::endl << "Sex : ";
            form >> inp_sex_str;
            form << ui::endl << "Key - M/F/T = Male/Female/Trans"
                    << ui::endl << "Date of Birth : ";

            form.setdefault("25/12/1991");
            form >> inp_dob_str;


            form << ui::endl << "Address"
                    << ui::endl << ui::endl
                    << "\tHouse # : ";
            form.setdefault("221B");
            form >> inp_adr.house_no;

            form << ui::endl << "\tStreet : ";
            form.setdefault("Baker Street");
            form >> inp_adr.street;

            form << ui::endl << "\tDistrict : ";
            form.setdefault("Idk");
            form >> inp_adr.district;

            form << ui::endl << "\tState : ";
            form.setdefault("London(?)");
            form >> inp_adr.state;


            form << ui::endl << ui::endl
                    << "Phone : ";
            form.setdefault("1234567890");
            form >> inp_phone;


            form << ui::endl << "Disease"
                    << ui::endl << ui::endl
                    << "\tName : ";
            form.setdefault("Melanoma");
            form >> inp_dis.name;

            form << ui::endl << "Type : ";
            form.setdefault(0);
            form >> inp_dis.type;

            form << ui::endl << "\tType key : " << ui::endl
                    << "\t0 – Brain\t1 – Heart" << ui::endl
                    << "\t2 – Skin\t3 – Lung" << ui::endl
                    << "\t4 – Bone\t5 – Eye" << ui::endl
                    << "\t6 – Throat\t7 – Teeth" << ui::endl
                    << "\t8 – Stomach\t9 – Blood" << ui::endl
                    << "\t10 – General/full body condition"
                    << ui::endl << "\tSymptoms"
                    << ui::endl << "\tSymptom 1 : ";

            form >> inp_dis.symptoms[0];

            form << ui::endl << "\tSymptom 2 : ";
            form >> inp_dis.symptoms[1];

            form << ui::endl << "\tSymptom 3 : ";
            form >> inp_dis.symptoms[2];

            form << ui::endl << "\tSymptom 4 : ";
            form >> inp_dis.symptoms[3];


            form << ui::endl << ui::endl
                    << "Guardian name : ";
            form.setdefault("Dr. John Watson");
            form >> inp_guard_name;

            form << ui::endl << "Emergency Contact : ";
            form.setdefault("Irene Adler");
            form >> inp_emer_contact;

            form << ui::endl << "Emer. Cont. Phone : ";
            form.setdefault("1234567890");
            form >> inp_emer_phone;


            form << ui::endl << "Insurance"
                    << ui::endl << ui::endl
                    << "\tProvider : ";
            form.setdefault("LIC");
            form >> inp_insur.provider;

            form << ui::endl << "\tAmount ($) : ";
            form.setdefault(30000);
            form >> inp_insur.amount;

            form << ui::endl << "\tExpiry";
            form.setdefault("25/12/2022");
            form >> inp_insur_expiry;


            form << ui::endl << ui::endl
                    << "Admission Date : ";
            char dnow[11];
            form.setdefault("01/01/2018");
            form >> inp_admdate_str;

            form << ui::endl << ui::endl;
            form.setexit_button("Submit");

            form.loop();

            form.hide();

            inp_insur.expiry = hospital::str_to_date(inp_insur_expiry);

            patient temp_pat = patient(inp_name, hospital::str_to_sex(inp_sex_str
                )
                                        , hospital::str_to_date(inp_dob_str),
                                          inp_adr
                                        , inp_phone, inp_dis, inp_guard_name
                                        , inp_emer_contact, inp_emer_phone
                                        , inp_insur, hospital::str_to_date(
                                          inp_admdate_str));

            hospital::write_patient(temp_pat);

            coord d(ui::scr_width / 3, ui::scr_height / 3);
            box message (d, ui::scr_width / 3, ui::scr_height / 3);

            message << ui::centeralign << "Patient has been admitted with ID #"
                    << temp_pat.get_id() << ui::endl << ui::endl;

            message.setexit_button("Okay");
            message.loop();
            message.hide();

            break;
        }

        case 2:
        {
            patient temp_patient;

            while(1){
                coord c(ui::scr_width / 3, ui::scr_height / 3);
                box login_box (c, ui::scr_width / 3, ui::scr_height / 2.5);

                long inp_pat_id;

                login_box << ui::endl << "Patient Discharge"
                            << ui::endl << "Enter patient ID : ";
                login_box.setdefault(1);
                login_box >> inp_pat_id;

                login_box << ui::endl;
                login_box.setexit_button("Submit");

                login_box.loop();

                login_box.hide();

                temp_patient = hospital::get_patient_by_id(inp_pat_id);

                if(temp_patient.get_id() == inp_pat_id){
                    break;
                    interface::clear_error();
                }
                else{
                    interface::error("Invalid Patient ID!!");
                    continue;
                }
            }

            coord c(ui::scr_width / 3, ui::scr_height / 3);
            box bill (c, ui::scr_width / 3, ui::scr_height / 2);

            str tt;
            sprintf(tt, "%d/%d/%d", temp_patient.get_admission_date(DAY),
                                                temp_patient.get_admission_date(MONTH),
                                                temp_patient.get_admission_date(YEAR));

                                                interface::log_this(tt);

            int stay_len = abs( hospital::get_date_difference(
                                            system::get_date(),
                                            Date(
                                                temp_patient.get_admission_date(DAY),
                                                temp_patient.get_admission_date(MONTH),
                                                temp_patient.get_admission_date(YEAR)
                                            )
                            ) );

            bill << ui::endl << "Bill for " << temp_patient.get_name()
                    << ui::endl << "1. Stay for "
                    << stay_len << " days"  << ui::endl;

            float total_bill;
            bill.settcolor(GREEN);
            bill << "$" << ( total_bill += hospital::calc_bill(stay_len) );

            for(int i = 0; i < 50; i++){
                    transaction temp_trans = temp_patient.get_transaction(i);

                    if( temp_trans.amount == 0 ){
                        break;
                    }

                    bill << i+2 << ". " << temp_trans.reason << ui::endl;
                    bill.settcolor(GREEN);
                    bill << "\t$" << temp_trans.amount << ui::endl;
                    bill.settcolor(ui::tcolor);

                    total_bill += temp_trans.amount;
            }

            bill.settcolor(CYAN);
            bill << ui::endl << "Final bill : $" << total_bill;
            bill.settcolor(ui::tcolor);
            bill.setexit_button("Pay Bill");
            bill.loop();
            bill.hide();

            hospital::discharge_patient(temp_patient);

            break;
        }

        case 3:
        {
            int choice = 0;

            patient temp_patient;

            while(1){
                coord c(ui::scr_width / 3, ui::scr_height / 3);
                box login_box (c, ui::scr_width / 3, ui::scr_height / 2.5);
                login_box.settcolor_input(YELLOW);

                long inp_pat_id;

                login_box << ui::endl << "Patient Data Alteration"
                            << ui::endl << "Enter patient ID : ";
                login_box.setdefault(1);
                login_box >> inp_pat_id;

                login_box << ui::endl;
                login_box.setexit_button("Submit");

                login_box.loop();

                login_box.hide();

                temp_patient = hospital::get_patient_by_id(inp_pat_id);

                if(temp_patient.get_id() == inp_pat_id){
                    break;
                    interface::clear_error();
                }
                else{
                    interface::error("Invalid Patient ID!!");
                    continue;
                }
            }

            while(choice < 1 || choice > 5){
                coord c(ui::scr_width / 3, ui::scr_height / 3);
                box menu (c, ui::scr_width / 3, ui::scr_height / 1.5);

                menu << "Choose item to edit:"
                        << ui::endl << "1. Disease/condition"
                        << ui::endl << "2. Guardian name"
                        << ui::endl << "3. Emergency contact"
                        << ui::endl << "4. Emergency contact no."
                        << ui::endl << "5. Insurance information"
                        << ui::endl << ui::endl << "Choice : ";
                menu.setdefault(1);
                menu.settcolor_input(YELLOW);
                menu >> choice;

                menu << ui::endl;
                menu.setexit_button("Submit");

                menu.loop();
                menu.hide();
            }
            switch(choice){
                case 1:
                {
                    coord c(ui::scr_width / 3, ui::scr_height / 3);
                    box edit_screen (c, ui::scr_width / 3, ui::scr_height / 2);
                    edit_screen.settcolor_input(YELLOW);

                    edit_screen <<  "Enter disease/condition for " << temp_patient.get_name()
                                << ui::endl << "Disease : ";
                    disease temp = temp_patient.get_dis();
                    edit_screen.setdefault(temp.name);
                    edit_screen >> temp.name;
                    edit_screen << ui::endl << "Type : ";
                    edit_screen.setdefault(temp.type);
                    edit_screen >> temp.type;
                    edit_screen << ui::endl << "Type key : " << ui::endl
                                << "0 - Brain\t1 - Heart" << ui::endl
                                << "2 - Skin\t3 - Lung" << ui::endl
                                << "4 - Bone\t5 - Eye" << ui::endl
                                << "6 - Throat\t7 - Teeth" << ui::endl
                                << "8 - Stomach\t9 - Blood" << ui::endl
                                << "10 - General/full body condition"
                                << ui::endl << ui::endl
                                << "Symptoms" << ui::endl
                                << "Symptom 1 : ";
                    edit_screen.setdefault(temp.symptoms[0]);
                    edit_screen >> temp.symptoms[0];
                    edit_screen << ui::endl << "Symptom 2 : ";
                    edit_screen.setdefault(temp.symptoms[1]);
                    edit_screen >> temp.symptoms[1];
                    edit_screen << ui::endl << "Symptom 3 : ";
                    edit_screen.setdefault(temp.symptoms[2]);
                    edit_screen >> temp.symptoms[2];
                    edit_screen << ui::endl << "Symptom 4 : ";
                    edit_screen.setdefault(temp.symptoms[3]);
                    edit_screen >> temp.symptoms[3];

                    edit_screen << ui::endl << ui::endl;
                    edit_screen.setexit_button("Submit");

                    edit_screen.loop();

                    edit_screen.hide();

                    temp_patient.set_dis(temp);
                    hospital::write_patient(temp_patient);

                    break;
                }

                case 2:
                {
                    coord c(ui::scr_width / 3, ui::scr_height / 3);
                    box edit_screen (c, ui::scr_width / 3, ui::scr_height / 2);
                    edit_screen.settcolor_input(YELLOW);

                    edit_screen <<  "Enter name of guardian for " << temp_patient
                        .get_name()
                                << ui::endl << "Guardian Name : ";
                    str temp;
                    edit_screen.setdefault(temp_patient.get_guardian_name());
                    edit_screen >> temp;

                    edit_screen << ui::endl << ui::endl;
                    edit_screen.setexit_button("Submit");

                    edit_screen.loop();

                    edit_screen.hide();

                    temp_patient.set_guardian_name(temp);
                    hospital::write_patient(temp_patient);

                    break;
                }

                case 3:
                {
                    coord c(ui::scr_width / 3, ui::scr_height / 3);
                    box edit_screen (c, ui::scr_width / 3, ui::scr_height / 2);
                    edit_screen.settcolor_input(YELLOW);

                    edit_screen <<  "Enter emergency contact no. for " <<
                        temp_patient.get_name()
                                << ui::endl << "Contact no. : ";
                    str temp;
                    edit_screen.setdefault(temp_patient.get_emergency_contact());
                    edit_screen >> temp;

                    edit_screen << ui::endl << ui::endl;
                    edit_screen.setexit_button("Submit");

                    edit_screen.loop();

                    edit_screen.hide();

                    temp_patient.set_emergency_contact(temp);
                    hospital::write_patient(temp_patient);

                    break;
                }

                case 4:
                {
                    coord c(ui::scr_width / 3, ui::scr_height / 3);
                    box edit_screen (c, ui::scr_width / 3, ui::scr_height / 2);
                    edit_screen.settcolor_input(YELLOW);

                    edit_screen <<  "Enter emergency contact no. for " <<
                        temp_patient.get_name()
                                << ui::endl << "Contact no. : ";
                    phone temp;
                    edit_screen.setdefault(temp_patient.get_emer_contact_no());
                    edit_screen >> temp;

                    edit_screen << ui::endl << ui::endl;
                    edit_screen.setexit_button("Submit");

                    edit_screen.loop();

                    edit_screen.hide();

                    temp_patient.set_emer_contact_no(temp);
                    hospital::write_patient(temp_patient);

                    break;
                }

                case 5:
                {
                    coord c(ui::scr_width / 3, ui::scr_height / 3);
                    box edit_screen (c, ui::scr_width / 3, ui::scr_height / 2);
                    edit_screen.settcolor_input(YELLOW);

                    edit_screen <<  "Enter insurance information for " <<
                        temp_patient.get_name()
                                << ui::endl << "Provider : ";
                    insurance temp = temp_patient.get_insur_info();
                    edit_screen.setdefault(temp.provider);
                    edit_screen >> temp.provider;
                    edit_screen << ui::endl << "Amount (in $) :";
                    edit_screen.setdefault(temp.amount);
                    edit_screen >> temp.amount;
                    edit_screen << ui::endl << "Expiry date (DD/MM/YYYY):";
                    char temp_date[11];
                    edit_screen >> hospital::date_validity >> temp_date;

                    edit_screen << ui::endl << ui::endl;
                    edit_screen.setexit_button("Submit");

                    edit_screen.loop();

                    edit_screen.hide();

                    temp.expiry = hospital::str_to_date(temp_date);
                    temp_patient.set_insur_info(temp);
                    hospital::write_patient(temp_patient);

                    break;
                }

            }

            break;
        }
        case 4:
        {
            break;
        }
    }
}
```

7. code/EMP.CPP

```
#include "hosp.hpp"
#include "iface.hpp"
#include "emp.hpp"
#include "base.hpp"
#include <fstream.h>

//////////////////////////////////
/// Function definitions for class employee

int employee::generate_id()
{
    mkdir("employee");
    unsigned long max_id;
    ifstream fin;
    fin.open("employee/max_id.dat", ios::binary);
    if(!fin)
    {
        interface::log_this("employee::generate_id() : File max_id.dat not found
            or error while loading file\nmax_id will be set to zero");
        max_id = 0;
    }
    else
    {
        fin.read((char *) &max_id, sizeof(unsigned long));
        if(fin.fail())
        {
            interface::log_this("employee::generate_id() : Error while reading
                from file max_id.dat (fin.fail())\nFunction aborted");
            id = 0;
            return 0;
        }
    }
    fin.close();
    ++max_id;
    id = max_id;
    ofstream fout;
    fout.open("employee/max_id.dat", ios::binary);
    fout.write((char *) &max_id, sizeof(unsigned long));
    if(fout.fail())
    {
        interface::log_this("employee::generate_id() : Error while writing to file max_id.dat (fout.fail())\nFunction aborted");
        return 0;
    }
    else
        return 1;
}

int employee::generate_id_status = 1;

employee::employee(str inp1, int inp2, Date inp3, address inp4, phone inp5, unsigned long inp6, Time inp7, Time inp8, str inp9, str inp10) : person(inp1, inp2, inp3, inp4, inp5), account(inp9, inp10)
{
    if(!generate_id_status)
    {
        interface::error("ID cannot be generated for this employee. Check log");
        interface::log_this("employee::employee() : ID generation using generate_id() unsuccessful as generate_id_status is set to zero.\nThis is because some error was encountered during the last ID generation");
    }
    else
    {
        employee::generate_id_status = generate_id();
        id_to_emp i1(id, OTHERS);
        if(!i1.status)
        {
            interface::error("ID not generated properly for this employee. Check log");
            interface::log_this("employee::employee() : i1.status was set to zero, i.e id_list.dat doesn't have a record of the employee's id");
        }
        salary = inp6;
        shift_start = inp7;
        shift_end = inp8;
    }
}

employee::employee() : person()
{
    id = 0;
}

int employee::get_age()
{
    ////////////////Updating age to present age///////////
    set_dob(dob);       //This function is used here to invoke calc_age() in it only(because calc_age is directly not accessible)
    void * temp = malloc( sizeof(doctor) );
    if(temp != NULL && hospital::get_employee_by_id(id, temp))          //if employee's file exists on disk
    {
        hospital::write_employee( this );               //overwrite that file
    }
    free(temp);
    return age;
}

unsigned long employee::get_salary(){
    return salary;
}

void employee::set_salary(unsigned long inp)
{
    salary = inp;
}

Time employee::get_shift(int inp){
    switch(inp){
        case START:
            return shift_start;
        case END:
            return shift_end;
        default:
            return Time(0,0,0);
    }
}

void employee::set_shift(int inp1, Time inp2)
{
    switch (inp1)
    {
        case START:
            shift_start = inp2;
            return;
        case END:
            shift_end = inp2;
            return;
        default:
            return;
    }
}

unsigned long employee::get_id()
{
    return id;
}

transaction * employee::get_last_5_transactions()
{
    transaction * t = (transaction *)malloc(5 * sizeof(transaction));
    if(t == NULL)
    {
        interface::log_this("employee::get_last_5_transactions() :Not enough
            memory to allocate buffer void * temp = malloc( sizeof(doctor) )\
            nFunction aborted");
        return NULL;
    }
    for(int i = 0; i < 5; ++i)
    {
        t[i] = transaction();
    }
    str temp;
    switch( id_to_emp::convert(id) )
    {
        case INVALID:
        {
            char log_msg[300];
            sprintf(log_msg, "employee::get_last_5_transactions() : The object
                has invalid id (%lu)\nFunction aborted", id);
            interface::log_this(log_msg);
            free(t);
            return NULL;
        }
        case DOCTOR:
        {
            sprintf(temp, "employee/doctor/%lu/trans.dat", id);
            break;
        }
        case NURSE:
        {
            sprintf(temp, "employee/nurse/%lu/trans.dat", id);
            break;
        }
        case RECEPTIONIST:
        {
            sprintf(temp, "employee/receptionist/%lu/trans.dat", id);
            break;
        }
        case OTHERS:
        {
            sprintf(temp, "employee/%lu/trans.dat", id);
            break;
        }
    }
    ifstream fin ( temp ,ios::binary | ios::in | ios::nocreate | ios::ate);
    if(!fin)
    {
        char log_msg[300];
        sprintf(log_msg, "employee::get_last_5_transactions() : Failed to open
            file trans.dat for id %lu\nFunction aborted", id);
        interface::log_this(log_msg);
        free(t);
        return NULL;
    }
    int max_i, size_of_file = fin.tellg();
    if( size_of_file >= ( 5 * sizeof(transaction) ) )
    {
        const int a = (-5) * sizeof(transaction);
        fin.seekg(a, ios::end);
        max_i = 5;
    }
    else
    {
        fin.seekg(0, ios::beg);
        max_i = (int)( size_of_file / sizeof(transaction) );
    }
    for(i = 0; i < max_i && !fin.eof(); ++i)
    {
```

```cpp
        fin.read((char *) (t+i), sizeof(transaction));
        if(fin.fail())
        {
            char log_msg[300];
            sprintf(log_msg, "employee::get_last_5_transactions() : Failed to
                read file trans.dat for id %lu(loop failed at i = %i)\nFunction
                aborted", id, i);
            interface::log_this(log_msg);
            free(t);
            return NULL;
        }
    }
    fin.close();
    return t;
}

int employee::get_generate_id_status()
{
    return generate_id_status;
}

//////////////////////////////////////////
//// Doctor, Nurse and Receptionist class member defs

doctor::doctor(str inp1, int inp2, Date inp3, address inp4, phone inp5, unsigned
    long inp6, Time inp7, Time inp8, int inp10, int inp11, str inp12, str inp13) :
     employee(inp1, inp2, inp3, inp4, inp5, inp6, inp7, inp8, inp12, inp13)
{
    id_to_emp i1(get_id(), DOCTOR);
    if(!i1.status)
    {
        interface::error("ID not generated properly for this employee. Check log"
            );
        interface::log_this("doctor::doctor() : i1.status was set to zero, i.e
            id_list.dat doesn't have a record of the employee's id");
    }
    speciality[0] = inp10;
    speciality[1] = inp11;

    for(int i = 0; i < 10; i++){
        patients[i] = 0;
    }
}

doctor::doctor() : employee()
{
    speciality[0] = speciality[1] = GEN + 1;    //storing an invalid value in
        speciality
    for(int i = 0; i < 10; ++i)
    {
        patients[i] = 0;
    }
}

int * doctor::get_speciality()
{
    return speciality;
}

```

```
long * doctor::get_patients()
{
    return patients;
}

void doctor::set_speciality(int inp[2])
{
    speciality[0] = inp[0];
    speciality[1] = inp[1];
}

void doctor::set_patients(long inp[10])
{
    for(int i = 0; i < 10; ++i)
    {
        patients[i] = inp[i];
    }
}

nurse::nurse(str inp1, int inp2, Date inp3, address inp4, phone inp5, unsigned
    long inp6, Time inp7, Time inp8, str inp10, str inp11) : employee(inp1, inp2,
    inp3, inp4, inp5, inp6, inp7, inp8, inp10, inp11)
{
    id_to_emp i1(get_id(), NURSE);
    if(!i1.status)
    {
        interface::error("ID not generated properly for this employee. Check log"
            );
        interface::log_this("nurse::nurse() : i1.status was set to zero, i.e
            id_list.dat doesn't have a record of the employee's id");
    }
    for(int i = 0; i < 5; i++){
        patients[i] = 0;
    }
}

nurse::nurse() : employee()
{
    for(int i = 0; i < 5; ++i)
    {
        patients[i] = 0;
    }
}

long * nurse::get_patients()
{
    return patients;
}

void nurse::set_patients(long inp[5])
{
    for(int i = 0; i < 5; ++i)
    {
        patients[i] = inp[i];
    }
}

receptionist::receptionist(str inp1, int inp2, Date inp3, address inp4, phone
    inp5, unsigned long inp6, Time inp7, Time inp8, str inp10, str inp11) :
```

```
    employee(inp1, inp2, inp3, inp4, inp5, inp6, inp7, inp8, inp10, inp11)
{
    id_to_emp i1(get_id(), RECEPTIONIST);
    if(!i1.status)
    {
        interface::error("ID not generated properly for this employee. Check log"
            );
        interface::log_this("receptionist::receptionist() : i1.status was set to
            zero, i.e id_list.dat doesn't have a record of the employee's id");
    }
}

receptionist::receptionist() : employee()
{}


///////////////////////////////////
/// Function definitions for class id_to_emp

id_to_emp::id_to_emp(unsigned long inp1, int inp2)
{
    status = 0;
    id = inp1;
    if(!id)
    {
        employee_type = INVALID;
    }
    else
    {
        employee_type = inp2;
    }
    mkdir("employee");
    ofstream fout;
    fout.open("employee/id_list.dat", ios::binary | ios::ate);
    if(!fout)
    {
        interface::log_this("id_to_emp::id_to_emp() : File id_list.dat couldn't
            be opened...\nFunction aborted");
    }
    else
    {
        fout.seekp(id * sizeof(id_to_emp), ios::beg);
        fout.write((char *) this, sizeof(id_to_emp));
        if(fout.fail())
        {
            interface::log_this("id_to_emp::id_to_emp() : Error while writing to
                file id_list.dat (fout.fail())\nFunction aborted");
        }
        else
        {
            status = 1;
        }
    }
}

id_to_emp::id_to_emp()
{
    id = employee_type = status = 0;
}
```

```cpp

int id_to_emp::convert(unsigned long ID)
{
    id_to_emp a;
    ifstream fin;
    fin.open("employee/id_list.dat", ios::binary);
    if(!fin)
    {
        interface::log_this("id_to_emp::convert() : File id_list.dat not found!!"
            );
        return INVALID;
    }
    fin.seekg( (ID * sizeof(id_to_emp)) );
    fin.read((char *) &a, sizeof(id_to_emp));
    if(fin.fail())
    {
        interface::log_this("id_to_emp::convert() : Error while reading from file
             id_list.dat (fin.fail())");
        return INVALID;
    }
    fin.close();
    if(a.id != ID)
    {
        interface::log_this("id_to_emp::convert() : (For dev only)Error in the
            code... Recheck it!!");
        return INVALID;
    }
    return a.employee_type;
}
```

8. code/PATIENT.CPP

```cpp
#include "patient.hpp"
#include <fstream.h>

///////FUNCTION DEFINITIONS FOR CLASS PATIENT/////////

patient::patient(str inp1, int inp2 , Date inp3, address inp4, phone inp5,
    disease inp6, str inp7, str inp8, phone inp9, insurance inp10, Date inp11) :
    person(inp1, inp2, inp3, inp4, inp5)    //if date_of_admission is the current
    system date, last argument is not needed
{
    fstream pat ("patient/max_id.dat", ios::in | ios::binary | ios::out);
    long max_id;
    pat.read( (char*) &max_id, sizeof(long) );
    max_id++;

    id = max_id;

    pat.seekp(0);
    pat.write( (char*) &max_id, sizeof(long) );
    pat.close();

    dis = inp6;
    strcpy(guardian_name, inp7);
    strcpy(emergency_contact, inp8);
    strcpy(emer_contact_no, inp9);
    insur_info = inp10;

    admission_date = inp11;
    Date dnow = system::get_date();

    if( admission_date.day != dnow.day ||
        admission_date.month != dnow.month ||
        admission_date.year != dnow.year        )
    {
        set_dob(inp3, inp11);
    }
    for(int i = 0; i < 50; i++){
        med[i][0] = med[i][1] = 0;
    }

    bill_amt = 0;   //bill_amt will be set by doctor after treatment
    discharged = 0;
}

patient::patient()
{
    id = 0;
}

long patient::get_id()
{
    return id;
}

disease patient::get_dis()
{
    return dis;
}

char* patient::get_guardian_name()
{
    return guardian_name;
}

char* patient::get_emergency_contact()
{
    return emergency_contact;
}

char* patient::get_emer_contact_no()
{
    return emer_contact_no;
}

insurance patient::get_insur_info()
{
    return insur_info;
}

int patient::get_admission_date(int inp)
{
    switch(inp)
    {
        case DAY:
            return admission_date.day;
        case MONTH:
            return admission_date.month;
        case YEAR:
            return admission_date.year;
        default:
            return 0;
    }
}

int patient::get_discharge_date(int inp)
{
    switch(inp)
    {
        case DAY:
            return discharge_date.day;
        case MONTH:
            return discharge_date.month;
        case YEAR:
            return discharge_date.year;
        default:
            return 0;
    }
}

unsigned long patient::get_bill_amt()
{
    return bill_amt;
}

int patient::get_med(int a, int b){
    return med[a][b];
}

transaction patient::get_transaction(int trans_num){
    str temp;
    transaction trans;
    sprintf(temp, "patient/%d/trans.dat", this->id);
    ifstream patient_file ( temp , ios::out | ios::binary | ios::app );

    int i = 0;
    while ( i<=trans_num && patient_file ){
        patient_file.read( (char*) &trans , sizeof(transaction) );
        i++;
    }
    if( i!= trans_num ){
        trans = transaction(0);
    }
    patient_file.close();
    return trans;
}

void patient::set_dis(disease a)
{
    dis = a;
}

void patient::set_guardian_name(char *a)
{
    strcpy(guardian_name, a);
```

```
}

void patient::set_emergency_contact(char *a)
{
    strcpy(emergency_contact, a);
}

void patient::set_emer_contact_no(char *a)
{
    strcpy(emer_contact_no, a);
}

void patient::set_insur_info(insurance a)
{
    insur_info = a;
}

void patient::set_admission_date(Date a)
{
    admission_date = a;
    set_dob(dob, admission_date);
}

void patient::set_bill_amt(unsigned long a)
{
    bill_amt = a;
}

void patient::set_med(int a, int b, int c){
    med[a][0] = b;
    med[a][1] = c;
}

void patient::set_discharge_date(Date inp){
    discharge_date = inp;
}

void patient::discharge(){
    discharged = 1;
}
```

9. code/PROC.CPP

```
#include <iostream.h>
#include <fstream.h>

typedef char str[100];

struct procedure{
    str name;
    float cost;
};

void main(){
    ofstream proc ("proc.dat" , ios::out | ios::binary | ios::app);
    procedure a;
    cin.ignore(1000, '\n');
    cout << "\nName:";
    cin.getline(a.name, 100, '\n');
    cout << "\nCost:";
    cin >> a.cost;
    cout << endl << "Procedure : " << a.name << "  $" << a.cost << ".\nEnter next
         procedure:";
    proc.write( (char*) &a , sizeof(a) );
}
```

10. code/UI/test.cpp

```
//No need to use ui::init() explicitly

#include "ui/ui.hpp"
#include "ui/test.hpp"

void test_weird_error()
{
    int shit = 14;
    box menu2(coord(2, 4), 40, 10 );
    menu2 << "Enter your shit: ";
    menu2 >> shit;
    menu2.setexit_button("Submit my shit");
    menu2.loop();

    menu2.clear();
    menu2 << "Your shit's coming up!" << ui::endl; getch();
    menu2 << "Here's your shit: ";
    menu2 << shit;
    menu2 << ". Deal with it!" << ui::endl;

    getch();
}

int exit_func()
{
    char c = getch();
    int x = wherex(), y = wherey();

    gotoxy(1, ui::scr_height - 1);
    if(c != '1')
    {
        cprintf("Returning 0"); getch();
        gotoxy(x, y);
        return 0;
    }
    else
    {
        cprintf("Returning 1"); getch();
        gotoxy(x, y);
        return 1;
    }
}

void test_back()
{
    box window;

    int a, b;
    window << "Here's some sample text" << ui::endl;
    window << "Enter some fake data I don't care about" << ui::endl;

    window << "Fake #1: "; window >> a;
    window << "Fake #2: "; window >> b;
    window.setexit_button("A fake button");

    window.setback_func(exit_func);

    window.loop();
}

void test_all()
{
    ui::clrscr();
    box menu2(coord(2, 4), 40, 10 );

    menu2.settcolor(GREEN);
    menu2 << ui::centeralign << "Employee Management" << ui::endl << ui::endl;
    menu2.settcolor(WHITE);
    int menu2_height;
    menu2_height = 10;
//  menu2.setheight(menu2_height);
    menu2 << "View employee data" << ui::endl;
    menu2.settcolor(ui::tcolor);
//  menu2 << "Enter employee's id: ";
    unsigned long id;
    menu2 >> id;
    menu2 << ui::endl;
    menu2.setexit_button("Submit");
    menu2.loop();

    menu2.clear();
    menu2.setheight(15);
    menu2.settcolor(GREEN);
    menu2 << ui::centeralign << "Employee Management" << ui::endl << ui::endl;
    menu2.settcolor(WHITE);
    menu2 << "Employee Details: " << ui::endl;
    menu2.settcolor(ui::tcolor);
        getch();
        menu2.hide();
        getch();
        menu2.display();
        getch();
    menu2 << "ID: " << 1 << ui::endl;
        getch();
        menu2.hide();
        getch();
        menu2.display();
        getch();

    char name[40], pwd[40];
    int age;
    long phn;
    float amt;
    char date[30];

    box window;
    window.settcolor(CYAN);
    window << ui::centeralign << "LHOSPITAL";
    window << ui::endl << ui::endl;
    window.settcolor(ui::tcolor);
    window.setfooter_tcolor(GREEN);

    window << box::setheader << "28/10/2017"
           << box::setheader << ui::rightalign << "11:45 PM"
           << box::setfooter << ui::centeralign
           << "Everything looks OK";

    window << "Fill the following form: " << ui::endl;

    coord c(ui::scr_width/4, ui::scr_height/3);
    box b(c, ui::scr_width / 3, 10);

    b.settcolor_input(YELLOW);
    b << "Enter details: " << ui::endl
      << "Name: "; b >> name;
    b << "Age: "; b >> age;
    b << "Phone num: "; b >> phn;
    b << "Date: ";
    b.setdefault("27/10/2017");
    b >> date;
    b << "Amount: "; b >> amt;
    b << "Password: "; b >> box::setpassword >> pwd;

    b.f.setvisibility_mode(frame::nosides);

    b.f.display();
    b.setexit_button("Submit");
    b.loop();

    b.hide();

    window << "You entered the following data: " << ui::endl
      << "Name: " << name << ui::endl
      << "Age: " << age << ui::endl
      << "Phone num: " << phn << ui::endl
      << "Date: " << date << ui::endl
      << "Amount: " << amt << ui::endl
      << "Password: " << pwd << ui::endl;
}


void test_listlayout()
{
    list_layout l;
    l.setpos(coord(2,1));
    l.setheight(6);

    interactive *list[10];

    //Setting the text boxes
    for(int i = 0; i < 9; i++)
    {
        char s[] = {'A'+i, ':', ' ', '\0'};
        l.settcolor(LIGHTGRAY);
        l << coord(2, i + 1) << s;
        l.settcolor(RED);
        list[i] = l.settext_box(coord(5, i + 1));
    }

    l.settcolor(LIGHTGRAY);
    list[9] = l.setbutton(coord(3, i + 1), "Submit");

    //Rudimentary scrolling
    i = 100;
    int j = 0;

    int lines_scrolled = l.getlines_scrolled(),
        height = l.getheight();

    coord pos_topleft(2,1);
    int y = pos_topleft.y;
    while(i--)
    {
        coord c = list[j]->getpos();
        if(c.y - lines_scrolled > height)
        {
            lines_scrolled = c.y - height;
        }
        else if(c.y - lines_scrolled < y)
        {
            lines_scrolled = c.y - y;
        }

        l.setlines_scrolled(lines_scrolled);
        int response = list[j]->input(-lines_scrolled);

        if(response == interactive::GOTONEXT)
        {
            if(j < 9) j++; else j = 0;
        }
        else if(response == interactive::GOTOPREV)
        {
            if(j > 0) j--; else j = 9;
        }
        else if(response == interactive::CLICKED)
        {
            coord init_pos(wherex(), wherey());
            gotoxy(1, ui::scr_height-1);
            cprintf("%s%d", "Clicked ", i);
            gotoxy(init_pos.x, init_pos.y);
        }
    }
}

void test_textbox()
{
    text_box t;
    t.setpos(coord(1,1));
    for(int i = 0; i < 5; i++)
    {
        int a = t.input();

        int x = wherex(), y = wherey();
        gotoxy(1, ui::scr_height-1);
        if(a == interactive::GOTONEXT)
        {
            cout << "GOTONEXT";
        }
        else if(a == interactive::GOTOPREV)
        {
            cout << "GOTOPREV";
        }
        else
        {
            cout << "UNDEFINED";
        }

        gotoxy(x, y);
    }
}

void test_frame()
{
    frame f;
    f.display();

    getch();

    f << ui::top << 't'
      << ui::left << 'l'
      << ui::bottom << 'b'
      << ui::right << 'r';

    f.settcolor(LIGHTBLUE);

    f.display();

    getch();

    f << (ui::top | ui::left) << (char) 201
      << (ui::bottom | ui::left) << (char) 200
      << (ui::top | ui::right) << (char) 187
      << (ui::bottom | ui::right) << (char) 188
      << ui::top << (char) 205
      << ui::bottom << (char) 205
      << ui::left << (char) 186
      << ui::right << (char) 186;

    f.settcolor(ui::tcolor);

    f.display();

    getch();

    f.setheight(ui::scr_height/2);
    getch();

    f.setwidth(ui::scr_width/3);
    getch();

    f.setcorner_top_left(coord( (ui::scr_width-f.getwidth()) / 2, (ui::scr_height
        -f.getheight()) / 2));
    getch();

    f.setvisibility_mode(frame::nosides);
}
```

11. code/UI/interact.cpp

```
#include "ui/ui.hpp"

string_node::string_node()
{
    next = NULL;
    prev = NULL;
    data = '\0';
}

interactive::interactive()
{
    prev = NULL;
    next = NULL;
}

interactive::~interactive()
{
    delete next;
    next = NULL;
    prev = NULL;
}

int interactive::input(int)
{
    return -1;
}

void interactive::setoffset(int o)
{
    offset = o;
}

int interactive::getoffset()
{
    return offset;
}

int interactive::getkey()
{
    char ch = getch();
    switch(ch)
    {
        case 9:     return TAB;
        case 13:    return ENTER;
        case 8:
        {
            unsigned char far *key_state_byte
                = (unsigned char far*) 0x00400017;
            int key_state = (int) *key_state_byte;

            if(key_state & 2) return SHIFT_BACKSPACE;
            else              return BACKSPACE;
        }
```

```cpp
        case 0:     break;
        default:    return ch;
    }

    ch = getch();

    unsigned char far *key_state_byte
    = (unsigned char far*) 0x00400017;
    int key_state = (int) *key_state_byte;

    switch(ch)
    {
        case 72:    return UP;
        case 80:    return DOWN;
        case 75:    return LEFT;
        case 77:    return RIGHT;
        case 15:    if (key_state & 2) return SHIFT_TAB;
        //           ^^ Checks if shift was pressed
        case 83:    return DELETE;
        case 71:    return HOME;
        case 79:    return END;
    }

    return -1;
}
```

12. code/UI/uibase.cpp

```cpp
#include "ui/ui.hpp"
#include "iface.hpp"

int init_lib_ui::counter = 0;

init_lib_ui::init_lib_ui()
{
    if(counter++ == 0)
    {
        ui::init();
    }
}

int manipulator::index = 0;

manipulator::manipulator()
{
    own_index = index;
    index++;
}

int manipulator::operator==(manipulator m)
{
    return own_index == m.own_index;
}

int ui::scr_height = 0,
    ui::scr_width = 0,
    ui::tcolor = LIGHTGRAY,
    ui::bcolor = BLACK;
```

```
manipulator ui::endl,
            ui::centeralign,
            ui::rightalign;

void ui::init()
{
    set_new_handler(ui::my_new_handler);

    ui::clrscr();

    textcolor(ui::tcolor);
    textbackground(ui::bcolor);

    struct text_info info;
    gettextinfo(&info);

    //height and width of screen
    scr_width = (int) info.screenwidth;
    scr_height = (int) info.screenheight;
}

void ui::clrscr()
{
    ::clrscr();
}

void ui::my_new_handler()
{
    interface::log_this("Error in allocating memory. Exiting...");
    exit(1);
}

coord::coord(int X, int Y)
{
    x = X;
    y = Y;
}

coord & coord::operator+=(coord b)
{
    x += b.x;
    y += b.y;

    return *this;
}

coord & coord::operator-=(coord b)
{
    x -= b.x;
    y -= b.y;

    return *this;
}

coord coord::operator+(coord b)
{
    coord temp = *this;
    return temp += b ;
}
```

```cpp

coord coord::operator-(coord b)
{
    coord temp = *this;
    return temp -= b;
}
```

13. code/UI/frame.cpp

```cpp
#include "ui/ui.hpp"

int frame::convert(int param)
{
    if(param & ui::top)
    {
        if(param & ui::left)
        {
            return 0;
        }
        else if(param & ui::right)
        {
            return 1;
        }
        else
        {
            return 2;
        }
    }
    else if(param & ui::bottom)
    {
        if(param & ui::left)
        {
            return 3;
        }
        else if(param & ui::right)
        {
            return 4;
        }
        else
        {
            return 5;
        }
    }
    else if(param & ui::left)
    {
        return 6;
    }
    else if(param & ui::right)
    {
        return 7;
    }

    return -1;
}

void frame::setside_visibility(int side, int visib)
{
    if( visib != 0 && visib != 1)
```

```
        return;     //No effect for invalid visibility

    if(side & ui::all)
    {
        for(int i = 0; i < 8; i++)
            sides_visibility[i] = visib;
        return;
    }

    int a = frame::convert(side);
    if(a == -1) return; //-1 indicates invalid side

    sides_visibility[a] = visib;
}

int frame::getside_visibility(int side)
{
    int a = convert(side);

    if(a == -1) return -1; //Wrong side selected

    return sides_visibility[a];
}


frame::frame(coord topleft, int w, int h)
{
    for(int i = 0; i < 8; i++)
    {
        border_chars[i] = '*';
        sides_visibility[i] = 1;
    }
    tcolor = ui::tcolor;
    bcolor = ui::bcolor;
    frame_visibility = 0;
    height = h;
    width = w;
    state = 0;
    corner_top_left = topleft;
}

void frame::display()
{
    print(1);
}

void frame::hide()
{
    print(0);
}

void frame::print(int param)
{
    textcolor(frame::tcolor);
    textbackground(frame::bcolor);

    char visible_chars[8];
    frame_visibility = param;

    int x = corner_top_left.x,
        y = corner_top_left.y;

    int arr[] = {
                ui::top,
                ui::bottom,
                ui::left,
                ui::right,
                ui::top | ui::left,
                ui::top | ui::right,
                ui::bottom | ui::left,
                ui::bottom | ui::right
            };

    char &top = visible_chars[0],
         &bottom = visible_chars[1],
         &left = visible_chars[2],
         &right = visible_chars[3],
         &top_left = visible_chars[4],
         &top_right = visible_chars[5],
         &bottom_left = visible_chars[6],
         &bottom_right = visible_chars[7];

    for(int i = 0; i < 8; i++)
    {
        if(param == 1 && getside_visibility(arr[i]))
        {
            visible_chars[i] = getborder_char(arr[i]);
        }
        else
        {
            visible_chars[i] = ' ';
        }
    }

    gotoxy(x, y);

    cprintf("%c", top_left);

    for(i = 1; i < width - 1; i++)
    {
        cprintf("%c", top);
    }
    cprintf("%c", top_right);

    for(i = 1; i < height - 1; i++)
    {
        gotoxy(x, y + i); cprintf("%c", left);
        gotoxy(x + width - 1, y + i); cprintf("%c", right);
    }

    gotoxy(x, y + height - 1);
    cprintf("%c", bottom_left);
    for(i = 1; i < width - 1; i++)
    {
        cprintf("%c", bottom);
    }
    cprintf("%c", bottom_right);

    gotoxy(corner_top_left.x, corner_top_left.y);

    textcolor(ui::tcolor);
}

void frame::setvisibility_mode(int param)
{
    frame::setside_visibility(frame::all, 1);
    if(param & nosides)
    {
        frame::setside_visibility(ui::left, 0);
        frame::setside_visibility(ui::right, 0);
    }
    frame::display();
}

//Operator << is used to set border char
frame & frame::operator<<(int side)
{
    int a = frame::convert(side);

    if(a == -1) return *this; //-1 indicates error

    state = a;

    return *this;
}

frame & frame::operator<<(char border_char)
{
    border_chars[frame::state] = border_char;
    return *this;
}

int frame::getheight()
{
    return height;
}

int frame::getwidth()
{
    return width;
}

coord frame::getcorner_top_left()
{
    return frame::corner_top_left;
}

int frame::getframe_visibility()
{
    return frame_visibility;
}

int frame::gettcolor()
{
    return tcolor;
}

```

```
int frame::getbcolor()
{
    return bcolor;
}

char frame::getborder_char(int side)
{
    int a = convert(side);

    if(a == -1) return '\0'; //Error

    return frame::border_chars[a];
}

void frame::setheight(int h)
{
    if(h > ui::scr_height) return;

    hide();
    frame::height = h;
    display();
}

void frame::setwidth(int w)
{
    if(w > ui::scr_width) return;

    hide();
    frame::width = w;
    display();
}

void frame::settcolor(int c)
{
    tcolor = c;
    display();
}

void frame::setbcolor(int b)
{
    bcolor = b;
    display();
}

void frame::setcorner_top_left(coord c)
{
    hide();
    frame::corner_top_left = c;
    display();
}
```

14. code/UI/box.cpp

```
#include "ui/ui.hpp"
#include "iface.hpp"

line::line()
{
    strcpy(left, "");
    strcpy(middle, "");
    strcpy(right, "");
    width = ui::scr_width - 2;
    tcolor = ui::tcolor;
    bcolor = ui::bcolor;
    corner_top_left = coord(0,0);
}

void line::display()
{
    print(1);
}

void line::hide()
{
    print(0);
}

void line::clear()
{
    hide();
    strcpy(left, "");
    strcpy(middle, "");
    strcpy(right, "");
    display();
}

void line::print(int mode)
{
    coord curr_pos = coord(wherex(), wherey()),
    &ctl = corner_top_left;
    gotoxy(ctl.x, ctl.y);
    textcolor(tcolor);
    textbackground(bcolor);

    if(mode == 1)
    {
        cprintf("%s", left);
    }
    else
    {
        for(int i = 0; i < strlen(left); i++)
        {
            cprintf(" ");
        }
    }

    gotoxy(ctl.x + (width - strlen(middle)) / 2,
                wherey());
    if(mode == 1)
    {
        cprintf("%s", middle);
    }
    else
    {
        for(int i = 0; i < strlen(middle); i++)
        {
            cprintf(" ");
        }
    }

    gotoxy(ctl.x + width - strlen(right), wherey());
    if(mode == 1)
    {
        cprintf("%s", right);
    }
    else
    {
        for(int i = 0; i < strlen(right); i++)
        {
            cprintf(" ");
        }
    }

    gotoxy(curr_pos.x, curr_pos.y);
}

int default_back_func()
{
    return 0;
}

int box::wrap(char str[], int length, int return_one_line)
{
    int num_lines = 1;
    char out_str[300] = "";

    int pos_old_newline = -1,
        pos_curr_newline = -1;

    int len_str = strlen(str);

    //Iterating upto len_str because the '\0' at the end of the string
    //would be interpreted as a newline
    for(int i = 0; i <= len_str; i++)
    {
        if(str[i] == '\n' || i == len_str)
        {
            pos_old_newline = pos_curr_newline;
            pos_curr_newline = i;

            if(pos_curr_newline != len_str) num_lines++;

            int chars_read = 0,
                read,
                written = 0;

            char word[30];

            str[pos_curr_newline] = '\0';

            char *line = str + pos_old_newline + 1;
            while(sscanf(line + chars_read, "%s%n", word, &read) > 0)
            {
                int word_len = strlen(word);
                if(written + word_len  > length)
                {
                    num_lines++;
                    sprintf(out_str + strlen(out_str), "\n%s ", word);
                    written = word_len + 1;
                }
                else if(written + word_len < length)
                {
                    sprintf(out_str + strlen(out_str), "%s ", word);
                    written += word_len + 1;
                }
                else //Not to add the space at the end if the line just completes
                {
                    sprintf(out_str + strlen(out_str), "%s", word);
                    written += word_len;
                }

                chars_read += read;
            }

            if(pos_curr_newline != len_str)
                sprintf(out_str + strlen(out_str), "\n");
                str[pos_curr_newline] = '\n';
        }
    }

    //An extra space is at the end of the string which has to be removed
    //out_str[strlen(out_str) - 1] = '\0';
    sprintf(str, "%s", out_str);

    if(!return_one_line)   return num_lines;

    len_str = strlen(str);

    for(i = 0; i <= len_str; i++)
    {
        if(i == len_str)
        {
            break;
        }
        else if(str[i] == '\n')
        {
            str[i] = '\0';
            break;
        }
    }

    return num_lines;
}

void box::set_tbox(int data_type, void *ptr)
{
    text_box *new_tbox;

    if(data_type == info_tbox::PASSWORD)
    {
        new_tbox =
            (text_box *) layout.settext_box(pos_pointer, 1);
    }
    else
    {
        new_tbox =
            (text_box *) layout.settext_box(pos_pointer);
    }

    if(default_toggle)
    {
        default_toggle = 0;
        new_tbox -> setstr(default_text);
    }

    pos_pointer.y++;
    pos_pointer.x = layout.getcorner_top_left().x;

    list_interactive[index_interactive]
        = (interactive *) new_tbox;
    info_tbox &t = list_tbox[index_tbox];
    index_interactive++;
    index_tbox++;

    t.tbox = new_tbox;
    t.type = data_type;
    t.data_store = ptr;
    t.validator = validation::getvalidator(data_type, temp_validator);

    temp_validator = NULL;
}

manipulator box::setheader,
            box::setfooter,
            box::setpassword;

box::box(coord c, int w, int h) : f(c, w, h)
{
    width = w;
    height = h;
    padding = 1;

    corner_top_left = c;

    f << (ui::top | ui::left) << (char) 201
      << (ui::bottom | ui::left) << (char) 200
      << (ui::top | ui::right) << (char) 187
      << (ui::bottom | ui::right) << (char) 188
      << ui::top << (char) 205
      << ui::bottom << (char) 205
      << ui::left << (char) 186
      << ui::right << (char) 186;

    layout.setwidth(w - 2 - 2 * padding);
    layout.setheight(h - 2 - 2 * padding);
    //                  ^bcoz of frame
    layout.setcorner_top_left(c +
        coord(1 + padding, 1 + padding));

    pos_pointer = layout.getcorner_top_left();

    for(int i = 0; i < 30; i++)
    {
        list_interactive[i] = NULL;
    }
    exit_btn = NULL;
    index_interactive = index_tbox = 0;
    center_toggle = 0;
    default_toggle = 0;
    right_toggle = 0;
    header_toggle = 0;
    footer_toggle = 0;
    password_toggle = 0;
    strcpy(default_text, "");
    temp_validator = NULL;

    header.width = footer.width = w - 2;
    header.corner_top_left = c + coord(1,0);
    footer.corner_top_left = c + coord(0, h-1);

    back_func = default_back_func;

    f.display();
}

coord box::getcorner_top_left()
{
    return corner_top_left;
}

int box::getheight()
{
    return height;
}

int box::getwidth()
{
    return width;
}

int box::getpadding()
{
    return padding;
}

void box::setcorner_top_left(coord c)
{
    corner_top_left = c;
    f.setcorner_top_left(c);
    c += coord(1 + padding, 1 + padding);
    layout.setcorner_top_left(c);

    pos_pointer = c;
}

void box::setheight(int h)
{
    height = h;
    f.setheight(h);
    layout.setheight(h - 2 - 2 * padding);
}

void box::setpadding(int p)
```

```
{
    hide();
    padding = p;
    setheight(height);
    display();
}

void box::settcolor(int c)
{
    layout.settcolor(c);
}

void box::setbcolor(int c)
{
    layout.setbcolor(c);
}

void box::settcolor_selected(int c)
{
    layout.settcolor_selected(c);
}

void box::setbcolor_selected(int c)
{
    layout.setbcolor_selected(c);
}

void box::settcolor_input(int c)
{
    layout.settcolor_input(c);
}

void box::setbcolor_input(int c)
{
    layout.setbcolor_input(c);
}

void box::setback_func( int(*f)(void) )
{
    back_func = f;
}

box & box::operator<< (char *inp_str)
{
    char string[100];
    char *str = string;
    strcpy(string, inp_str);

    coord c = layout.getcorner_top_left();

    if(header_toggle || footer_toggle)
    {
        line *lp;
        if(header_toggle)
        {
            header_toggle = 0;
            lp = &header;
        }
        if(footer_toggle)
        {
            footer_toggle = 0;
            lp = &footer;
        }
        line &l = *lp;

        int len = strlen(string);
        if(center_toggle)
        {
            center_toggle = 0;
            if(len <= l.width)
            {
                if((l.width - len) / 2 > strlen(l.left))
                {
                    strcpy(l.middle, string);
                }
            }
        }
        else if(right_toggle)
        {
            right_toggle = 0;
            if(len <= l.width)
            {
                if(len < (l.width - strlen(l.middle)) / 2)
                {
                    strcpy(l.right, string);
                }
            }
        }
        else
        {
            if(len < (l.width - strlen(l.middle)) / 2)
            {
                strcpy(l.left, string);
            }
        }

        //Printing the newly set line
        l.hide();
        l.display();

        return *this;
    }

    if(center_toggle)
    {
        int len = strlen(string);
        center_toggle = 0;
        if(len <= layout.getwidth())
        {
            int x_center_pos =
                c.x + (layout.getwidth() - len) / 2;

            if(pos_pointer.x > x_center_pos)
            {
                pos_pointer.y++;
            }
            pos_pointer.x = x_center_pos;
            layout << pos_pointer << str;
            pos_pointer.x += len;
            return *this;
        }
    }
    else if(right_toggle)
    {
        int len = strlen(string);
        right_toggle = 0;
        if(len <= layout.getwidth())
        {
            int x_right_pos =
                c.x + (layout.getwidth() - len);

            if(pos_pointer.x > x_right_pos)
            {
                pos_pointer.y++;
            }
            pos_pointer.x = x_right_pos;
            layout << pos_pointer << str;
            pos_pointer.y++;
            pos_pointer.x = c.x;
            return *this;
        }
    }

    int num_lines;

    if(pos_pointer.x != c.x)
    {
        int remaining_space = layout.getwidth() -
        (pos_pointer.x - layout.getcorner_top_left().x);
        char s[100];
        strcpy(s, str);
        num_lines = wrap(s, remaining_space, 1);

        layout << pos_pointer << s;

        if(num_lines > 1)
        {
            pos_pointer.x = c.x;
            pos_pointer.y++;
        }
        else
        {
            pos_pointer.x += strlen(s);
        }

        if (num_lines == 1 ||
            str[strlen(str) - 1] == '\n')   return *this;

        str += strlen(s); //There's an extra space at the end of s
    }

    num_lines = wrap(str, layout.getwidth());

    int len_str = strlen(str),
        pos_curr_newline = -1,
        chars_to_forward = 0;

    for(int i = 0; i < len_str; i++)
    {
        if(str[i] == '\n')
        {
            pos_curr_newline = i;

            str[pos_curr_newline] = '\0';
            layout << pos_pointer << str + chars_to_forward;
            pos_pointer.y++;

            chars_to_forward +=
                strlen(str + chars_to_forward) + 1;
        }
    }

    if(i == len_str - 1)    return *this;

    layout << pos_pointer << str + chars_to_forward;
    pos_pointer.x += strlen(str + chars_to_forward);

    return *this;
}

box & box::operator<<(char ch)
{
    char str[] = {ch, '\0'};
    return (*this) << str;
}

box & box::operator<<(int i)
{
    return (*this) << (long) i;
}

box & box::operator<<(long l)
{
    char str[100];
    sprintf(str,"%ld", l);
    return (*this) << str;
}

box & box::operator<<(unsigned long ul)
{
    char str[100];
    sprintf(str, "%lu", ul);
    return (*this) << str;
}

box & box::operator<<(double d)
{
    char str[100];
    sprintf(str, "%g", d);
    return (*this) << str;
}

box & box::operator<<(float f)
{
    char str[100];
    sprintf(str, "%f", f);
    return (*this) << str;
}

box & box::operator<<(manipulator m)
{
    if(m == ui::endl)
    {
        pos_pointer.y++;
        pos_pointer.x = layout.getcorner_top_left().x;
    }
    else if(m == ui::centeralign)
    {
        center_toggle = 1;
    }
    else if(m == ui::rightalign)
    {
        right_toggle = 1;
    }
    else if(m == box::setheader)
    {
        header_toggle = 1;
    }
    else if(m == box::setfooter)
    {
        footer_toggle = 1;
    }
    return *this;
}

box & box::operator>>(char *&s)
{
    if(password_toggle)
    {
        password_toggle = 0;
        set_tbox(info_tbox::PASSWORD, (void *) s);
    }
    else
    {
        set_tbox(info_tbox::STRING, (void *) s);
    }
    return *this;
}

box & box::operator>>(char &ch)
{
    set_tbox(info_tbox::CHAR, (void *) &ch);
    return *this;
}

box & box::operator>>(int &i)
{
    set_tbox(info_tbox::INT, (void *) &i);
    return *this;
}

box & box::operator>>(long &l)
{
    set_tbox(info_tbox::LONG, (void *) &l);
    return *this;
```

```
}

box & box::operator>>(unsigned long &ul)
{
    set_tbox(info_tbox::UNSIGNED_LONG, (void *) &ul);
    return *this;
}

box & box::operator>>(double &d)
{
    set_tbox(info_tbox::DOUBLE, (void *) &d);
    return *this;
}

box & box::operator>>(float &f)
{
    set_tbox(info_tbox::FLOAT, (void *) &f);
    return *this;
}

box & box::operator>>(manipulator m)
{
    if(m == box::setpassword)
    {
        password_toggle = 1;
    }
    return *this;
}

box & box::operator>>(int (*f)(const char *))
{
    temp_validator = f;
    return *this;
}

void box::setexit_button(char *str)
{
    coord c = layout.getcorner_top_left();
    if(pos_pointer.x != c.x)
        pos_pointer.y++;

    pos_pointer.x = c.x + (layout.getwidth() - strlen(str)) / 2;

    button * new_btn =
        (button *) layout.setbutton(pos_pointer, str);

    pos_pointer.y++;
    pos_pointer.x = c.x;

    exit_btn = new_btn;
    list_interactive[index_interactive]
        = (interactive *) new_btn;
    index_interactive++;
}

void box::setdefault(char *s)
{
    default_toggle = 1;
    strcpy(default_text, s);
```

```
}

void box::setdefault(char c)
{
    char s[] = {c, '\0'};
    setdefault(s);
}

void box::setdefault(int i)
{
    setdefault( (long) i);
}

void box::setdefault(long l)
{
    char s[100];
    sprintf(s, "%ld", l);
    setdefault(s);
}

void box::setdefault(unsigned long ul)
{
    char s[100];
    sprintf(s, "%lu", ul);
    setdefault(s);
}

void box::setdefault(double d)
{
    char s[100];
    sprintf(s, "%g", d);
    setdefault(s);
}

void box::setdefault(float f)
{
    char s[100];
    sprintf(s, "%f", f);
    setdefault(s);
}

void box::loop()
{
    int j = 0,
    lines_scrolled = layout.getlines_scrolled(),
    height = layout.getheight(),
    index_last_interactive = index_interactive - 1,
    &ili = index_last_interactive;
    int temp_tbox_color, temp_index = -1;

    inf_loop:
    while(1)
    {
        coord c = list_interactive[j]->getpos(),
              ctl = layout.getcorner_top_left();
        if(c.y - ctl.y - lines_scrolled + 1 > height)
        {
            lines_scrolled = c.y - ctl.y - height + 1;
        }
        else if(c.y - lines_scrolled < ctl.y)
        {
            lines_scrolled =
                c.y - ctl.y;
        }

        layout.setlines_scrolled(lines_scrolled);
        int response =
            list_interactive[j]->input(-lines_scrolled);

        if(response == interactive::GOTONEXT)
        {
            if(j < ili) j++; else j = 0;
        }
        else if(response == interactive::GOTOPREV)
        {
            if(j > 0) j--; else j = ili;
        }
        else if(response == interactive::CLICKED)
        {
            break;
        }
        else if(response == interactive::BACK && back_func())
        {
            return;
        }
    }

    interface::clear_error();
    if(temp_index != -1)
    {
        list_tbox[temp_index].tbox->settcolor(temp_tbox_color);
    }
    for(int i = 0; i < index_tbox; i++)
    {
        if(list_tbox[i].setdata() == 0)
        {
            interface::error("INVALID INPUT!");
            temp_tbox_color = list_tbox[i].tbox->gettcolor();
            list_tbox[i].tbox->settcolor(RED);
            temp_index = i;
            goto inf_loop;
        }
    }
}

void box::display()
{
    layout.display();
    f.display();
    header.display();
    footer.display();
}

void box::hide()
{
    layout.hide();
    f.hide();
    header.hide();
    footer.hide();
}

void box::clear()
{
    layout.hide();
    layout.clear();
    pos_pointer = layout.getcorner_top_left();
    index_interactive = index_tbox = 0;
    exit_btn = NULL;
    f.display();
}

void box::setheader_tcolor(int c)
{
    header.tcolor = c;
}

void box::setfooter_tcolor(int c)
{
    footer.tcolor = c;
}

void box::clear_header()
{
    header.clear();
    f.display();
    footer.display();
}

void box::clear_footer()
{
    footer.clear();
    f.display();
    header.display();
}
```

15. code/UI/validation.cpp

```
#include "ui/ui.hpp"

int validation::vint(const char *str)
{
    if(!validation::vlong(str)) return 0;

    char *end;
    long l = strtol(str, &end, 10);
    if(l > INT_MAX || l < INT_MIN)
    {
        return 0;
    }

    return 1;
}

int validation::vlong(const char *str)
{
    char *end;
    long val = strtol(str, &end, 10);

    if (errno == ERANGE || (errno != 0 && val == 0))
    {
        //If the converted value would fall
        //out of the range of the result type.
        return 0;
    }
    if (end == str)
    {
       //No digits were found.
       return 0;
    }

    //Check if the string was fully processed.
    return *end == '\0';
}

int validation::vunsigned_long(const char *str)
{
    char *end;
    unsigned long val = strtoul(str, &end, 10);

    if (errno == ERANGE || (errno != 0 && val == 0))
    {
        return 0;
    }
    if (end == str || *end != '\0')
    {
        return 0;
    }

    int len = strlen(str);
    for(int i = 0; i < len && isspace(str[i]); i++);

    if(str[i] == '-')  return 0;

    return 1;
}

int validation::vstring(const char *str)
{
    return 1;
}

int validation::vchar(const char *str)
{
    if(strlen(str) == 1 && isalnum(str[0]))
    {
        return 1;
    }
    return 0;
}

int validation::vdouble(const char *str)
{
    char *end;
    double val = strtod(str, &end);

    if (errno == ERANGE)
    {
        //If the converted value would fall
        //out of the range of the result type.
        return 0;
    }
    if (end == str)
    {
       //No digits were found.
       return 0;
    }

    return *end == '\0';
}

int validation::vfloat(const char *str)
{
    return validation::vdouble(str);
}

validator_f validation::getvalidator
                (int type, validator_f v)
{
    if(v != NULL) return v;

    switch(type)
    {
        case info_tbox::INT:
            return validation::vint;
        case info_tbox::LONG:
            return validation::vlong;
        case info_tbox::UNSIGNED_LONG:
            return validation::vunsigned_long;
        case info_tbox::STRING:
        case info_tbox::PASSWORD:
            return validation::vstring;
        case info_tbox::CHAR:
            return validation::vchar;
        case info_tbox::DOUBLE:
            return validation::vdouble;
        case info_tbox::FLOAT:
            return validation::vfloat;
    }

    //TODO: log undefined behaviour
    return NULL;
}
```

16. code/UI/llayout.cpp

```
#include "ui/ui.hpp"

list_layout_node::list_layout_node()
{
    next = NULL;
    tcolor = ui::tcolor;
    bcolor = ui::bcolor;
    strcpy(str, "");
    print_type = DEFAULT;
}

list_layout_node::~list_layout_node()
{
    delete next;
    next = NULL;
}

//Setters
void list_layout_node::setnext(list_layout_node *n)
{
    next = n;
}

void list_layout_node::setpos(coord p)
{
    pos = p;
}

void list_layout_node::settcolor(int t)
{
    tcolor = t;
}

void list_layout_node::setbcolor(int b)
{
    bcolor = b;
}

void list_layout_node::setstr(const char * s)
{
    strcpy(str, s);
}

void list_layout_node::setprint_type(int p)
{
    print_type = p;
}

//Getters
list_layout_node * list_layout_node::getnext()
{
    return next;
}

coord list_layout_node::getpos()
{
    return pos;
}

int list_layout_node::gettcolor()
{
    return tcolor;
}

int list_layout_node::getbcolor()
{
    return bcolor;
```

```
}

const char * list_layout_node::getstr()
{
    return str;
}

int list_layout_node::getprint_type()
{
    return print_type;
}

void list_layout::print(int print_mode)
{
    coord init_pos(wherex(), wherey());
    for(list_layout_node *curr = head; curr; curr = curr->getnext())
    {
        coord c = curr->getpos();
        int new_y = c.y - lines_scrolled;

        coord ctl = getcorner_top_left();
        if(new_y < ctl.y || new_y > ctl.y + height - 1) continue;

        gotoxy(c.x, new_y);
        textcolor(curr->gettcolor());
        textbackground(curr->getbcolor());
        if(print_mode == DISPLAY)
        {
            if(curr->getprint_type() ==
                 list_layout_node::PASSWORD)
            {
                int len = strlen(curr->getstr());
                for(int i = 0; i < len; i++)
                {
                    cprintf("*");
                }
            }
            else if(current->getprint_type() ==
                        list_layout_node::DEFAULT)
            {
                cprintf("%s", curr->getstr());
            }
        }
        else if(print_mode == HIDE)
        {
            int len = strlen(curr->getstr());
            for(int i = 0; i < len; i++)
            {
                cprintf(" ");
            }
        }
    }
    gotoxy(init_pos.x, init_pos.y);
}

list_layout::list_layout()
{
    head = NULL,
    current = NULL;

    tcolor = ui::tcolor;
    bcolor = ui::bcolor;
    tcolor_selected = ui::bcolor;
    bcolor_selected = ui::tcolor;
    tcolor_input = tcolor;
    bcolor_input = bcolor;

    height = ui::scr_height - 1;
    width = ui::scr_width;
    lines_scrolled = 0;
}

list_layout& list_layout::operator<<(coord c)
{
    pos = c;
    return *this;
}

list_layout& list_layout::operator<<(const char *str)
{
    if(!head) //empty list
    {
        head = new list_layout_node;
        current = head;
    }
    else
    {
        list_layout_node *new_node = new list_layout_node;
        current->setnext(new_node);
        current = current->getnext();
    }

    current->setpos(pos);
    current->setstr(str);
    current->settcolor(tcolor);
    current->setbcolor(bcolor);

    print();

    return *this;
}

interactive * list_layout::settext_box(coord c, int is_pwd)
{
    interactive *new_node = new text_box;
    new_node->setpos(c);
    new_node->settcolor(tcolor_input);
    new_node->setbcolor(bcolor_input);

    if(is_pwd)
    {
        ((text_box *) new_node)->setis_password(1);
        new_node->setprint_type(list_layout_node::PASSWORD);
    }

    current->setnext(new_node);
    current = current->getnext();

    return new_node;
}

interactive * list_layout::setbutton(coord c, const char *s)
{
    button *new_node = new button;
    new_node->setpos(c);
    new_node->settcolor(tcolor);
    new_node->setbcolor(bcolor);
    new_node->settcolor_selected(tcolor_selected);
    new_node->setbcolor_selected(bcolor_selected);
    new_node->setstr(s);

    interactive *n = (interactive *) new_node;
    current->setnext(n);
    current = current->getnext();

    return n;
}

void list_layout::settcolor(int c)
{
    tcolor = c;
    tcolor_input = c;
}

void list_layout::setbcolor(int c)
{
    bcolor = c;
    bcolor_input = c;
}

void list_layout::settcolor_selected(int c)
{
    tcolor_selected = c;
}

void list_layout::setbcolor_selected(int c)
{
    bcolor_selected = c;
}

void list_layout::settcolor_input(int c)
{
    tcolor_input = c;
}

void list_layout::setbcolor_input(int c)
{
    bcolor_input = c;
}

void list_layout::setcorner_top_left(coord c)
{
    hide();

    coord offset = c - corner_top_left;
    //offset isn't a coordinate but it's just a pair of values

    for(list_layout_node *curr = head; curr; curr = curr->getnext())
    {
        coord a = curr->getpos();
        a += offset;
        curr->setpos(a);
    }

    corner_top_left += offset;
    pos += offset;

    display();
}

void list_layout::setheight(int h)
{
    hide();
    height = h;
    display();
}

void list_layout::setwidth(int w)
{
    width = w;
}

void list_layout::setlines_scrolled(int l)
{
    hide();
    lines_scrolled = l;
    display();
}

void list_layout::setpos(coord c)
{
    pos = c;
}

int list_layout::getheight()
{
    return height;
}

int list_layout::getwidth()
{
    return width;
}

int list_layout::getlines_scrolled()
{
    return lines_scrolled;
}

coord list_layout::getpos()
{
    return pos;
}

coord list_layout::getcorner_top_left()
{
    return corner_top_left;
}

void list_layout::display()
{
    print(DISPLAY);
}

void list_layout::hide()
{
    print(HIDE);
}

void list_layout::clear()
{
    list_layout_node *curr = head;
    head = current = NULL;

    while(curr)
    {
        list_layout_node *temp = curr->getnext();
        delete curr;
        curr = temp;
    }

    lines_scrolled = 0;
    pos = corner_top_left;
}
```

17. code/UI/button.cpp

```
#include "ui/ui.hpp"

button::button()
{
    tcolor_selected = BLACK;
    bcolor_selected = LIGHTGRAY;
}

void button::settcolor_selected(int c)
{
    tcolor_selected = c;
}

void button::setbcolor_selected(int c)
{
    bcolor_selected = c;
}

int button::gettcolor_selected()
{
    return tcolor_selected;
}

int button::getbcolor_selected()
{
    return bcolor_selected;
}
```

```

int button::input(int offset)
{
    coord c = getpos();
    setoffset(offset);
    c.y += offset;
    gotoxy(c.x, c.y);

    print(1);

    int state_to_return;
    while(1)
    {
        if(kbhit())
        {
            char ch = interactive::getkey();
            switch((int) ch)
            {
                case interactive::ENTER :
                    state_to_return = interactive::CLICKED;
                    goto next;
                case interactive::DOWN :
                case interactive::TAB  :
                    state_to_return = interactive::GOTONEXT;
                    goto next;
                case interactive::UP :
                case interactive::SHIFT_TAB :
                    state_to_return = interactive::GOTOPREV;
                    goto next;
                case interactive::SHIFT_BACKSPACE :
                    state_to_return = interactive::BACK;
                    goto next;
            }
        }
    }

    next:
    {
        if (
            state_to_return == interactive::GOTONEXT ||
            state_to_return == interactive::GOTOPREV
           )
        {
            print(0);
        }

        return state_to_return;
    }
}

void button::print(int isselected)
{
    if(isselected)
    {
        textcolor(tcolor_selected);
        textbackground(bcolor_selected);
    }
    else
    {
        textcolor(gettcolor());
        textbackground(getbcolor());
    }

    coord init_pos(wherex(), wherey());
    coord c = getpos();
    gotoxy(c.x, c.y + getoffset());
    cprintf(getstr());
    gotoxy(init_pos.x, init_pos.y);
}
```

18. code/UI/textbox.cpp

```
#include "ui/ui.hpp"

text_box::text_box()
{
    is_password = 0;
}

/*
* Despite trying, this function has grown quite large
* Basically, it allows the user to enter text in the box
* and stores it.
* Returns GOTONEXT or GOTOPREV as per user's request to
* go to the next or the previous text box respectively
*/
int text_box::input(int a)
{
    coord c = getpos();
    setoffset(a);
    c.y += a;
    gotoxy(c.x, c.y);

    const char *string = getstr();
    char str[100];
    strcpy(str, string);

    string_node *head = new string_node,
                *current = head;

    int len = strlen(str);
    string_node *temp_prev = NULL;
    for(int i = 0; i < len ; i++)
    {
        current->data = str[i];
        current->next = new string_node;
        current->prev = temp_prev;
        temp_prev = current;
        current = current->next;
    }

    //At the end is a box with \0
    current->data = '\0';
    current->prev = temp_prev;
    current = head;

    int state_to_return = -1;

    while(1)
    {
        if(kbhit())
        {
            char ch = interactive::getkey();

            switch((int)ch)
            {
                case TAB :
                case ENTER :
                    state_to_return = GOTONEXT;
                    goto convert_to_str;
                case BACKSPACE :
                    if(current)
                    {
                        if(!current->prev)  break; //No character to be deleted

                        string_node *node_to_delete = current->prev;

                        if(node_to_delete->prev) node_to_delete->prev->next = current;
                        else                     head = current; //If the node to be deleted is the head

                        current->prev = node_to_delete->prev;

                        delete node_to_delete;

                        gotoxy(wherex() - 1, wherey());

                        print_str(head);
                    }
                    break;
                case DELETE:
                    if(current)
                    {
                        if(current->data == '\0') break; //No character to be deleted

                        string_node *node_to_delete = current;

                        if(current->prev) current->prev->next = current->next;
                        else              head = current->next;

                        if(current->next) current->next->prev = current->prev;

                        current = current->next;
                        delete node_to_delete;

                        print_str(head);

                    }
                    break;
                case HOME:
                    gotoxy(c.x, c.y);
                    current = head;
                    break;
                case END:
                    while(current->next)
                    {
                        current = current->next;
                        gotoxy(wherex()+1, wherey());
                    }
                    break;
                case SHIFT_BACKSPACE:
                    state_to_return = BACK;
                    goto convert_to_str;
                case SHIFT_TAB:
                    state_to_return = GOTOPREV;
                    goto convert_to_str;
                case UP:
                    state_to_return = GOTOPREV;
                    goto convert_to_str;
                case DOWN:
                    state_to_return = GOTONEXT;
                    goto convert_to_str;
                case LEFT:
                    if(current->prev)
                    {
                        current = current->prev;
                        gotoxy(wherex()-1, wherey());
                    }
                    break;
                case RIGHT: //Right arrow key
                    if(current->next)
                    {
                        current = current->next;
                        gotoxy(wherex()+1, wherey());
                    }
                    break;
                default:
                    if(isprint(ch))
                    {
                        /*
                        * When a new node is to be added, it is added behind
                        * the current node
                        */

                        string_node *new_node = new string_node;
                        new_node->data = ch;
                        new_node->next = current;
                        new_node->prev = current->prev;

                        if(current->prev) current->prev->next = new_node;
                        else              head = new_node;
                        current->prev = new_node;

                        gotoxy(wherex()+1, wherey());

                        print_str(head);
                    }
            }
        }
    }

    convert_to_str:
    {
        char a[100]; int insert_pointer = 0;
        for(current = head; current; current = current->next)
        {
            a[insert_pointer] = current->data;
            insert_pointer++;
        }

        setstr(a);

        //Deleting the list
        current = head;
        head = NULL;
        while(current)
        {
            string_node *temp = current->next;
            delete current;
            current = temp;
        }

        return state_to_return;
    }

}

/*
* Prints the string as represented by a doubly
* linked list whose head is pointed to by the
* parameter.
*/
void text_box::print_str(string_node *head)
{
    coord init = coord(wherex(), wherey());
    coord c = getpos();
    gotoxy(c.x, c.y + getoffset());
    textcolor(gettcolor());
    textbackground(getbcolor());
    for(string_node *current = head; current; current = current->next)
    {
        if(is_password)
        {
            if(current->data != '\0')
            {
                cprintf("*");
            }
            else
            {
                cprintf(" ");
            }
        }
        else            cprintf("%c", current->data);
    }
    gotoxy(init.x, init.y);
}

void text_box::setis_password(int a)
{
    is_password = a;
}
```

19. code/UI/infotbox.cpp

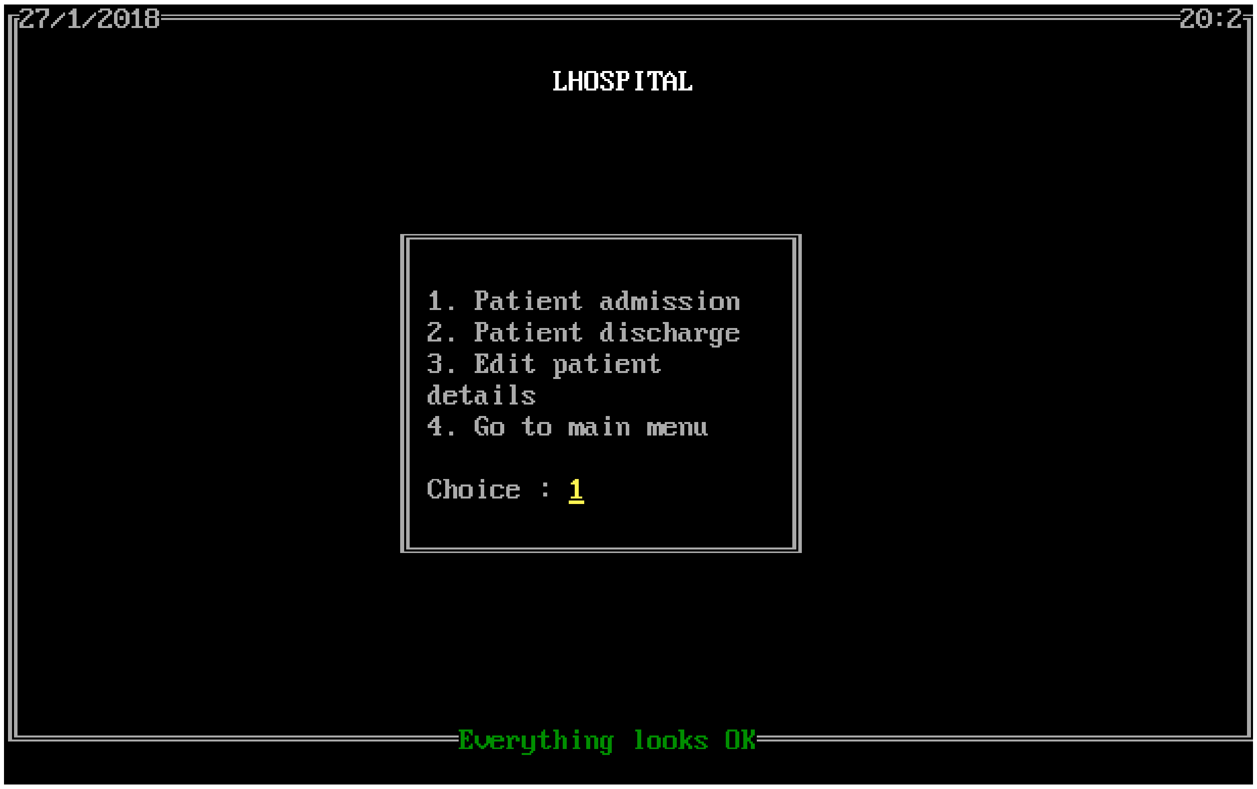
```
#include "ui/ui.hpp"
#include "iface.hpp"

info_tbox::info_tbox()
{
    tbox = NULL;
    data_store = NULL;
    type = OTHER;
    validator = NULL;
}

int info_tbox::setdata()
{
    if(validator(tbox->getstr()) == 0)
    {
        return 0;
    }

    char *fstr;
    switch(type)
    {
        case INT:
        {
            fstr = "%d";
            break;
        }
        case LONG:
        {
            fstr = "%ld";
            break;
        }
        case UNSIGNED_LONG:
        {
            fstr = "%lu";
            break;
        }
        case STRING:
        case PASSWORD:
        {
            char *s = (char *) data_store;
            strcpy(s, tbox->getstr());
            return 1;
        }
        case CHAR:
        {
            fstr = "%c";
            break;
        }
        case DOUBLE:
        {
            fstr = "%g";
            break;
        }
        case FLOAT:
        {
            fstr = "%f";
            break;
        }
        default:
            return 0;
    }

    sscanf(tbox->getstr(), fstr, data_store);

    return 1;
}
```

Data files

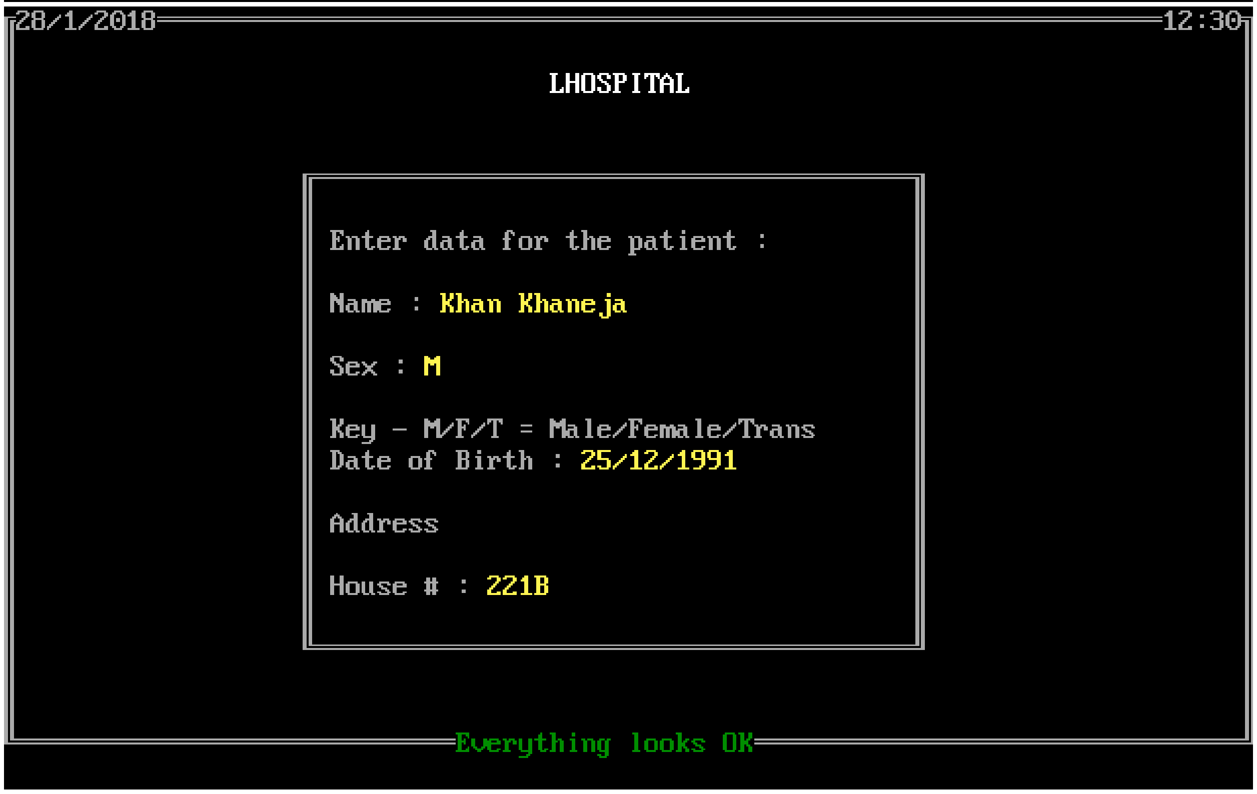
1. code/TRANSACT.DAT
2. code/PROC.DAT
3. code/PATIENT/MAXID.DAT
4. code/PATIENT/5/TRANS.DAT
5. code/PATIENT/5/BASE.DAT
6. code/PATIENT/1/BASE.DAT
7. code/PATIENT/3/BASE.DAT
8. code/PATIENT/14/TRANS.DAT
9. code/PATIENT/14/BASE.DAT
10. code/PATIENT/12/BASE.DAT
11. code/PATIENT/2/BASE.DAT
12. code/PATIENT/7/BASE.DAT
13. code/PATIENT/0/BASE.DAT
14. code/PATIENT/8/TRANS.DAT
15. code/PATIENT/8/BASE.DAT
16. code/PATIENT/13/TRANS.DAT
17. code/PATIENT/13/BASE.DAT
18. code/PATIENT/11/TRANS.DAT
19. code/PATIENT/11/BASE.DAT
20. code/PATIENT/15/BASE.DAT
21. code/PATIENT/9/TRANS.DAT
22. code/PATIENT/9/BASE.DAT
23. code/PATIENT/6/TRANS.DAT
24. code/PATIENT/6/BASE.DAT
25. code/PATIENT/10/BASE.DAT
26. code/PATIENT/4/BASE.DAT
27. code/EMPLOYEE/IDLIST.DAT
28. code/EMPLOYEE/MAXID.DAT

29. code/EMPLOYEE/1/TRANS.DAT
30. code/EMPLOYEE/1/BASE.DAT
31. code/EMPLOYEE/RECEPTIO/5/TRANS.DAT
32. code/EMPLOYEE/RECEPTIO/5/BASE.DAT
33. code/EMPLOYEE/DOCTOR/2/TRANS.DAT
34. code/EMPLOYEE/DOCTOR/2/BASE.DAT
35. code/EMPLOYEE/DOCTOR/7/TRANS.DAT
36. code/EMPLOYEE/DOCTOR/7/BASE.DAT
37. code/EMPLOYEE/6/TRANS.DAT
38. code/EMPLOYEE/6/BASE.DAT
39. code/EMPLOYEE/NURSE/3/TRANS.DAT
40. code/EMPLOYEE/NURSE/3/BASE.DAT
41. code/STOCK/MED.DAT
42. code/STOCK/MEDICINE.DAT

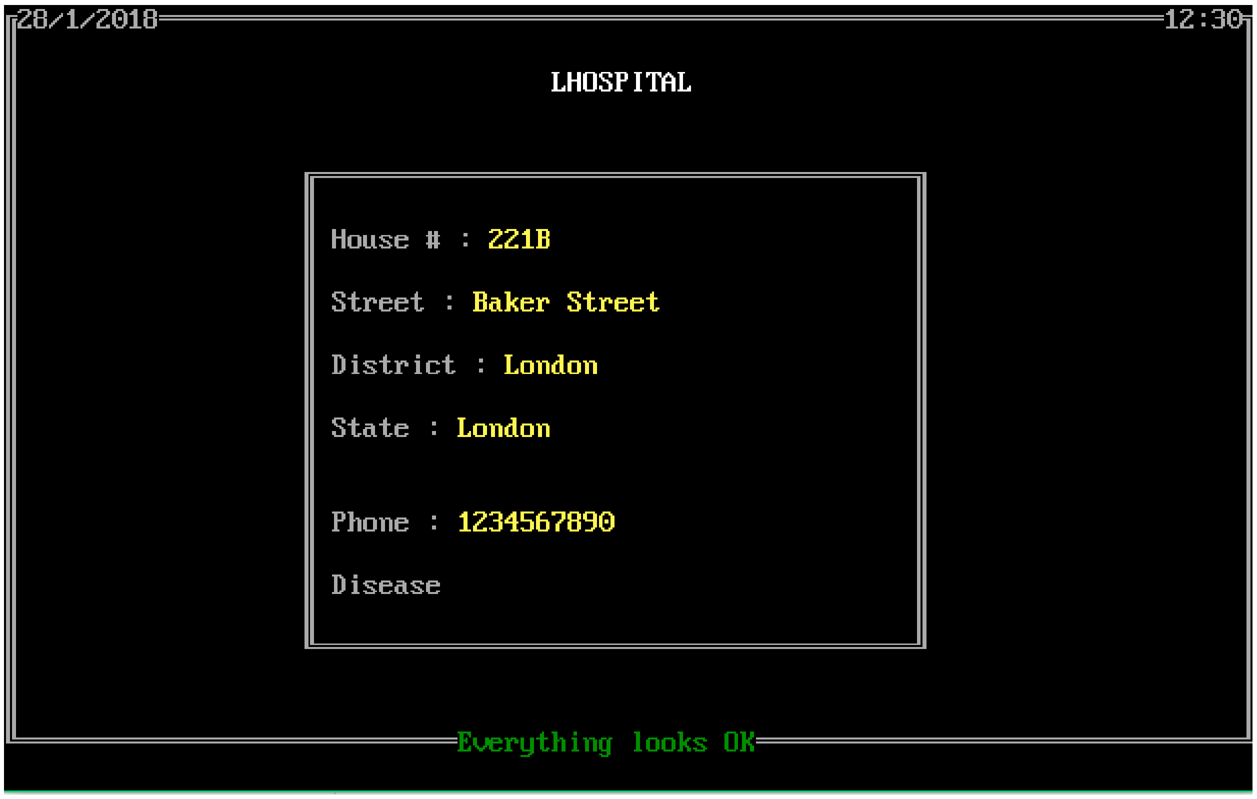
Output



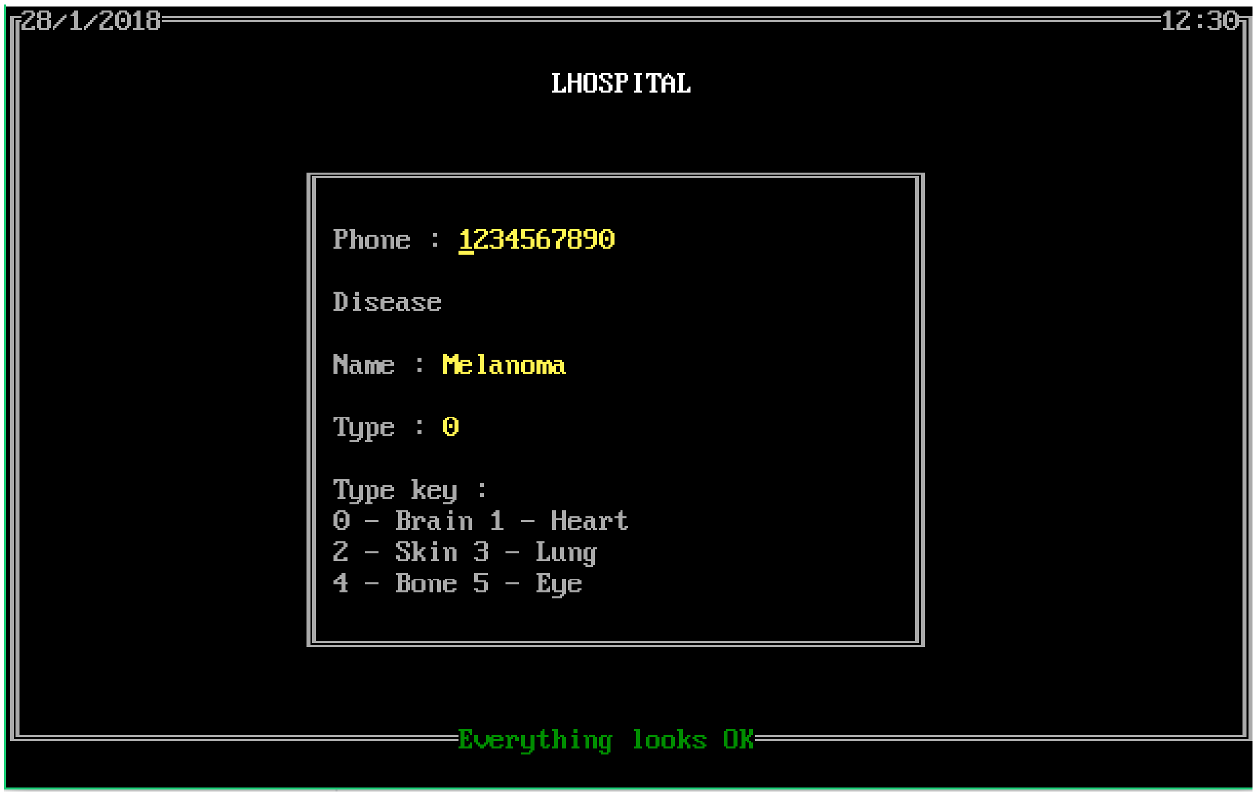
```
27/1/2018                                                20:2

                         LHOSPITAL

                1. Patient admission
                2. Patient discharge
                3. Edit patient
                details
                4. Go to main menu

                Choice : 1

                    Everything looks OK
```



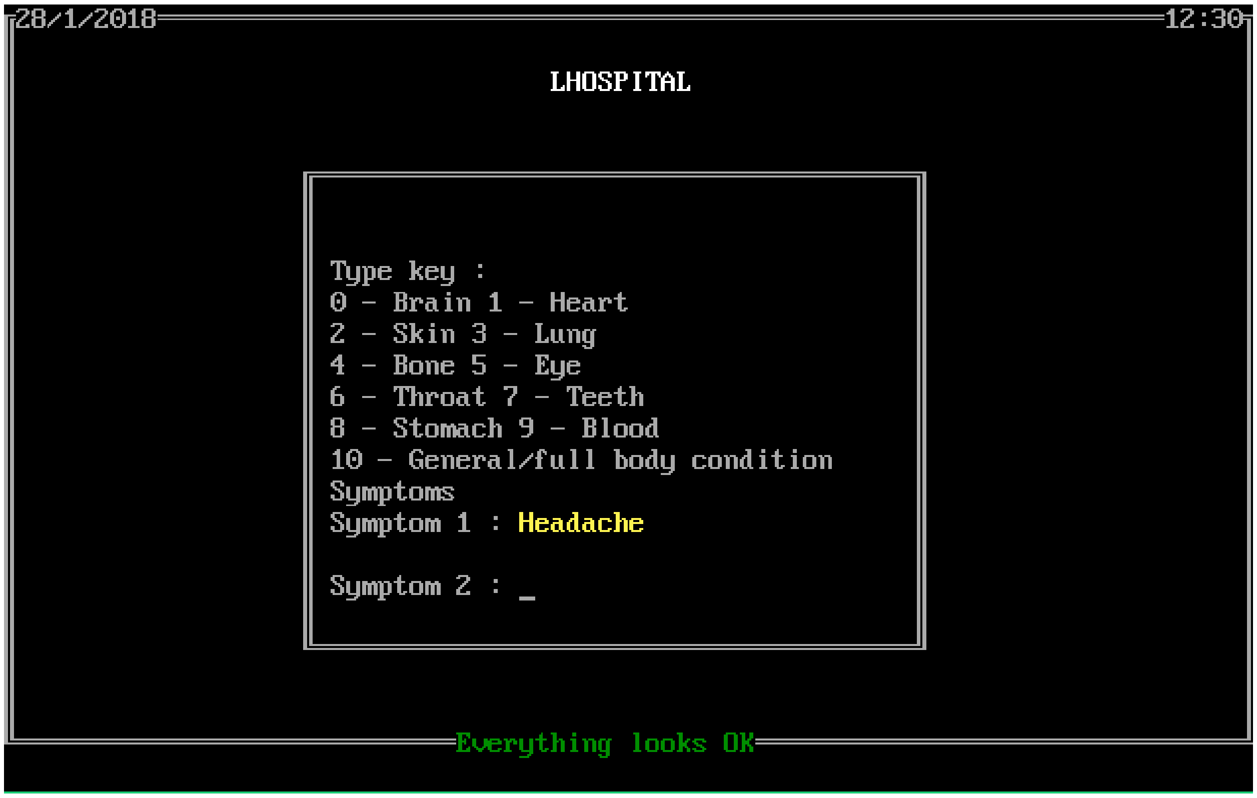
```
28/1/2018                                                12:30

                         LHOSPITAL

            Enter data for the patient :

            Name : Khan Khaneja

            Sex : M

            Key - M/F/T = Male/Female/Trans
            Date of Birth : 25/12/1991

            Address

            House # : 221B

                    Everything looks OK
```



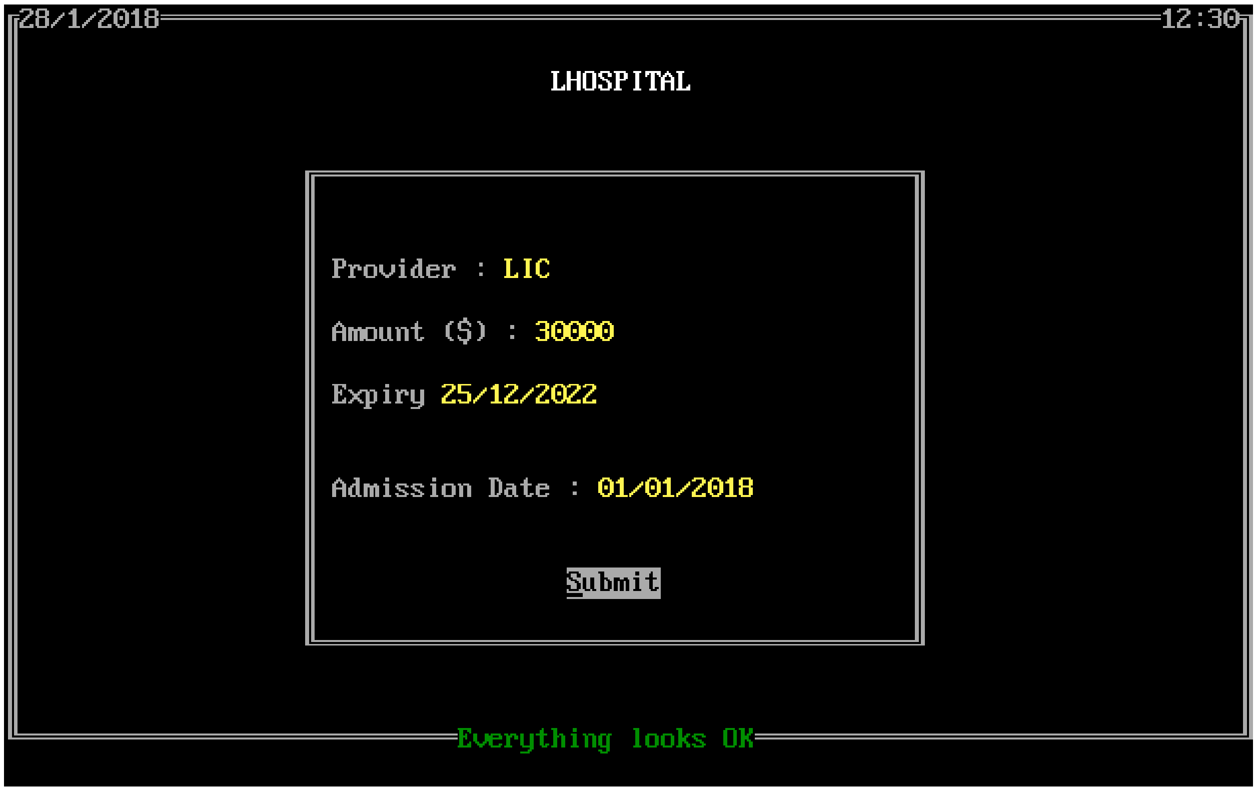
```
28/1/2018                                                              12:30

                              LHOSPITAL

         House # : 221B

         Street : Baker Street

         District : London

         State : London


         Phone : 1234567890

         Disease

                        Everything looks OK
```



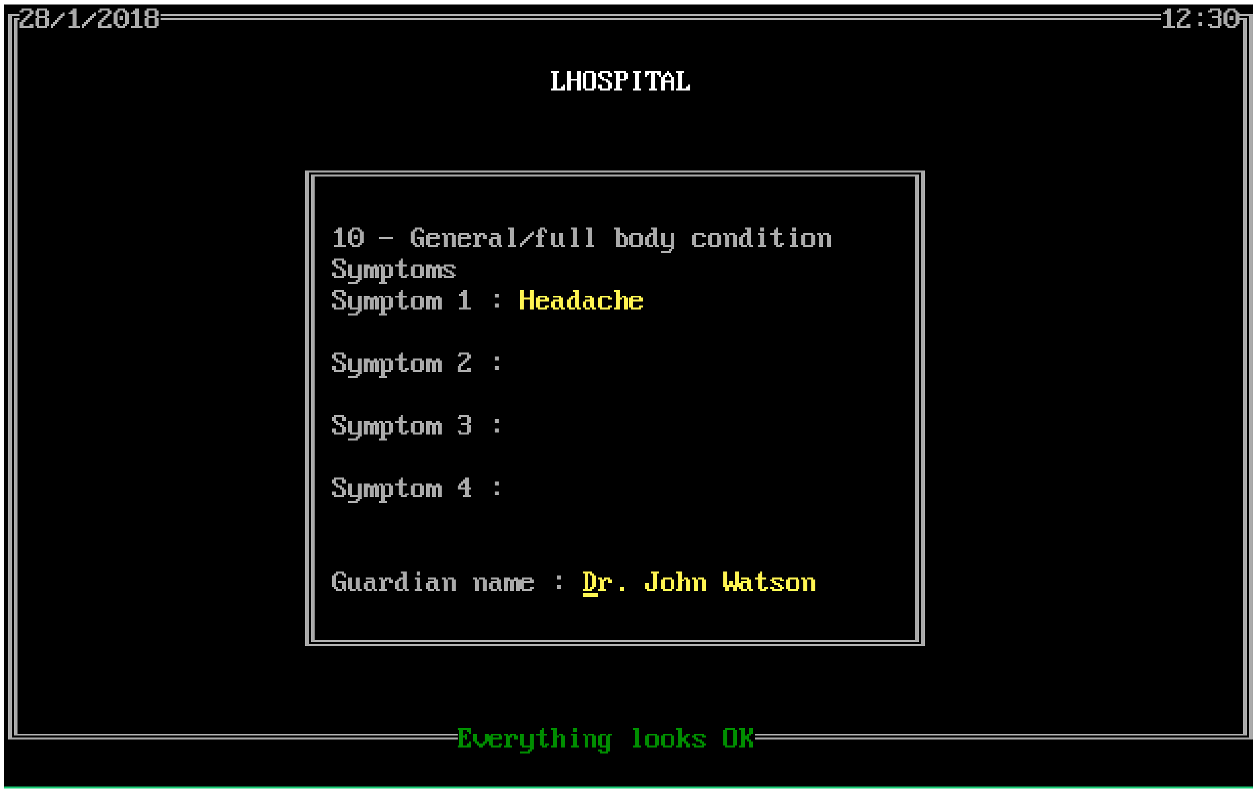
```
28/1/2018                                                              12:30

                              LHOSPITAL

         Phone : 1234567890

         Disease

         Name : Melanoma

         Type : 0

         Type key :
         0 - Brain 1 - Heart
         2 - Skin 3 - Lung
         4 - Bone 5 - Eye

                        Everything looks OK
```



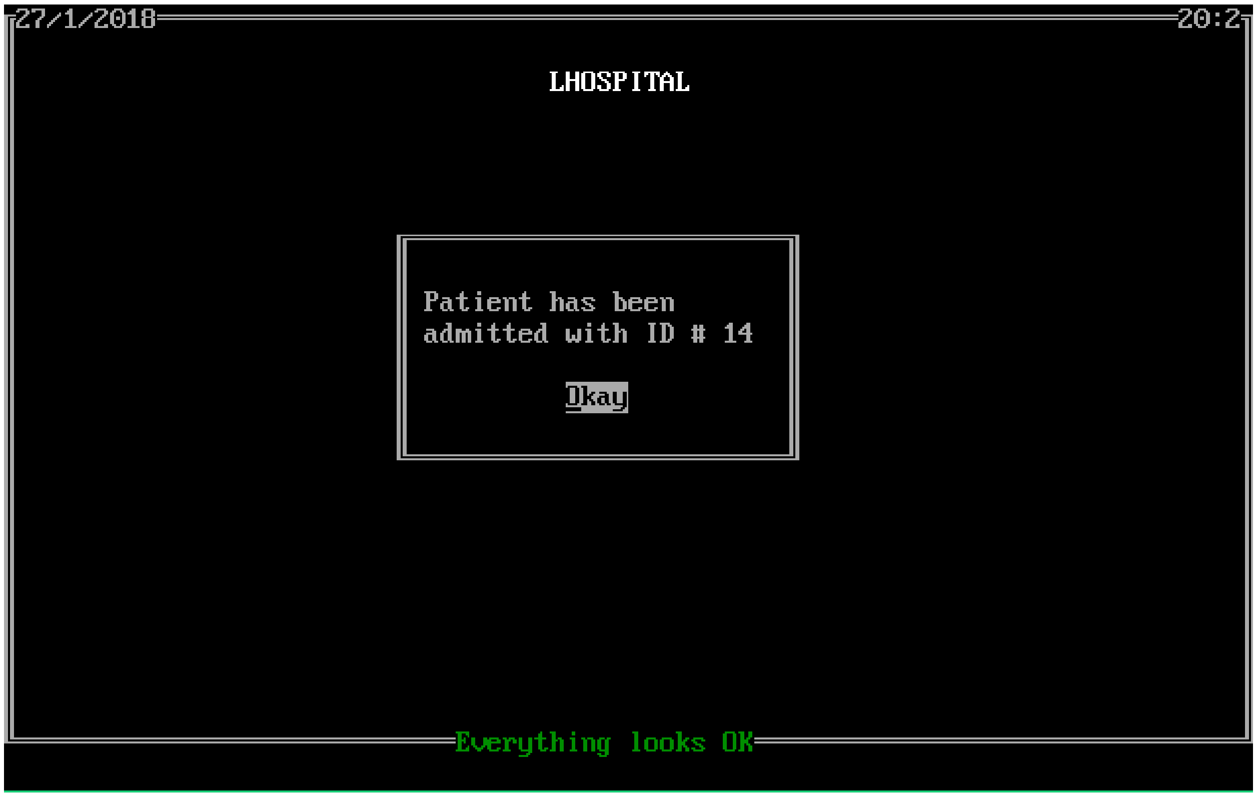
```
28/1/2018                                                    12:30

                          LHOSPITAL

              Type key :
              0 - Brain 1 - Heart
              2 - Skin 3 - Lung
              4 - Bone 5 - Eye
              6 - Throat 7 - Teeth
              8 - Stomach 9 - Blood
              10 - General/full body condition
              Symptoms
              Symptom 1 : Headache

              Symptom 2 : _

                    Everything looks OK
```



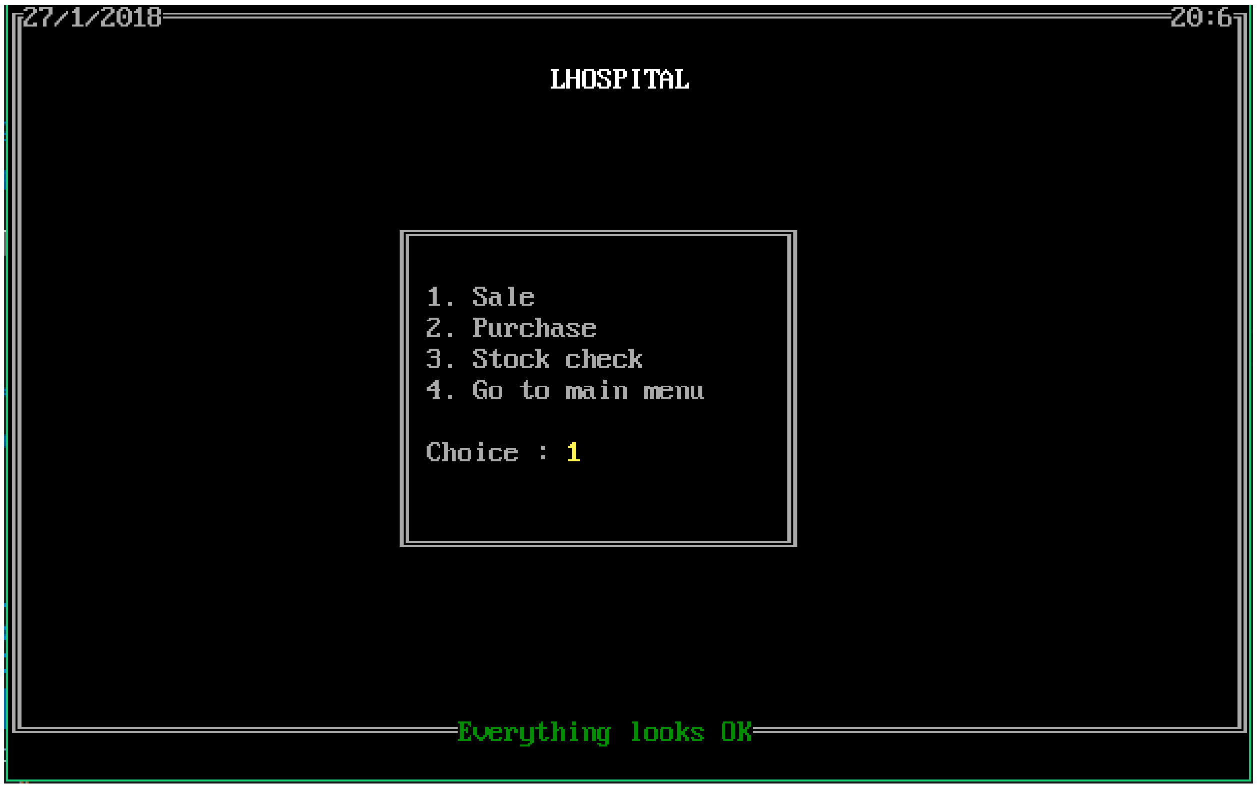
```
28/1/2018                                                    12:30

                          LHOSPITAL

              Provider : LIC

              Amount ($) : 30000

              Expiry 25/12/2022


              Admission Date : 01/01/2018


                          Submit

                    Everything looks OK
```



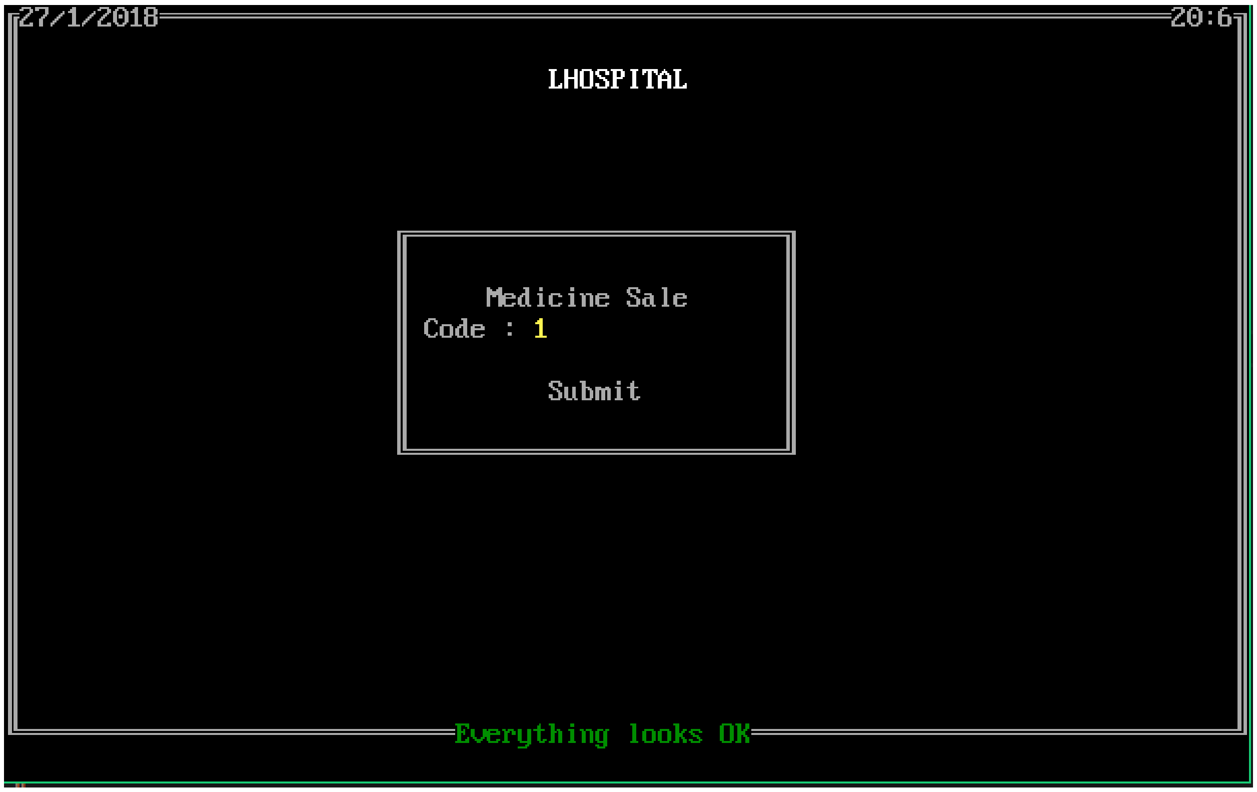
```
28/1/2018                                                  12:30

                         LHOSPITAL

          10 - General/full body condition
          Symptoms
          Symptom 1 : Headache

          Symptom 2 :

          Symptom 3 :

          Symptom 4 :


          Guardian name : Dr. John Watson

                    Everything looks OK
```



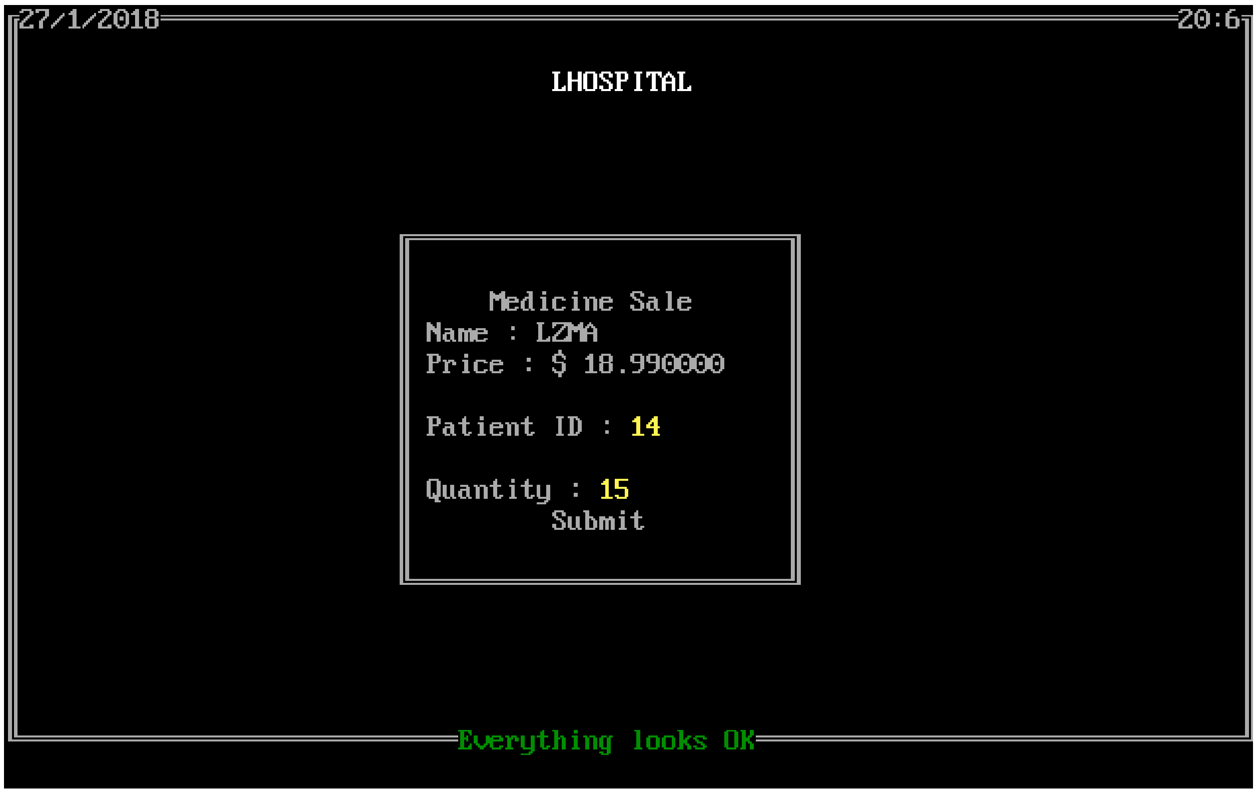
```
27/1/2018                                                  20:2

                         LHOSPITAL



              Patient has been
              admitted with ID # 14

                    Okay

                    Everything looks OK
```



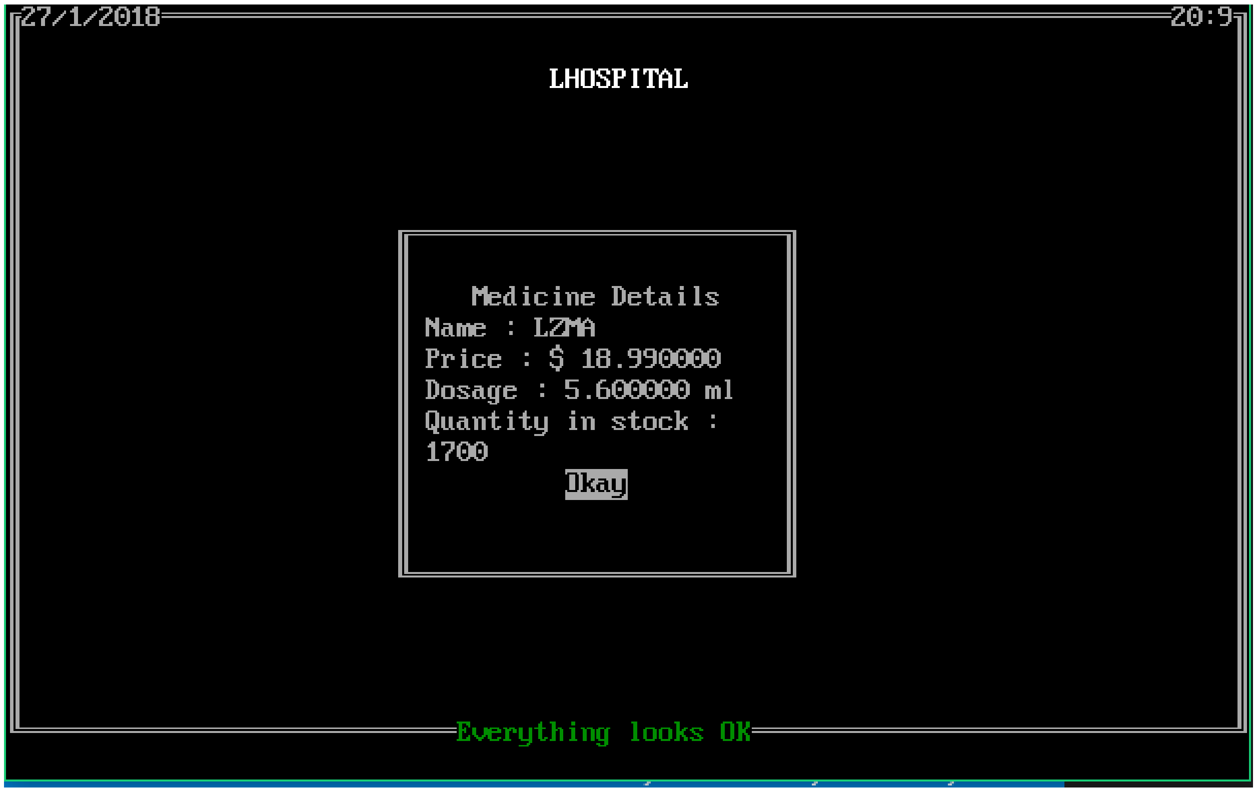
```
27/1/2018                                                        20:6

                         LHOSPITAL

                   1. Sale
                   2. Purchase
                   3. Stock check
                   4. Go to main menu

                   Choice : 1

                   Everything looks OK
```



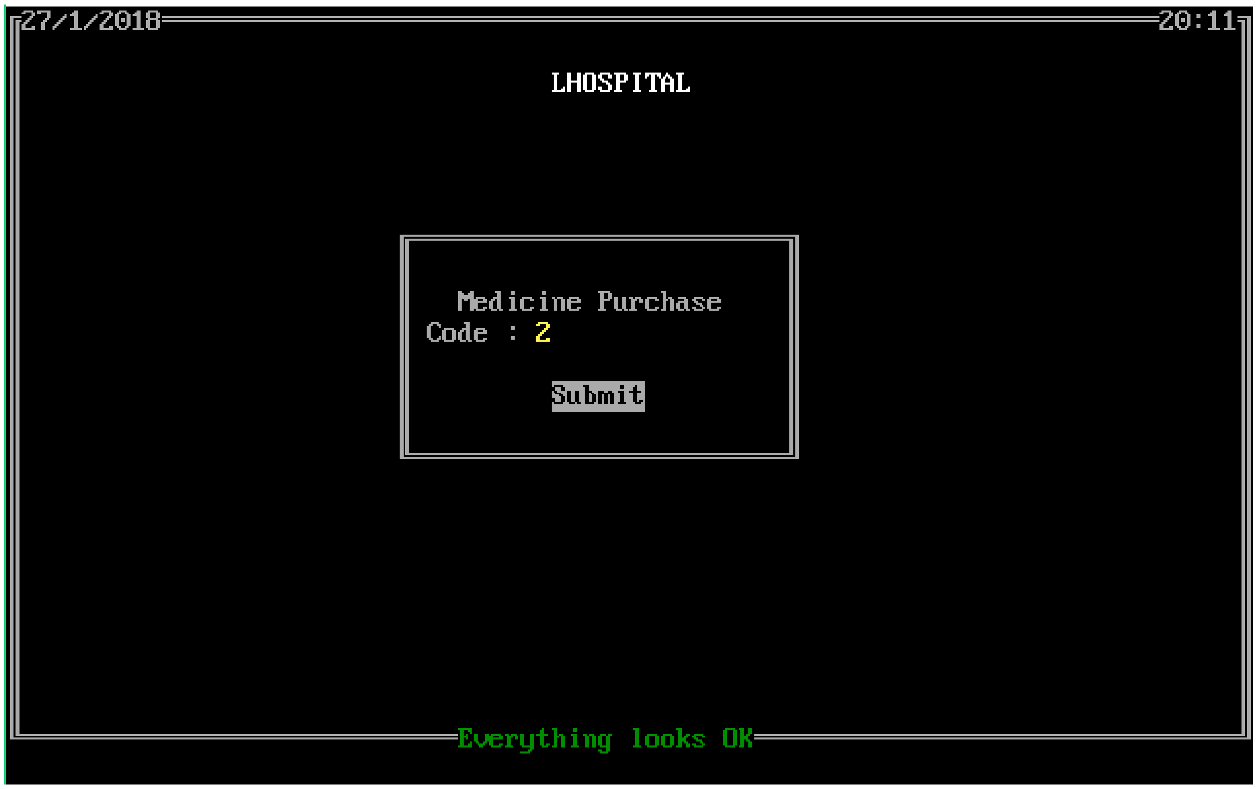
```
27/1/2018                                                        20:6

                         LHOSPITAL

                      Medicine Sale
                   Code : 1

                          Submit

                   Everything looks OK
```



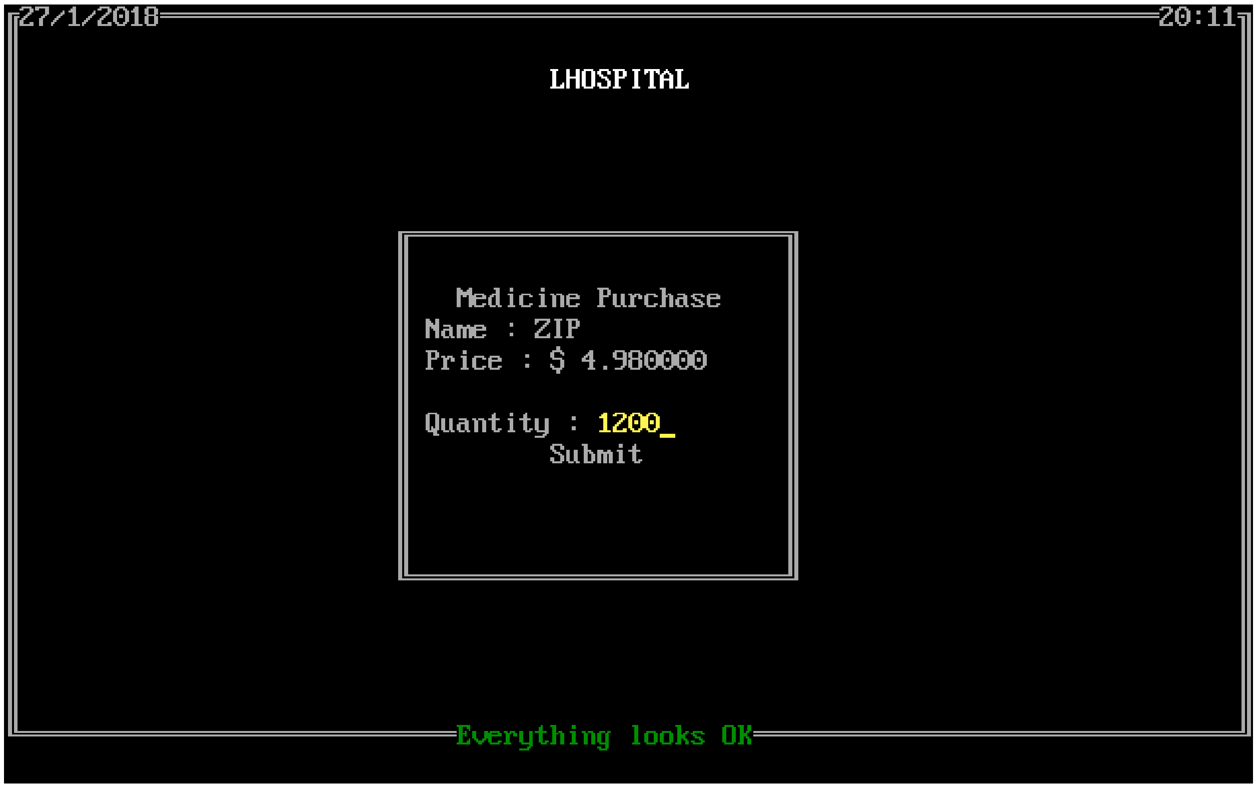
```
27/1/2018                                                        20:6

                         LHOSPITAL



                  Medicine Sale
                Name : LZMA
                Price : $ 18.990000

                Patient ID : 14

                Quantity : 15
                        Submit



                   Everything looks OK
```



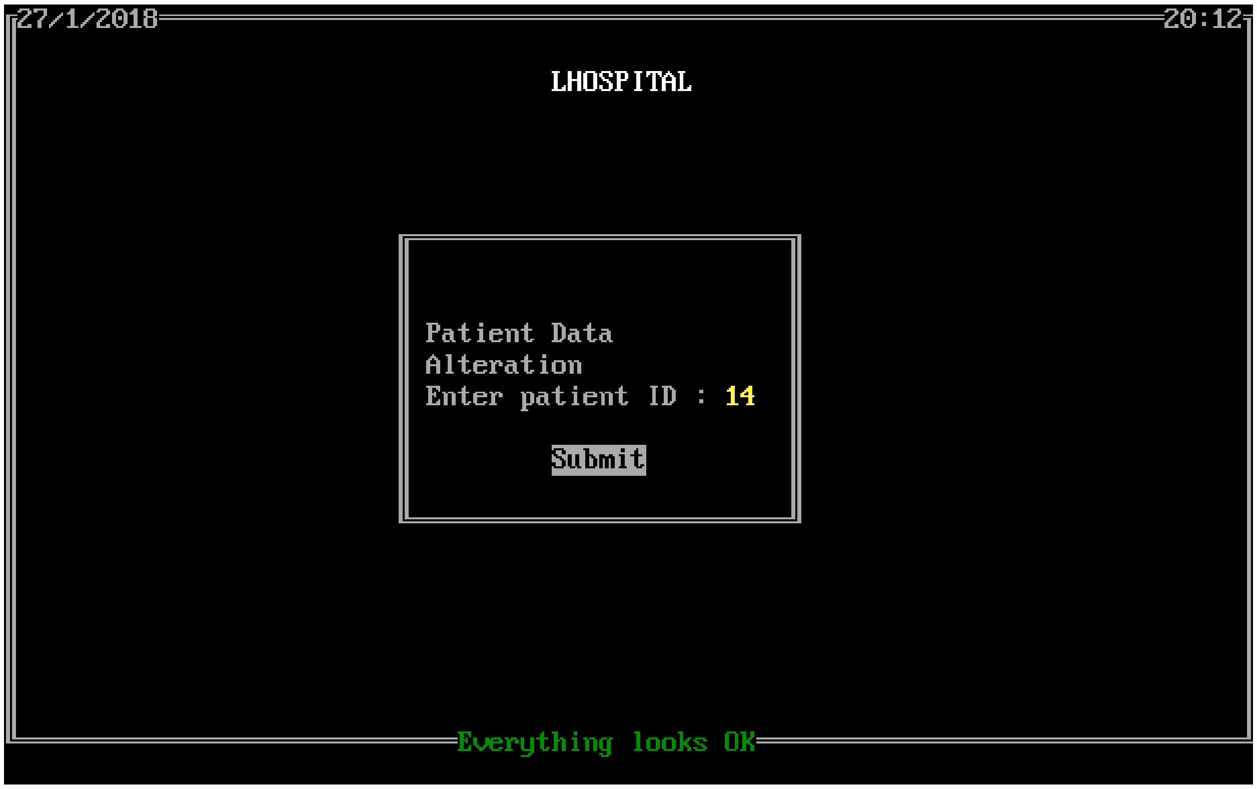
```
27/1/2018                                                        20:9

                         LHOSPITAL



                 Medicine Details
                Name : LZMA
                Price : $ 18.990000
                Dosage : 5.600000 ml
                Quantity in stock :
                1700
                        Okay



                   Everything looks OK
```



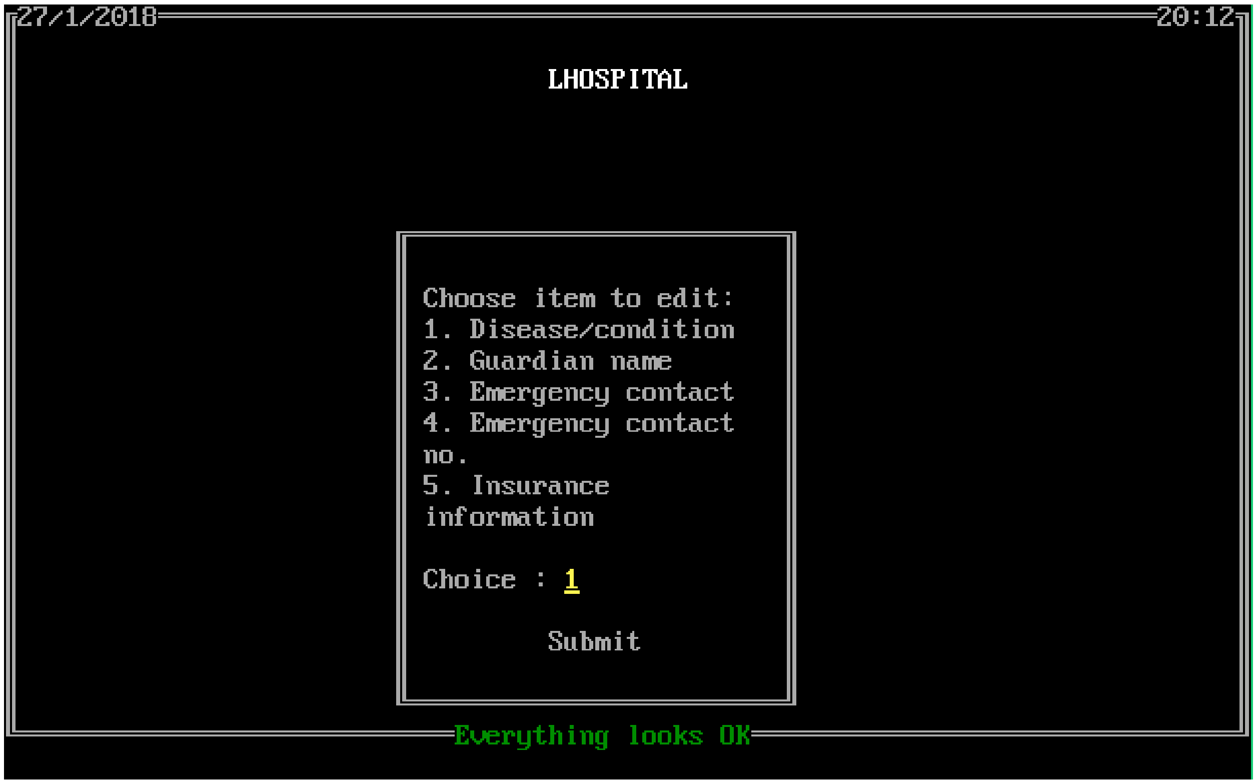
```
27/1/2018                                                        20:11

                              LHOSPITAL



                         Medicine Purchase
                        Code : 2

                               Submit



                          Everything looks OK
```



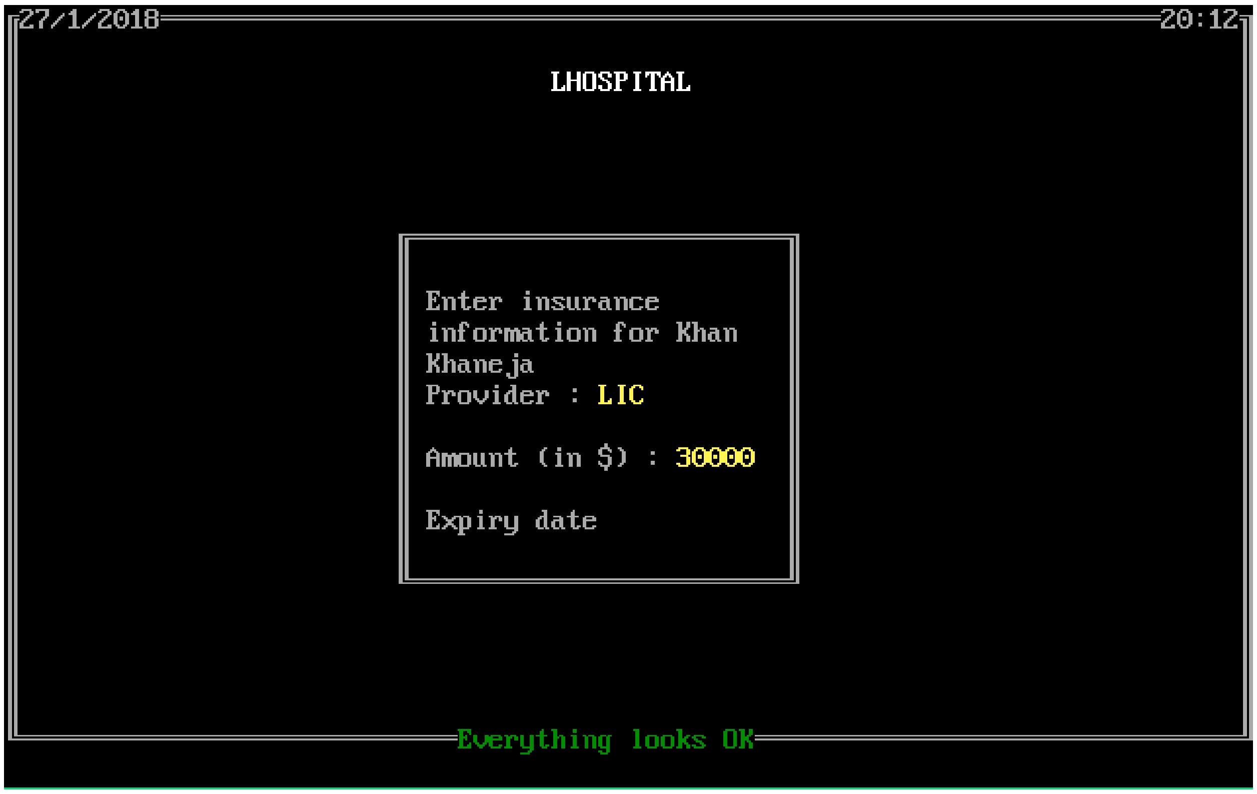
```
27/1/2018                                                        20:11

                              LHOSPITAL



                         Medicine Purchase
                        Name : ZIP
                        Price : $ 4.980000

                        Quantity : 1200_
                               Submit



                          Everything looks OK
```



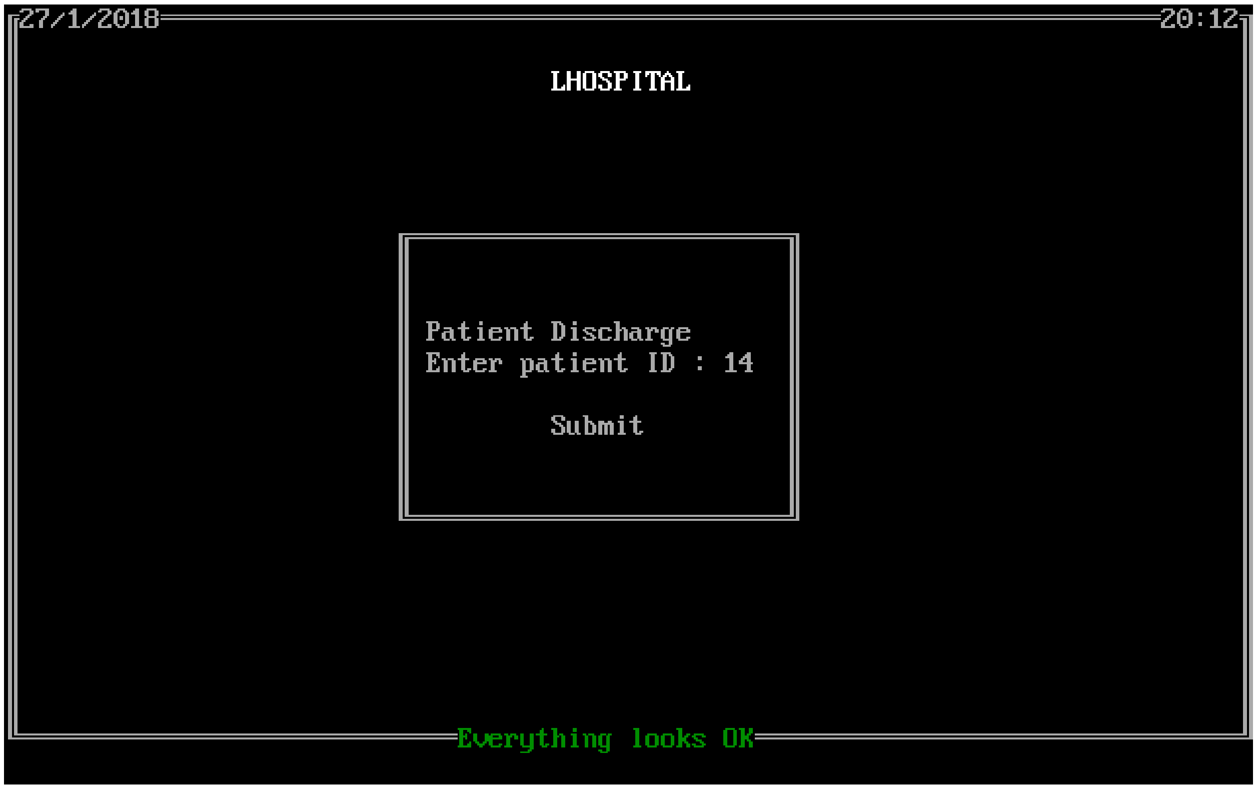
```
27/1/2018                                                    20:12

                          LHOSPITAL

                  Patient Data
                  Alteration
                  Enter patient ID : 14

                        Submit

                  Everything looks OK
```



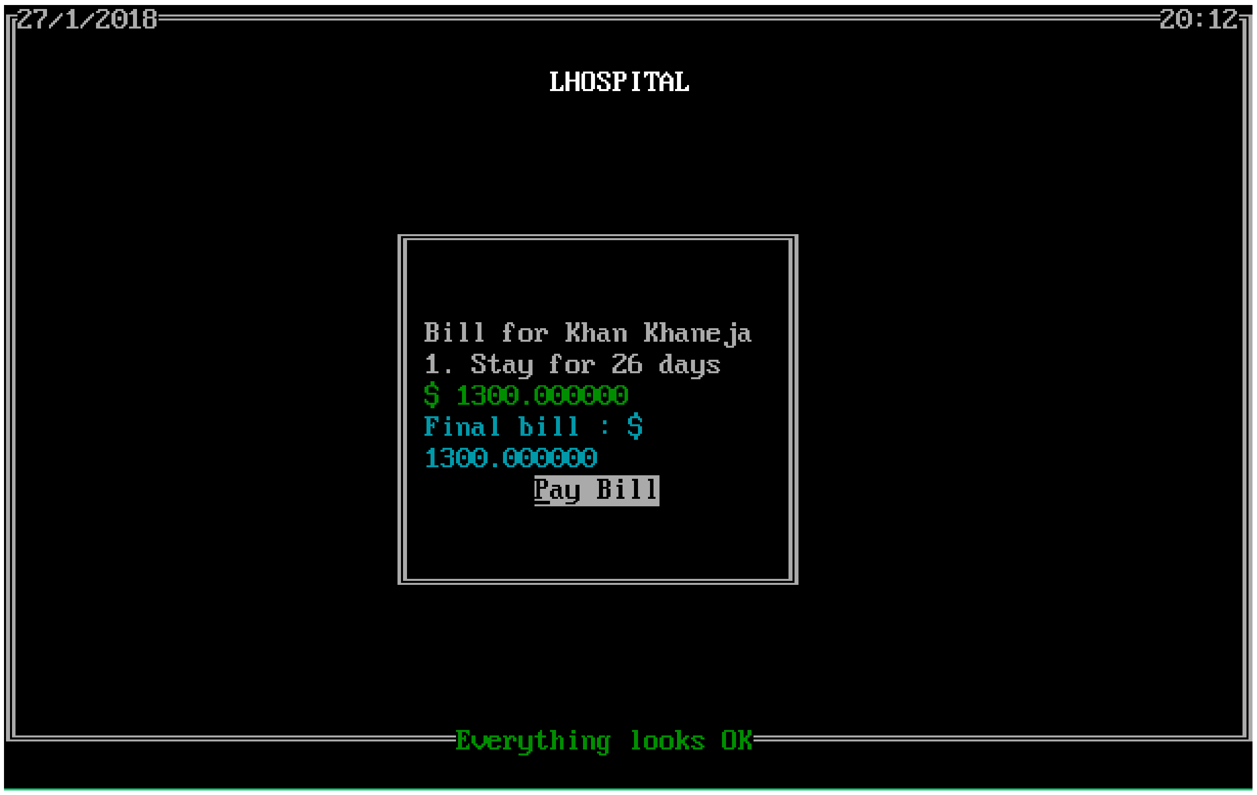
```
27/1/2018                                                    20:12

                          LHOSPITAL

                  Choose item to edit:
                  1. Disease/condition
                  2. Guardian name
                  3. Emergency contact
                  4. Emergency contact
                  no.
                  5. Insurance
                  information

                  Choice : 1

                        Submit

                  Everything looks OK
```



```
27/1/2018                                                     20:12

                              LHOSPITAL


                    Enter insurance
                    information for Khan
                    Khaneja
                    Provider : LIC

                    Amount (in $) : 30000

                    Expiry date


                       Everything looks OK
```



```
27/1/2018                                                     20:12

                              LHOSPITAL


                    Patient Discharge
                    Enter patient ID : 14

                            Submit


                       Everything looks OK
```



```
27/1/2018                                                  20:12
                          LHOSPITAL

                Bill for Khan Khaneja
                1. Stay for 26 days
                $ 1300.000000
                Final bill : $
                1300.000000
                       Pay Bill

                    Everything looks OK
```